Elektrobit

EB tresos® AutoCore OS documentation

product release 6.0

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

Europe	Japan	USA
Phone: +49 9131 7701 6060	Phone: +81 3 5577 6110	Phone: +1 888 346 3813

Support URL

https://www.elektrobit.com/support

Legal notice

Confidential and proprietary information

Table of Contents

Begin here

1. If you are upgrading from a previous version

- What's new in this EB tresos AutoCore OS version?

  See the document EB tresos AutoCore OS release notes, located in your EB tresos AutoCore OS <INSTALL_PATH>/doc directory.

- Which known problems, fixed problems, incompatibilities to previous releases, limitations and restrictions have been reported for the current EB tresos AutoCore OS version?

  See the EB tresos AutoCore known problems, which you may download from the download server *EB Command*. The link to EB Command as well as your user login and password were sent to you via email.

- How do I migrate from an older version to the current version of EB tresos AutoCore OS?

  See the EB tresos AutoCore OS release notes, located in your EB tresos AutoCore OS <INSTALL_PATH>/doc directory.

2. If you are using EB tresos AutoCore OS for the first time

- If you are a first time user of EB tresos AutoCore OS, you may want to get familiar with some of the concepts behind the AUTOSAR `Os` module and the OSEK operating system at Chapter 3, “Background information“.

- Where can I find an example?

  Chapter 4, “Application example“ gives you an example of an EB tresos AutoCore OS project along with instructions you may follow for practice.

3. If you want to know how to work with EB tresos AutoCore OS

In Section 5.2.1, “Development workflow” you find a description of the typical workflow when you work with EB tresos AutoCore OS. Instructions for performing these single steps are available in the user's guide, arranged in the order of the workflow:

- Configuring the `Os` module (Section 5.3, "Configuring and using `Os` objects").
- Verifying and generating the `Os` (Section 5.4, "Generating the code of the `Os` module").
- Generating a linker script (Section 5.5, "Creating a linker script").

4. If you are an advanced user

- If you want to import your OIL or EPC files into EB tresos Studio, see Section 5.6.1, "Importing OIL/EPC files".
- If you want to optimize the code of your module, see Section 5.6.2, "Optimizing your `Os` module".
- If you want to build the `Os` with your own build environment, see Section 5.6.3, "Compiling EB tresos AutoCore OS in custom build environments".

5. If you need help/further information

- Chapter 2, "EB tresos product line support"

  Receive technical support via email or phone from the Chapter 2, "EB tresos product line support" hotline.
- Chapter 1, "About this documentation"

  Find out about:

  - Typographical and style conventions used throughout this documentation. Defines usage of special fonts in the documentation.
- Index "Index"

  You cannot find what you are looking for? Try the Index "Index" list (alphabetically sorted).
- "Bibliography"

  Would you like to read more detailed information? Find bibliographic references and further reading suggestions in the "Bibliography".

1. About this documentation

1.1. Typography and style conventions

Throughout the documentation you see that words and phrases are displayed in bold or italic font, or in Monospace font. To find out what these conventions mean, consult the following table. All default text is written in Arial Regular font without any markup.

Convention	Item is used	Example
Arial italics	to define new terms	The *basic building blocks* of a configuration are module configurations.
Arial italics	to emphasize	If your project's release version is mixed, all content types are available. It is thus called *mixed version*.
Arial italics	to indicate that a term is explained in the glossary	...exchanges *protocol data unit*s (*PDU*s) with its peer instance of other ECUs.
Arial boldface	for menus and submenus	Choose the **Options** menu.
Arial boldface	for buttons	Select **OK**.
Arial boldface	for keyboard keys	Press the **Enter** key
Arial boldface	for keyboard combination of keys	Press **Ctrl**+**Alt**+**Delete**
Arial boldface	for commands	Convert the XDM file to the newer version by using the **legacy convert** command.
Monospace font (Courier)	for file and folder names, also for chapter names	Put your script in the `function_name/abc-folder`
Monospace font (Courier)	for code	`for (i=0; i<5; i++) { /* now use i */ }`
Monospace font (Courier)	for function names, methods, or routines	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Monospace font (Courier)	for user input/indicates variable text	Enter a `three-digit prefix` in the menu line.
Square brackets [ ]	for optional parameters; for command syntax with optional parameters	`insertBefore [<opt>]`

Convention	Item is used	Example
Curly brackets {}	for mandatory parameters; for command syntax with mandatory parameters (in curly brackets)	insertBefore {<file>}
Three dots …	for further parameters	insertBefore [<opt>…]
A vertical bar |	to separate parameters in a list from which one parameters must be chosen or used; for command syntax, indicates a choice of parameters	allowinvalidmarkup {on|off}
Warning	to show information vital for the success of your configuration	**WARNING** **This is a warning** This is what a warning looks like.
Notice	to give additional important information on the subject	**NOTE** **This is a notice** This is what a notice looks like.
Tip	to provide helpful hints and tips	**TIP** **This is a tip** This is what a tip looks like.

2. EB tresos product line support

Europe

Phone: +49 9131 7701 6060

Japan

Phone: +81 3 5577 6110

USA

Phone: +1 888 346 3813

Support URL

https://www.elektrobit.com/support

3. Background information

The EB tresos AutoCore OS kernel represents an implementation of the AUTOSAR Os standard operating system kernel which in turn is an extension of the OSEK/VDX-OS standard.

The purpose of this chapter is to give an overview of the EB tresos AutoCore OS kernel. It describes the concepts of OSEK/VDX and the AUTOSAR Os but is not intended to replace the official OSEK or AUTOSAR specifications. This chapter discusses the multitasking kernel, tasking facilities, inter-task communication, and interrupt handling facilities, which are at the heart of the EB tresos AutoCore OS kernel.

Modern real-time systems are based on the complementary concepts of multitasking and inter-task communications. A multitasking environment allows a real-time system to be constructed as a set of independent tasks, each with its own thread of execution and set of system resources. The inter-task communication facilities allow these tasks to synchronize and communicate in order to coordinate their activities. In EB tresos AutoCore OS, the inter-task communication facilities include resources and events.

Another key facility in real-time systems is hardware interrupt handling, because the interrupt mechanism is commonly used to inform a system of external events. To get the fastest possible response to interrupts, *interrupt service routines* (ISRs) in EB tresos AutoCore OS run in a special context of their own outside the context of a task. Counters and alarms provide additional support for recurring events.

As each target architecture is different, this may place certain constraints on an AUTOSAR operating system. Such architecture-specific details are discussed in a supplement to this guide: EB tresos AutoCore OS architecture notes. See the appropriate supplement for the target system that you are using.

3.1. The OS in the AUTOSAR environment

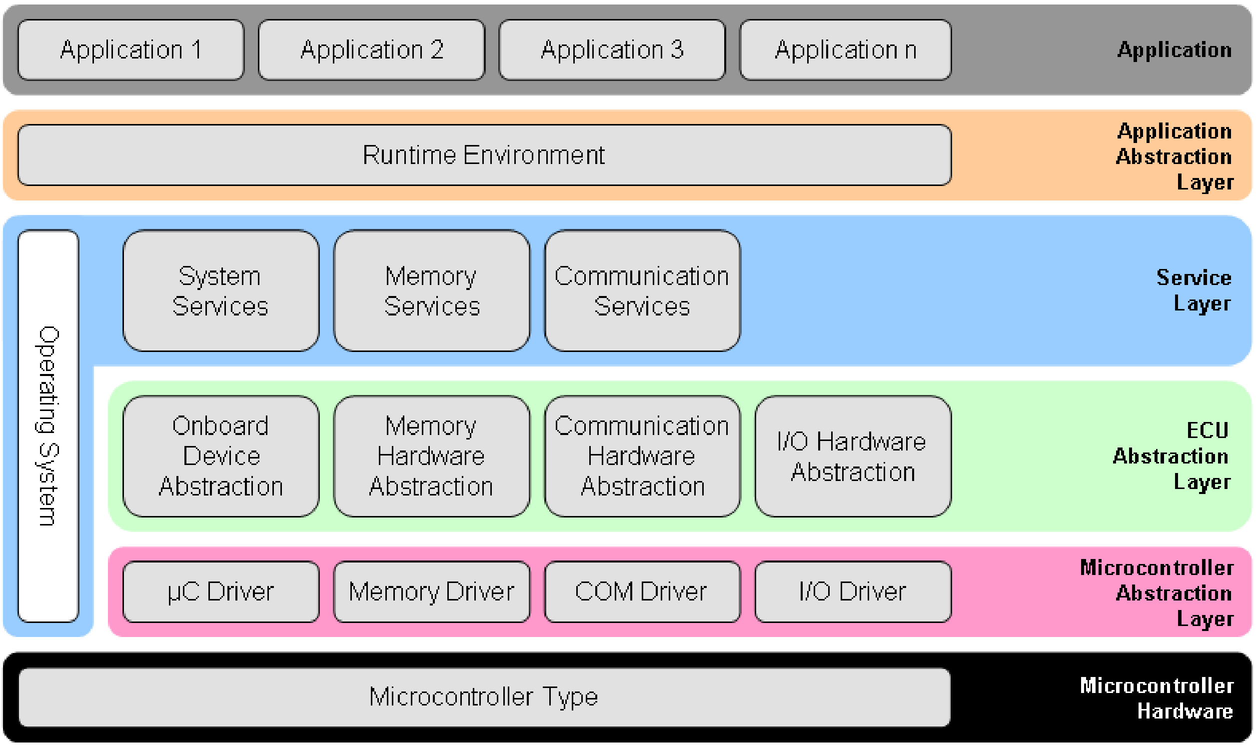


Figure 3.1. Overview of AUTOSAR software layers

Figure 3.1, "Overview of AUTOSAR software layers" depicts the layered AUTOSAR software architecture. The AUTOSAR operation system is part of the service layer and also straddles the lower two layers.

3.2. A static operating system

The AUTOSAR operating system specification is targeted at deeply embedded systems with hard real-time constraints. In such systems any non-deterministic behavior that might conflict with the real-time requirements should be avoided. Therefore the specification defines AUTOSAR Os as a static operating system. This means that system objects can never be created dynamically - the entire layout of the system, including every single object, must be determined before the system is built.

The EB tresos AutoCore OS kernel implements this static notion of an operating system, thus no system services are provided to dynamically create new objects like tasks or resources. As a consequence, the application programmer must specify the layout of their system prior to starting the system. EB tresos AutoCore OS provides a tool, the Generator, which supports the process of system configuration. The Generator includes tools which can verify a configuration, generate the configuration for a system and more. See Section 5.2.3, "The Os Generator" for a detailed discussion of this tool.

3.3. Tasks

It is often essential to organize a system into independent cooperating programs. Each of these independent programs, while executing, is called a task. In EB tresos AutoCore OS, tasks have immediate access to their own code and data, while also maintaining enough separate context to maintain individual threads of control. Tasks may also be permitted shared access to global resources, depending on the privileges assigned.

3.3.1. Multitasking

Multitasking provides the fundamental mechanism for a system to control and react to multiple, discrete real-world events. The EB tresos AutoCore OS kernel provides a basic multitasking environment. Multitasking creates the appearance of many threads running concurrently when in fact the kernel interleaves their execution on the basis of a scheduling algorithm. Each apparently independent program is called a task. Each task has its own context, which is the CPU environment and system resources that the task sees each time it is scheduled to run by the kernel. On a context switch, a task's context is saved in the appropriate save area. A task's context includes:

- a thread of execution, that is, the task's program counter
- the CPU registers
- a stack for dynamic variables and function calls
- kernel control structures

In EB tresos AutoCore OS, one important resource that is not part of a task's context is memory address space. All code executes in a single common address space, although access may be restricted to certain portions of that space depending on the task's privileges.

3.3.2. Task types and conformance classes

Task contexts are expensive resources in a lightweight operating system like OSEK. In order to provide the best optimization possible, tasks are subdivided into two categories, *basic* and *extended*, depending on their synchronization needs.

In practice, because of the requirement that EB tresos AutoCore OS shall be certifiable and deliverable as a precompiled library, there is little difference between basic and extended tasks, and the compiled code is always capable of supporting an ECC2[1] system. However, the Generator still supports the conformance classes and issues warnings or errors when a system exceeds the OSEK-specified limits, and the kernel itself checks that the system keeps within the limitations at runtime. If you are compiling the kernel from source and have chosen

[1] For details about conformance classes, see chapter Section 3.3.2.3, “Conformance classes”.

the optimized kernel option at build time, some of the possible optimisations will be made, resulting in a smaller, faster kernel.

3.3.2.1. Basic tasks

The key property of a basic task is that it does not use blocking synchronization to coordinate its activity with that of other tasks. Thus, basic tasks release the processor only if one of the following situations occurs:

- the task is terminated
- a higher priority task is scheduled
- an interrupt causes the processor to execute an interrupt service routine (ISR)

3.3.2.2. Extended tasks

The key property of an extended task is that it may use blocking synchronization. In EB tresos AutoCore OS, the only system service that may block a task is `WaitEvent()`. This service blocks if the specified event has not yet been set.

3.3.2.3. Conformance classes

The OSEK specification describes a scalable operating system that can be optimized according to the needs of the system. The configurable capabilities are described by conformance classes (CC). A conforming implementation of OSEK must implement at least one of the four conformance classes defined by the OSEK specification. EB tresos AutoCore OS supports all four of them. The characteristics of the four conformance classes are listed in the following table:

Characteristics	**`BCC1`[a]** **Basic Conformance Class 1**	**`BCC2`** **Basic Conformance Class 2**	**`ECC1`[b]** **Extended Conformance Class 1**	**`ECC2`** **Extended Conformance Class 2**
Multiple requesting of task activation	no	yes	no	basic tasks only
Multiple tasks per priority	no	yes	no	yes

Characteristics	**BCC1**[a] **Basic Conformance Class 1**	**BCC2 Basic Conformance Class 2**	**ECC1**[b] **Extended Conformance Class 1**	**ECC2 Extended Conformance Class 2**
Events allowed?	no	no	yes	yes
Resources allowed?	no[c]	yes	yes	yes
Alarms allowed?	yes	yes	yes	yes

[a]`BCC: Basic Conformance Class`

[b]`ECC: Extended Conformance Class`

[c]Can only use the resource RES SCHEDULER

Table 3.1. OSEK conformance classes

3.3.3. Task state transition

The kernel maintains the current state of each task in the system. A task changes from one state to another as the result of kernel calls made by the system. As EB tresos AutoCore OS is a static operating system, there is no means of creating new tasks. Tasks are always present in the system.

The default task state is the *suspended* state. When activated, a task enters the *ready* state. Only one task at a time is in the *running* state, that is the task that currently holds the CPU. Depending on the type of the task, a fourth state may be applicable: extended tasks are in the *waiting* state while they are waiting for an event to be set. EB tresos AutoCore OS adds a fifth state to these four standard OSEK states: *quarantined*, in which the task is suspended and cannot be activated. This state is entered when a task has committed a serious error such as a protection violation.

3.3.4. Task scheduling

Multitasking requires a scheduling algorithm to allocate the CPU between tasks which are in the *ready* state. Scheduling in the EB tresos AutoCore OS kernel is always priority driven. Priorities are assigned during configuration and cannot be changed during runtime except by using resources. It is possible to choose between fully preemptive scheduling, purely non-preemptive scheduling, and mixed scheduling. The choice of the scheduling algorithm is a static decision that must be made during configuration, before the system is generated. It cannot be changed at run-time.

3.3.4.1. Preemptive scheduling

With a preemptive scheduler, the kernel ensures that the CPU is allocated to the highest priority task that is ready to run. This scheduling method is *preemptive* in that if a task with a higher priority than the current task becomes ready to run, the kernel immediately saves the current task's context and switches to the context of the higher priority task.

However, this action will only be performed as a result of a call to a system service by either the task currently running or an ISR running asynchronously to it. The next table, Table 3.2, "Scheduling points for preemptive scheduling" presents a full list of the *scheduling points* for preemptive scheduling.

Situation	System Service
Activation of a task	`ActivateTask()`
Termination of a task	`TerminateTask()` `ChainTask()`
Explicit call to the scheduler	`Schedule()`
Task state transition to the *waiting* state	`WaitEvent()`
Sending an event to a task	`SetEvent()`
Incrementing a counter	`IncrementCounter()` [a]
Releasing a resource	`ReleaseResource()`
Leaving a category 2 ISR	

[a] Point of rescheduling if an alarm is triggered which leads to a call of `SetEvent()` or `ActivateTask()`.

Table 3.2. Scheduling points for preemptive scheduling

3.3.4.2. Non-preemptive scheduling

With a non-preemptive scheduler, rescheduling can only occur at a subset of the scheduling points mentioned above. These are system services which explicitly allow for rescheduling. Table 3.3, "Scheduling points for non-preemptive scheduling" presents a full and complete list of EB tresos AutoCore OS scheduling points for non-preemptive scheduling.

Situation	System Service
Termination of a Task	`TerminateTask()` `ChainTask()`
Explicit call to the scheduler	`Schedule()`
Task state transition to the *waiting* [a] state	`WaitEvent()`

[a] In the case that the event which WaitEvent() should wait for is already set, then this is NOT a point of rescheduling!

Table 3.3. Scheduling points for non-preemptive scheduling

WARNING

The use of non-preemptive scheduling can result in a high priority task being delayed by a running task of lower priority if the latter does not hit any scheduling points. Therefore special care must be taken for timing considerations in non-preemptive real-time systems.

NOTE

The OSEK specification explicitly entitles conforming implementations to restrict the use of system services that serve as scheduling points to top level functions. Programs that do not comply with this restriction will run in an EB tresos AutoCore OS system but may not be portable to other OSEK- or AUTOSAR Os-compliant systems.

3.3.4.3. Mixed scheduling

Mixed scheduling is a combination of preemptive and non-preemptive scheduling. The justification for a mixed scheduling algorithm is that in a fully preemptive system there might be reasons for introducing non-preemptive tasks, such as:

- A task must not be preempted for system-design reasons.
- A task is so small that its execution time is in the same order of magnitude as the time needed for a task switch.
- RAM which is used by the system to save task contexts needs to be used economically.

When mixed scheduling is used, the user has to configure for each task whether it may be preempted or not.

3.3.4.4. Task grouping

Groups of tasks can be defined by associating an internal resource with them. Tasks in such a group are mutually exclusive and, once they have occupied the CPU, cannot be preempted by any other task in the group or by any other task with a priority not greater than the highest priority of the group. They can, however, be preempted by tasks with higher priority than the highest in the group.

Before acquiring the CPU (or after waiting for an event) the tasks in a group retain their own priorities. This means that if two or more tasks in a group are activated (or receive an event) simultaneously, they will execute in order of their configured priority.

3.3.5. Task control

Table 3.4, “Services for task control” gives an overview of the basic EB tresos AutoCore OS tasking routines. These routines provide the means for task control and gaining run-time information. For a detailed description of the EB tresos AutoCore OS operating system services, see Section 7.2.1, “OSEK/VDX API”

Service	Description
`ActivateTask()`, `ChainTask()`	Activation of a task
`GetTaskState()`	Get state of a task
`TerminateTask()`, `Chain-Task()`	Change the current task's state to *Suspended*
`Schedule()`	Call the scheduler explicitly
`GetTaskID()`	Returns the ID of the currently *running* task

Table 3.4. Services for task control

3.3.6. Task stacks

In order to optimize the stack requirements of a system, the Generator can share stacks of basic tasks of the same priority. This means that all basic tasks with the same priority will use the same task stack.

3.4. Task coordination

The complement to the multitasking routines described in Section 3.3, “Tasks” are the task coordination facilities. These facilities permit independent tasks to coordinate their actions.

EB tresos AutoCore OS provides a set of task coordination mechanisms, including:

- *Resources*, for granting exclusive or semi-exclusive access to critical sections of code
- *Events*, for notification of changes to the environment (external or internal)

3.4.1. Resources

In a multitasking environment, tasks typically share access to a number of physical as well as system resources. In EB tresos AutoCore OS these include for example: code sections, the scheduler, memory, and hardware devices. In EB tresos AutoCore OS, *Resources* provide a means to coordinate concurrent access to these shared facilities.

OSEK imposes restrictions on system services which often make the use of resources unnecessary. Resources are only needed in two cases:

- preemptive systems
- non-preemptive systems in which some tasks need to exclude ISRs from system resources.

For some architectures it makes sense to support the *Resource* concept for coordination between tasks and interrupts as well. Consult the *Architecture Notes* to find out whether a specific architecture offers this feature.

3.4.1.1. Deadlock prevention

A deadlock occurs when two tasks wait for resources that are held each by the other. OSEK prevents deadlock situations by imposing a set of restrictions:

- a task makes the transition from *ready* to *running* only if all of the tasks resources are available.
- a task must neither terminate nor enter the *waiting* state while it holds resources.
- multiple resources must be acquired and released in LIFO (last in first out) order. That means each task logically manages the resources it needs on a stack.
- a task must not acquire a resource which it already holds (when required, this can be facilitated using linked resources. See Section 3.4.1.5, “Linked resources” for more information)

3.4.1.2. Priority inversion

Priority inversion arises when a higher-priority task is forced to wait an indefinite amount of time for a lower-priority task to complete. This situation can arise when the lower-priority task `TL` holds a resource that is required by the higher-priority task `TH`. The task `TH` will not enter the running state as long as it cannot acquire the resource. The resource does not become available while `TL` holds it. However, any task that has a priority between that of `TL` and that of `TH` can preempt `TL` if it does not request access to the resource. `TH` will see an indefinite number of lower-priority tasks run while `TL` occupies the resource.

OSEK prevents this situation by imposing the *priority ceiling protocol* on its resource management. Ceiling priorities are calculated according to the following rules:

- when a task acquires a resource, the task's priority is set to a value equal to or higher than that of the highest-priority task `TH` that uses this resource at some time.
- the new priority of the task is lower than the priority of any task which has higher priority than `TH`.
- when the task releases the resource, the priority of the task is immediately reset to the level that the task had before acquiring the resource.

This protocol ensures that no task of lower priority than task `TH` will execute while task `TL` holds the resource. Task `TL` is executed preferentially until the moment it releases its resources, thus enabling task `TH` to change into the *running* state.

3.4.1.3. Scheduler resource

A preemptive task might be in a situation in which it must avoid preemption by all means. For this purpose, the system can provide a predefined resource `RES_SCHEDULER` if configured. Acquisition of this resource deactivates the scheduler. Releasing `RES_SCHEDULER` reactivates scheduling.

NOTE

Interrupts are received and processed independently of the current state of the scheduler.

3.4.1.4. Internal resources

Internal resources are not visible to the API; they exist only during configuration. Each task can be associated with at most one internal resource. Internal resources are automatically handled by EB tresos AutoCore OS. When a task enters the *running* state it takes the resource. The resource is only released when the task is terminated, calls `Schedule()` or enters the *waiting* state by calling `WaitEvent()`. If the task changes to the *ready* state by being pre-empted, it still holds the resource. See Section 3.3.4.4, "Task grouping" for further information.

3.4.1.5. Linked resources

Linked resources can be used in place of nested acquisitions of a single resource. A linked resource acts as a second copy of the resource. Linked resources can link to `RES_SCHEDULER` only if it is explicitly defined.

3.4.1.6. Resource control

Table 3.5, "Services for resource control" provides an overview of the basic EB tresos AutoCore OS resource services. These routines provide the means for getting and releasing resources. See Section 7.2.1, "OSEK/VDX API" for a detailed description of the EB tresos AutoCore OS operating system services.

Description	Service
Get (acquire) a resource	`GetResource()`
Release a resource	`ReleaseResource()`

Table 3.5. Services for resource control

3.4.1.7. Restrictions while resources are held

The system services `TerminateTask()`, `ChainTask()`, `Schedule()` and `WaitEvent()` must not be called while a resource is held. If the architecture supports resources for interrupt service routines, the ISR must not finish while a resource is held. The only exception is internal resources as they are managed internally by EB tresos AutoCore OS.

3.4.2. Events

In an embedded system it is often necessary to inform a task of a state transition. This may be a value which is calculated by another task or an input from a device that must be sent from the interrupt handler to a certain task.

An event is an exclusive signal which is assigned to an arbitrary extended task. More than one event can be assigned to the same task. A task may wait for one or more events and thus enter the *waiting* state. Any task (basic or extended) can set events for arbitrary tasks. This causes the receiving task to enter the *ready* state if it has been waiting for at least one of these events.

3.4.2.1. Event control

Table 3.6, “Services for event control” provides an overview of the basic EB tresos AutoCore OS event services. These routines provide the tools for manipulating event services. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the EB tresos AutoCore OS operating system services.

Description	**Service**
Set an Event of a specific task	`SetEvent()`
Wait for specific events	`WaitEvent()`
Get the events of a specific task	`GetEvent()`
Clear events	`ClearEvent()`

Table 3.6. Services for event control

3.5. Counters and alarms

The EB tresos AutoCore OS operating system provides services to handle recurring events typically associated with an interrupt. Such events may occur at regular intervals, such as timer interrupts, or irregularly, such as interrupts transmitting the state of a certain device. The support of such events is divided into two elements:

- *counters* which register the occurrences of the events.
- *alarms* which are triggered when a counter reaches a certain value.

3.5.1. Counters

A counter consists of a current value expressed in ticks and a number of constants describing the behavior of the counter. These include:

MAXALLOWEDVALUE
: the number of ticks at which the counter rolls over and restarts from zero

MINCYCLE
: the minimum number of ticks a cyclic alarm associated with the counter must specify for expiration

TICKSPERBASE
: the number of ticks that are needed to reach a specific value defined by the user

A counter can be either a hardware counter or a software counter.

3.5.1.1. Software counters

Software counters are implemented entirely in software and are incremented using one of the system services provided for this purpose, such as `IncrementCounter`. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the EB tresos AutoCore OS operating system services.

3.5.1.2. Hardware counters

Hardware counters are software abstractions of the timers and counters present on most microcontrollers and are assumed to be advanced automatically in hardware, typically by a constant-frequency signal derived from the CPU's clock frequency.

The software abstraction is provided by a device driver. The EB tresos AutoCore OS operating system includes drivers for the timers most commonly used on each architecture, but it is possible to use other timers and counters by writing a suitable driver. The hardware support that is needed is a counter that counts impulses and a compare register that generates an interrupt when the counter reaches a predetermined value. Some microcontrollers offer different kinds of counters, such as counters that are preset to a specified value, then count down and generate an interrupt at zero. Such counters cannot be used directly by EB tresos AutoCore OS, and the driver must emulate or approximate the expected behaviour.

3.5.1.3. Counter control

Table 3.7, “Services for counter control” gives an overview of the basic EB tresos AutoCore OS counter services. These routines provide the mechanism for manipulating counters and getting information about counters. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the AUTOSAR OS operating system services.

Description	Service
Get the current tick value of a specific counter	`GetCounterValue()`
Get the number of ticks between the current tick value and a previously read tick value	`GetElapsedValue()`
Increment a software counter	`IncrementCounter()`

Table 3.7. Services for counter control

3.5.2. Alarms

Counters are used to register the occurrence of certain events. Typically, when a certain number of these events have occurred, some action needs to be taken by the system. Alarms are the mechanism used to connect a counter to an action - either the activation of a task, the setting of an event, the running of a user-defined alarm-callback routine or incrementing a software counter.

Each alarm is associated with exactly one counter but multiple alarms can use the same counter. When the counter reaches a user-defined value the alarm expires. Alarm expiration results in the defined action being carried out (e.g. task activation or setting an event). The action depends on the alarm's static properties and can be one of:

- Activating a task
- Setting an event
- Incrementing a user-defined software counter
- Calling an alarm-callback routine. The callback should be defined using `ALARMCALLBACK(name)`.

An alarm callback function is executed in the current context of the `IncrementCounter()` call. That means that for a counter associated with a hardware counter it is run at interrupt level. Since the Generator does not know about the stack usage of this callback, the user must provide for this. It might be even necessary to introduce a dummy ISR to artificially increase the interrupt stack size. Generally it is a good idea to do only small things in a callback (setting a global variable, etc.).

3.5.2.1. Alarm control

Table 3.8, “Services for alarm control” gives an overview of the basic EB tresos AutoCore OS alarm services. These routines provide the mechanism for manipulating alarms and getting information about alarms. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the AUTOSAR OS operating system services.

Description	Service
Set or activate an alarm	`SetRelAlarm()` `SetAbsAlarm()`
Cancel an alarm	`CancelAlarm()`
Get information about an alarm and the counter to which the alarm belongs	`GetAlarm()` `GetAlarmBase()`

Table 3.8. Services for alarm control

3.5.3. Schedule tables

Closely related to the alarm concept is the schedule table. A schedule table is a predefined sequence of actions that must take place at intervals (not necessarily regular) after a trigger event.

Schedule tables are associated with counters in a way identical to that of alarms. Ticks of the counter provide the underlying interval measurement for the schedule table. Schedule tables can be single-shot, in which case they run through their sequence of actions once after being started, or periodic, in which case the sequence repeats at regular intervals.

Periodic schedule tables can be maintained in synchronization with an external stimulus. A schedule table that is out-of-sync with its stimulus can be brought into sync suddenly or gradually in steps of a configured size.

At each defined expiry point, the schedule table can activate tasks and set events for tasks. There can be multiple activations and/or events set at each expiry point. Task activations are performed chronologically before events are set, so that a task can be activated and have an event set for it at the same expiry point.

3.5.3.1. Schedule table control

Table 3.9, “Services for schedule table control” gives an overview of the basic EB tresos AutoCore OS schedule table services. These routines provide the mechanism for manipulating schedule tables and getting information about schedule tables. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the EB tresos AutoCore OS operating system services.

Description	Service
Start a schedule table with a relative offset	`StartScheduleTableRel()`
Start a schedule table with an absolute offset	`StartScheduleTableAbs()`
Stop a schedule table	`StopScheduleTable()`

Description	Service
Start a schedule table when another schedule table completes	`NextScheduleTable()`
Synchronize a schedule table with an external signal	`SyncScheduleTable()`
De-synchronize a schedule table when the external signal is lost	`SetScheduleTableAsync()`
Get information about a schedule table	`GetScheduleTableStatus()`

Table 3.9. Services for schedule table control

3.6. Interrupt service routines (ISRs)

ISRs are typically short pieces of code which differ in their use of system services. OSEK defines two categories of ISRs, both of which are implemented in the EB tresos AutoCore OS kernel.

Irrespective of the category, no scheduling will take place inside an interrupt routine. Rescheduling may take place once the execution of a *category 2* interrupt has finished as long as it does not return to another interrupt routine, and no other interrupts are pending.

A triggered interrupt that has no configured ISR will raise an *UnknownInterrupt* error, which will cause an ErrorHook to be executed if one is configured.

3.6.1. ISR category 1

This is the fastest category of ISR. With a few exceptions, system services may not be called from category 1 ISRs. After processing the interrupt, execution of the interrupted code will continue at the very instruction at which it was interrupted. No rescheduling will take place. EB tresos AutoCore OS nevertheless provides a minimal frame in which the ISR executes in order to protect itself against misuse of system services in a category 1 ISR and to guarantee correct operation of those services that are permitted.

Category 1 ISRs must be of higher priority than the highest category 2 ISR. The AUTOSAR Os Generator enforces this rule and gives an error if the user specifies interleaved category 1 and 2 priorities.

3.6.2. ISR category 2

ISRs of this category are permitted to call system services, so the kernel provides a full execution frame for the ISR. The user code for the ISR will be called from this frame.

The operating system will reschedule the tasks after returning from a category 2 ISR that returns to a task.

NOTE

A category 2 ISR is not permitted to call arbitrary system services. See the API appendices in order to find out which routines can be called from an interrupt.

3.6.3. Interrupt control

Table 3.10, “Services for interrupt control” provides an overview of the basic EB tresos AutoCore OS interrupt services. See Section 7.2.1, “OSEK/VDX API” for a detailed description of the EB tresos AutoCore OS operating system services.

Description	Service

Enable or disable all category 2 interrupts	`ResumeOSInterrupts()`
	
	`SuspendOSInterrupts()`

Enable or disable all interrupts
 	
(including category 1, nesting not allowed)
	`EnableAllInterrupts()`
	
	`DisableAllInterrupts()`

Enable or disable all interrupts
 	
(including category 1, nesting allowed)
	`ResumeAllInterrupts()`
	
	`SuspendAllInterrupts()`

Enable or disable a specific ISR	`OS_EnableInterruptSource()`
	
	`OS_DisableInterruptSource()`

Table 3.10. Services for interrupt control

3.6.4. Architecture issues

On some architectures, there are only one or two hardware pins for an interrupt. On such hardware systems, the interrupts are often connected to a dispatcher which will detect the actual source of the interrupt. This may be due to a timer under-/overflow, data sent or received on a serial device, etc.

Other architectures may provide prioritized interrupt handling for interrupt sources. In those cases a dispatcher is not required.

See the *Architecture Notes* specific to your target architecture to learn more about the implementation of interrupt handling.

3.6.5. Compiler issues

Many compilers for embedded systems offer direct support for interrupt service routines written in C through special keywords (for example, `interrupt`). Such a keyword is not normally supported in EB tresos AutoCore OS because of the need to protect the kernel against faulty application code. See the *Architecture Notes* for your architecture.

3.7. Miscellaneous features

3.7.1. System startup and shutdown

A pair of system services, `StartOS()` and `ShutdownOS()` are provided to start EB tresos AutoCore OS and shut it down.

`StartOS()` must be called by the user's startup code, typically at the end of the `main()` function, before any other system service can be called.

`ShutdownOS()` is called by the kernel when it encounters a fatal error. It may be called explicitly by an application as well. After this call, the EB tresos AutoCore OS system services are no longer available.

3.7.2. Hook routines

EB tresos AutoCore OS supports a number of hook routines which are called at specific points in the system. They execute at a higher priority than any other task in the system.

The following needs to be done in order to establish a hook:

- A function must be implemented that uses the exact interface of the hook.
- The hook must be included into your configuration by setting the appropriate hook attribute to TRUE. (see Section 7.1, “EB tresos AutoCore OS Configuration Language”.

This chapter gives an overview of the hooks and when they are executed. See Section 7.2.1, “OSEK/VDX API” for the hook interface specifications.

NOTE

Many system services cannot be called from a hook. See the API appendices in order to find out which routines can be called by individual hooks. The EB tresos AutoCore OS kernel detects such erroneous calls so that no damage to the system occurs. The action taken on violation of the rule depends on the status mode of the system.

3.7.2.1. Error hook

This hook is called by the kernel when a system service that returns a StatusType returns anything other than `E_OK` and by other system services when an error condition is detected. It can also be called by the kernel if internal checking detects an error condition. More information on the cause of the error can be retrieved by using the services `OSErrorGetServiceId()` and `OSError_x1_x2()`. `GetISRID()` and `GetTaskID()` can be used to get the ID of the ISR or task that produced the error.

NOTE

Errors which occur inside the `ErrorHook()` do not lead to recursive calls of this hook.

3.7.2.2. Protection hook

The protection hook is called if a serious error occurs, for example exceeding the worst case execution time or stepping outside the memory protection boundary. Depending on the return value the OS will either kill the task or category 2 ISR which causes the problem, kill and optionally restart the OS-Application to which the task or category 2 ISR belongs or shutdown the system.

3.7.2.3. Pre-ISR hook

See Section 3.7.7, “ISR hooks”.

3.7.2.4. Post-ISR hook

See Section 3.7.7, “ISR hooks”.

3.7.2.5. Pre-task hook

This hook is called in the context of the kernel just before a new task context is entered. The current task as returned by GetTaskID() will be the incoming task, and the incoming task's state as returned by GetTaskState() will be RUNNING.

3.7.2.6. Post-task hook

This hook is called in the context of the kernel just after leaving the current task's context. This may be due to preemption, task termination, or a task waiting for an event. The current task as returned by GetTaskID() will be the outgoing task, and the outgoing task's state as returned by GetTaskState() will be RUNNING.

3.7.2.7. Startup hook

This hook is called by the kernel at system startup after the system has been initialized but before the scheduler is activated. While the hook is executing, all interrupts are disabled. Note that according to the OSEK specification the `StartupHook()` must not be used to initiate alarms or activate tasks.

3.7.2.8. Shut down hook

This hook is called whenever `ShutdownOS()` has been called either explicitly by an application or implicitly by the system. It can be used, for example, for reinitialization or error logging. When the hook returns, the system shuts down all its services.

3.7.2.9. Application-specific startup hook

This application-specific hook is called by the kernel with the access rights of the associated OS-Application on startup of the OS but after the system-specific StartupHook.

3.7.2.10. Application-specific shut down hook

This application-specific hook is called by the kernel with the access rights of the associated OS-Application on shutdown of the OS and before the system-specific ShutdownHook.

3.7.2.11. Application-specific error hook

This application-specific hook is called by the kernel just after the global `ErrorHook()` is called if the error was caused by the application that owns the hook.

3.7.3. Debugging support

OSEK provides two execution modes (called status in the OSEK specification): standard and extended. The intention of the OSEK specification is to permit the system to run without the overhead of error checking.

However, the safety-critical nature of typical EB tresos AutoCore OS systems and the fact that there is little or no generated or conditionally-compiled code means that error checking is always present. So EB tresos AutoCore OS uses the status setting in a different way.

If the system is operating in *standard* mode, most system services do not return if an error that can be attributed to a faulty program is detected. Instead, the caller is terminated. This permits the system to continue without the faulty component.

In *extended* mode system services normally return an error code to the caller unless the error is so serious that the kernel cannot permit the application to continue. See the APIs described in the appendices of this document for error codes and their semantics.

If you are using *standard* mode and are compiling an optimized kernel from source code, most of the *extended* mode error checking is turned off. This mode of operation must not be used for safety-critical systems.

In all cases the kernel calls the error hook before finally taking the necessary action.

3.7.4. OS API

The OSEK API functions described so far are implemented in terms of the underlying OS API. With a few exceptions, these are simply OSEK macros that call the underlying OS user-side library routine, which in turn makes the corresponding system-call to the kernel. The use of C macros means that the user has to take special care regarding the following facts:

- The existence of a function pointer for an OSEK API function cannot be assumed. However the address of the underlying OS user-side function can sometimes be taken, but the programmer needs to take account of the small differences between the OSEK and OS APIs.
- In general, macro calls should not use as arguments expressions that could have unexpected effects if evaluated more than once. Problematic expressions are those that have side effects, such as assignment operators, increment operators, function calls and accesses to hardware registers. Accessing a volatile variable should also be avoided, as different values could be obtained on successive reads. However, in practice the OSEK API macros are designed such that each parameter is only evaluated once, so such expressions should not cause problems. If in doubt, see the macro definitions in the header file `Os_osek.h` for the exact definition of any macro.

3.7.5. Version macros

All versions of EB tresos AutoCore OS define the following macros:

- `OS_AUTOSAROS_VER` is a positive number indicating the version number of the EB tresos AutoCore OS release.

- `OS_AUTOSAROS_REV` is a positive number indicating the revision number of the EB tresos AutoCore OS release.
- `OS_AUTOSAROS_CORE_BUILD` is the build number of the kernel core of this EB tresos AutoCore OS release.
- `OS_AUTOSAROS_ARCH_BUILD` is the build number of the architecture specific parts of the kernel of this EB tresos AutoCore OS release.

NOTE

`OS_AUTOSAROS_CORE_BUILD` and `OS_AUTOSAROS_ARCH_BUILD` must be the same, otherwise reliable operation is not guaranteed

3.7.6. The stack checker

The stack-check feature can be used to check if a stack over- or underflow occurred. For this purpose the stacks are filled with a known value (typically `0xEE` or `0xEB`) at startup. There is an API which allows the system to investigate the stack use by counting the number of bytes that are not yet overwritten and still contain the marker code. The stack-check feature includes the following functions:

Description	Service
Check the calling context for over- or underflow.	`stackCheck()`
Determine the number of unused bytes on the stack of a specified task.	`getUnusedTaskStack()`
Determine the number of bytes used on the stack of a specified task.	`getUsedTaskStack()`
Determine the number of unused bytes on the kernel stack.	`getUnusedIsrStack()`
Determine the number of bytes used on the kernel stack.	`getUsedIsrStack()`

Table 3.11. Services for stack checking

Before calling a category 2 ISR or a trusted function, the kernel checks to ensure that there is enough kernel stack to satisfy the function's needs. If not, the ISR or trusted function is not called.

Additionally, the system checks during every task-switch and after the termination of an ISR category 2 if the stack is in a valid state. If this is not the case the system calls the `ShutdownOS()` service with either `E_OS_SYS_StackOverflow` or `E_OS_SYS_StackUnderflow`.

NOTE

Some implementations of EB tresos AutoCore OS may deviate from this API. Consult the proper *Architecture Notes*.

NOTE

Memory protection, when available, can give an immediate indication of stack over- or underflow before any damage has been done.

3.7.7. ISR hooks

ISR hooks can be used to evaluate the time consumption of interrupt service routines. This feature can be enabled globally by setting the switches `PREISRHOOK` and `POSTISRHOOK` to true in the OS object. The `PreIsrHook(IsrType)` is called just before entering the user code of the interrupt service routine and `PostIsrHook(IsrType)` is called just afterwards.

The parameter can be used to detemine which ISR has occurred. The Generator creates constant macros that give numeric values to the names used in the OIL file, and these values are passed to the hook functions as parameters.

The environment of the ISR hooks is the same as the environment of the ISR itself. Therefore the same restrictions for calling system services apply.

NOTE

Unlike the PreTaskHook() and the PostTaskHook() the ISR hooks are called exactly once for each invocation of an interrupt service routine (i.e. at start and termination).

3.7.8. Scalability classes

In order to customize the operating system to the needs of the user and to take full advantage of the processor features, the operating system can be scaled according to the following scalability classes:

Feature	**SC1**	**SC2**	**SC3**	**SC4**	**Hardware requirements**
OSEK OS (all conformance classes)	yes	yes	yes	yes	

Feature	SC1	SC2	SC3	SC4	Hardware requirements
Counter Interface	yes	yes	yes	yes	
Schedule Tables	yes	yes	yes	yes	
Stack Monitoring	yes	yes	yes	yes	
ProtectionHook	no	yes	yes	yes	
Timing Protection	no	yes	no	yes	Timers(s) with high priority interrupt
Global Time /Synchronization Support	no	yes	no	yes	Global time source
Memory Protection	no	no	yes	yes	MPU
OS-Applications	no	no	yes	yes	
Service Protection	no	no	yes	yes	
CallTrustedFunction	no	no	yes	yes	(Non-)privileged modes

Table 3.12. Scalability Classes

3.8. Timing protection and runtime measurement

3.8.1. Timing protection

EB tresos AutoCore OS offers various timing protection features to detect timing deadline errors in a real-time system and to prevent degraded performance if a task uses too much CPU time.

3.8.1.1. Task/ISR execution time control

The execution budget of a task or category 2 ISR sets a statically-configured upper bound on the execution time of the task or ISRs.

The maximum allowed time can be configured individually for each task and category 2 ISR.

The measurement of a task's execution time starts at zero when it is activated and ends when it successfully calls `TerminateTask()` or `ChainTask()`. The measurement also ends and restarts from zero on a successful call to WaitEvent() regardless of whether the event is already pending or not.

The measurement of an ISR's execution time starts when the ISR is called and ends when it returns.

Time spent away from the task or ISR when interrupted during its execution by a higher priority task or category 2 ISR does not count as part of its execution time.

If a timing violation is detected the protection hook is called with `E_OS_PROTECTION_TIME`.

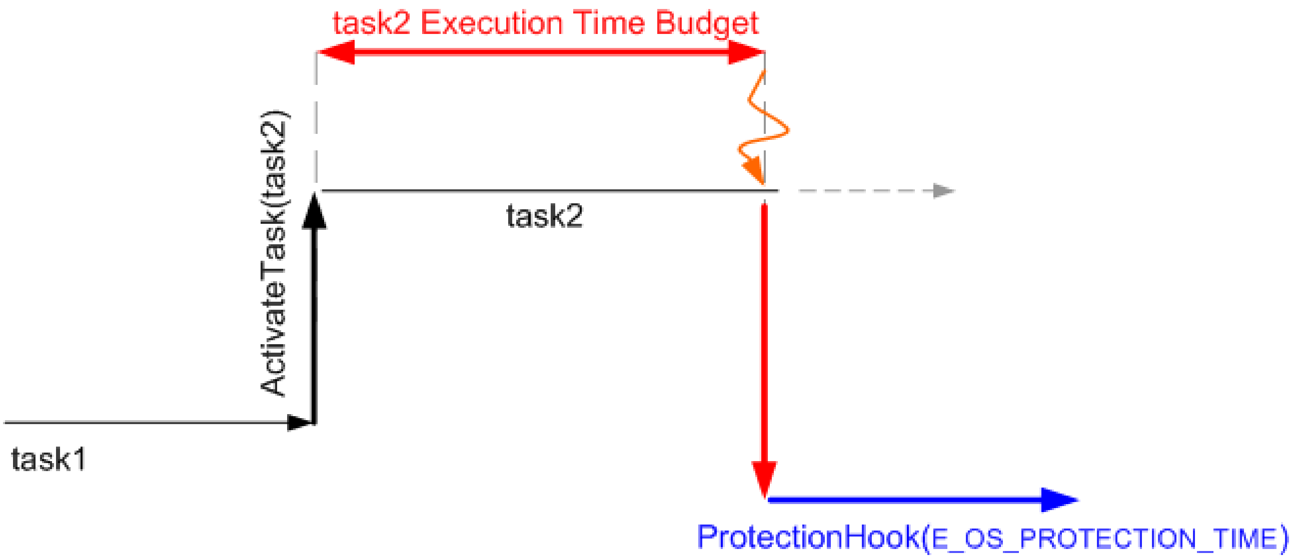


Figure 3.2. Task Execution Time Control

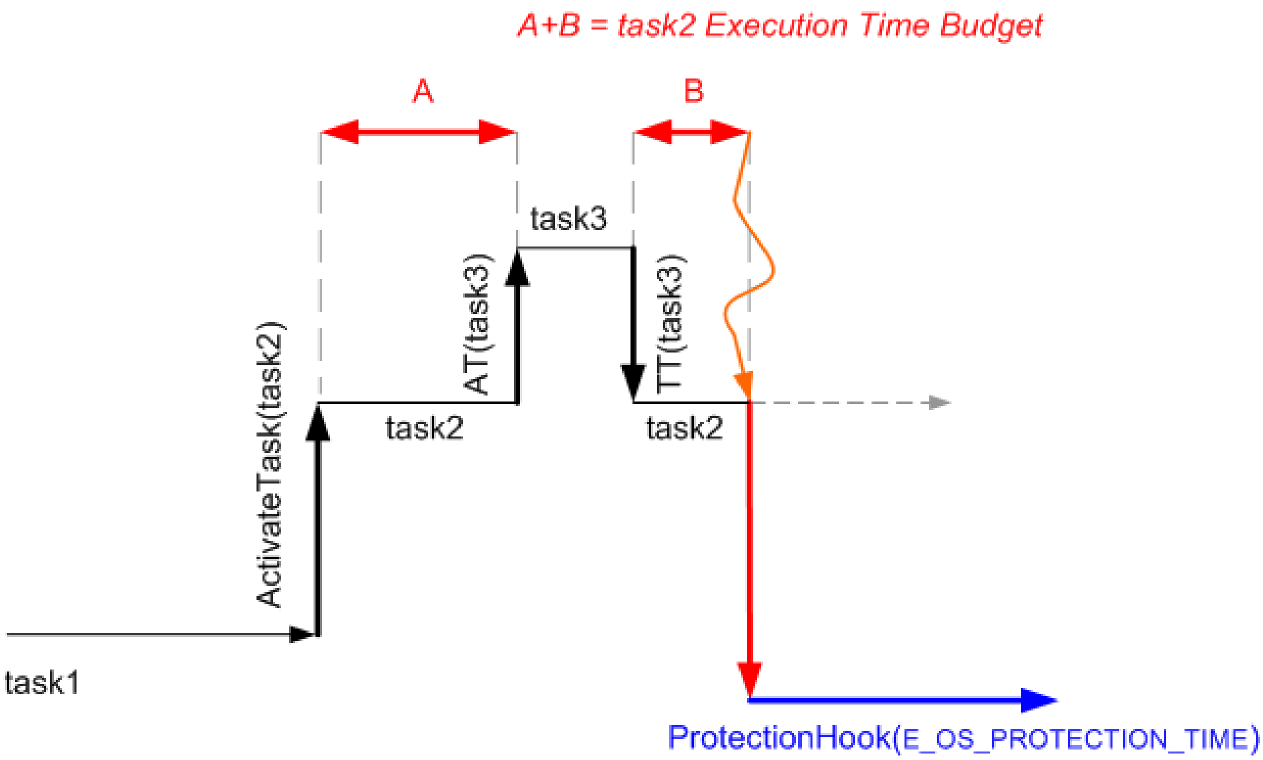


Figure 3.3. Task Execution Time Control (preempted)

3.8.1.2. Task/ISR execution time control while holding shared resources

The resource lock time of a task or ISR for a resource sets a statically-configured upper bound on the execution time that the task or ISR can use while holding the resource.

The resource lock time can be configured individually for each resource and for each task or ISR that uses the resource.

Resource lock timing starts when the resource is acquired and stops when it is released. The imposition of a resource lock time will never result in the extension any other time limit that is in force for the calling task or ISR.

As for execution budget measurement, the time spent away from the task or ISR while interrupted by a higher-priority task or category 2 ISR does not count towards the interrupted task or ISR.

If a timing violation is detected the protection hook is called with `E_OS_PROTECTION_LOCKED`.

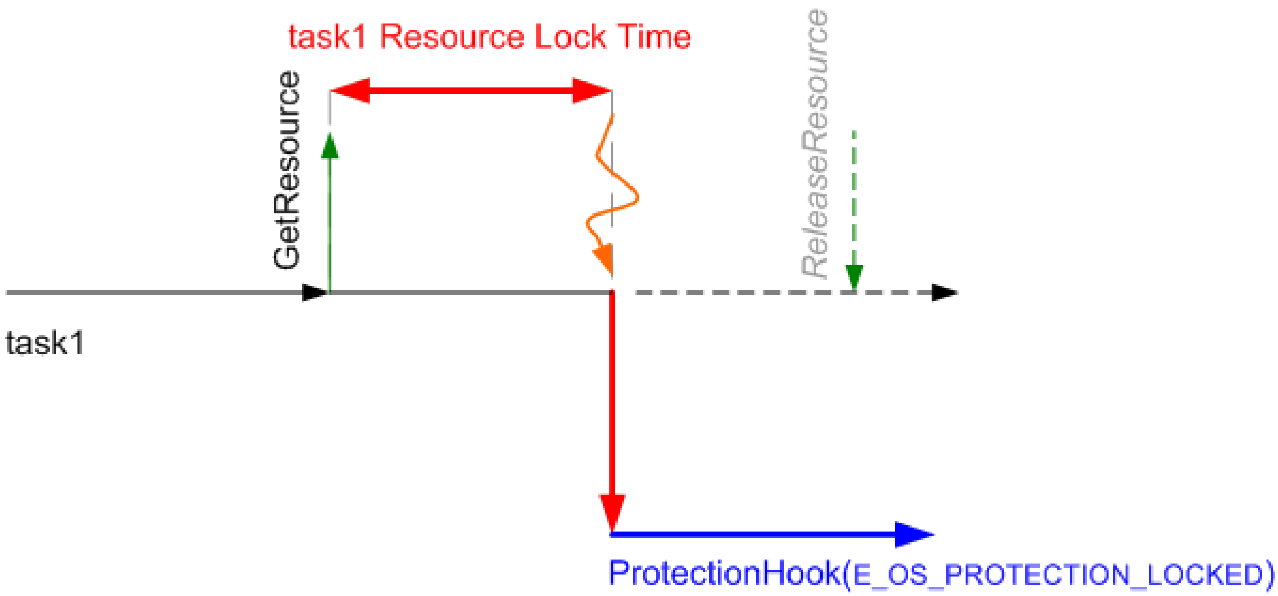


Figure 3.4. Task Execution Time Control while holding shared resources

3.8.1.3. Task/ISR execution time control for interrupt locks

The Interrupt Lock Times of a task or category 2 ISR set statically configured upper bounds on the execution time of the task or ISR while it has locked interrupts using one of the kernel interrupt-locking APIs. There are two Interrupt Lock time limits: one for the SuspendOSInterrupts() lock and one for the SuspendAllInterrupts() and DisableAllInterrupts() lock.

Interrupt lock timing starts when the lock function is called, and ends when the corresponding unlock function is called. Nested calls to the same type of lock/unlock functions do not restart the timing, so the budget sets a limit on the execution time between outermost lock/unlock pair.

The DisableAllInterrupts() and SuspendAllInterrupts() lock functions can be used nested inside a SuspendOSInterrupts()-ResumeOSInterrupts() pair and the lock timing will be nested as expected, but if SuspendOSInterrupts() is called while interrupts are locked using either SuspendAllInterrupts() or DisableAllInterrupts() the lock timing for SuspendOSInterrupts() will not start. It is assumed that the lock time for "all" interrupts is more critical (and therefore shorter) than the lock time for "OS" interrupts. It is therefore an error to configure an "OS" lock time but no "all" lock time.

If a timing violation is detected the protection hook is called with `E_OS_PROTECTION_LOCKED`.

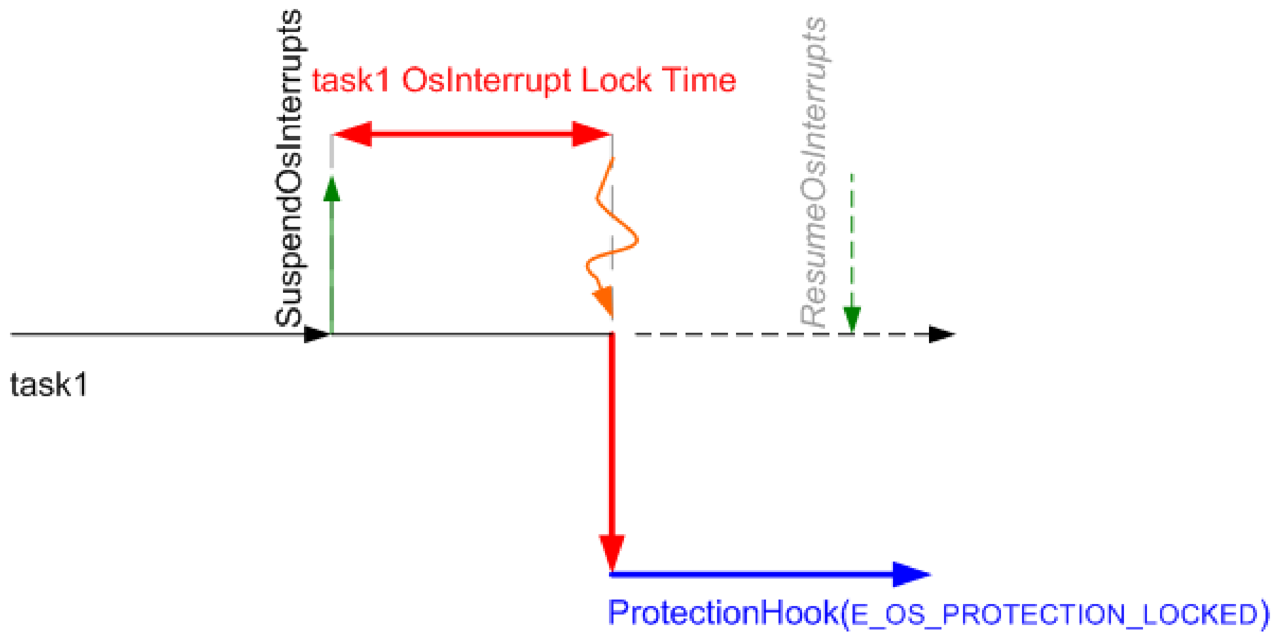


Figure 3.5. Task Execution Time Control while disable interrupts

3.8.1.4. Task and ISR arrival rate control

Arrival rate enforcement sets a statically-configured upper bound on the number of times that a task or ISR can be invoked in a given timeframe.

An ISR is invoked whenever it is called. A task is invoked whenever it is activated. The return from a successful call to `WaitEvent()` also counts as an invocation, regardless of whether the event was pending or not. Thus an "invocation" occurs at precisely those times when the execution-time measurement is set to zero.

The AUTOSAR Os specification merely requires a minimum time between two invocations to be enforced. The drawback with this is that an interrupt can be requested at regular time interval but the invocation can be delayed by the execution of higher-priority ISRs or by the use of interrupt or resource locking. If an invocation is delayed, the time since the last invocation will appear longer and the time to the next invocation will appear shorter. If the configured minimum time between two invocations does not take the worst-case delay into account the arrival-rate enforcement might occur spuriously. However, if the worst-case delay is taken into account, the minimum time might be so short that an "out-of-control" interrupt source might adversely affect the system's timing without being detected.

To reduce the effect of this drawback, EB tresos AutoCore OS permits a number of interrupts to be specified over a longer time-frame, thus averaging the jitter. The worst-case delay still needs to be take into account, but this is spread over a longer time-frame and so lessens the possibility of a fault going undetected.

Naturally, it is possible to achieve strict AUTOSAR conformance by setting the invocation limit to 1, and this value is used if no invocation limit is specified.

If a rate violation is detected the protection hook is called with `E_OS_PROTECTION_RATE`.

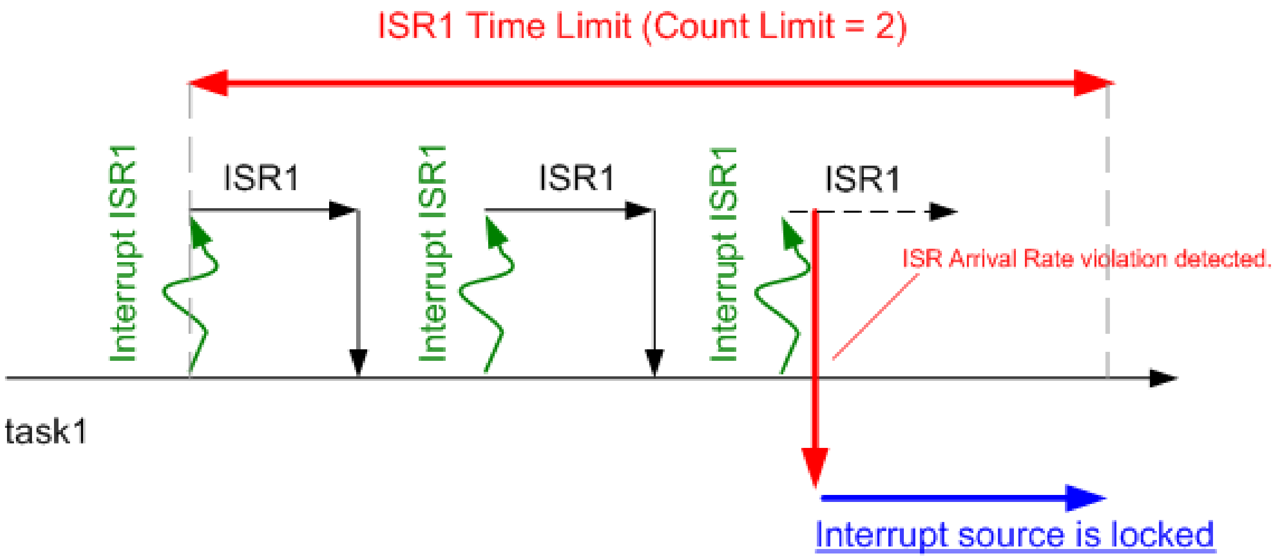


Figure 3.6. ISR arrival rate control

3.8.2. Time stamp

The time stamp feature is designed to provide a readable timer that increases at a constant rate and never wraps around back to zero. The value returned by the time stamp API is a 64-bit value.

If a suitable long-duration timer is available in the hardware, it is used directly by the OS. In this case, the **OsTimestampTimer** parameter is fixed at INTERNAL and cannot be changed.

If no long-duration hardware timer is available the OS constructs the time stamp in software using a hardware timer of shorter duration. You can select the short duration timer by means of the **OsTimestampTimer** parameter. A hardware timer that you select for use as the time stamp can be used for another purpose at the same time.

When the time stamp is read directly from hardware, the time stamp feature relies on the fact that a 64-bit value will never wrap around while the OS is running. Depending on the reset behavior of the timer, the OS may need to initialize the timer during `StartOS()` to a sufficiently low value. You can find more details on the actual timer hardware and its properties in the architecture notes.

3.8.3. CPU-load measurement

WARNING **Not recommended for production use**

The CPU load measurement feature is intended as a development aid. The computation time is essentially unbounded and could lead to significant latencies up to the point where a watchdog - if present - might reset the system. Unless your system can avoid or tolerate those latencies and resets, it is recommended to disable the CPU-load measurement for production use.

The CPU load measurement feature measures the total CPU load of tasks, ISRs etc. It works by calculating how much time the CPU spends while not in the idle loop of the OS, as a percentage of the total time. Time is measured using the time stamp feature, and is assumed to be monotonically increasing. If the time stamp feature is based on a long-duration hardware timer the load measurement is slightly more accurate because the idle exit time can be obtained in assembly-language code closer to the start of the interrupt handling before a full C environment is available.

The time is measured in configurable intervals. The value returned by `GetCpuLoad()` is an 8-bit integer representing the percentage load (range 0..100) averaged over a configurable number of intervals. The calculated value is rounded to an integer in a configurable way. The computation also stores the highest percentage value seen and returns this value when `GetMaxCpuLoad()` is called. The function `InitCpuLoad()` can be called to set the "highest percentage" back to zero. To change the interval length, use the **OsCpuLoadIntervalDuration** parameter. The default interval length is 100ms. To change the number of intervals, use the **OsCpuLoadNumIntervals** parameter. The default number is 10. To change the rounding behavior, use the **OsCpuLoadRounding** parameter.

The timer is read and stored ("idle exit time") whenever the CPU leaves the idle loop (by interrupt).

The measurement is performed whenever the idle loop is (re-)entered. The timer is read at the start, thus the computation is counted as part of the idle loop. The idle time ("last idle entry time" to "idle exit time") and the busy time ("idle exit time" to "idle entry time") are calculated. These times are then allocated to sequential intervals. Each time an interval gets filled, the percentage over all intervals is calculated and stored ("current load") and if that's higher than the "peak load" gets stored there too. Then the computation moves to the next interval in a circular fashion ("number of intervals"). This repeats until all the time spent idle and busy is accumulated.

When `GetCpuLoad()` or `GetMaxCpuLoad()` is called, by definition the CPU is busy. The computation is performed by briefly switching to "idle" to perform the computation, then back to "busy".

NOTE

CPU-load measurement and background tasks

If your system has a background task that never terminates, the CPU load will always be 100%.

If you have a system with 100% load and call `GetCpuLoad()` or `GetMaxCpuLoad()` infrequently, the computation will take a long time as the algorithm tries to account for all the time spent "busy" since the last call. For example, if you have a 100ms interval (default) and call `GetCpuLoad()` every 10 minutes, it will calculate the results for 6000 intervals during the call. The time stamp range is long enough to cover several years without overflowing, so in the extreme the calculation could take a very long time.

Only integer operations are used. Floating-point registers are not affected by the calculation of the CPU load.

NOTE

For performance reasons, the API functions `GetCpuLoad()`, `GetMaxCpuLoad()` and `InitCpuLoad()` are implemented as function calls to kernel functions. As a result, calling them from non-trusted applications when using a system-call kernel will not work.

If you need to access the CPU load measurement facilities from a non-trusted application in a system-call kernel, use the API functions `OS_UserGetCpuLoad()` and `OS_UserResetPeakCpuLoad()` instead. See Section 7.2.4.2, “Reference” for a detailed description of these functions.

4. Application example

4.1. Overview

The idea behind the application examples is to give an example of how to start a new project. It is kept as simple as possible. Keep in mind that application examples are only a rudimentary starting point and must not be used as basis for a real ECU. Projects for real ECUs are much more complex and you need knowledge about several parts of the AUTOSAR standard.

The workflow of the `os_demo` application example is as follows:

1. Importing the application example as a project into EB tresos Studio.
2. Adapting your build environment.
3. Configuring the `Os` module.
4. Building the sample code of the application example.
5. Checking whether the sample code was built correctly.

The following chapter provides you with background information, as well as instructions for setting up and working with the `os_demo` application example:

- Where do I find which file?

  What does the application example actually do?

  Which prerequisites do I need to work with the `os_demo` application example?

  Section 4.2, “Background information” answers these questions.
- How do I import the application example into EB tresos Studio?

  How do I adapt the build environment?

  Section 4.3, “Setting up the application example” answers these questions.
- How do I generate the code of the application example?

  Section 4.4, “Building the application example” answers this question.
- How do I check the code output of the application example?

  Section 4.5, “Checking whether your code was built correctly” answers this question.

4.2. Background information

4.2.1. Prerequisites for running the `os_demo` application example

To run the `os_demo` application example, you need the following prerequisites:

- EB tresos Studio installed on your PC.
- The EB tresos AutoCore `Os` module in the `plugin` folder of your EB tresos Studio installation.
- The EB tresos AutoCore `Make` module in the `plugin` folder of your EB tresos Studio installation. The `Make` module implements hardware-independent parts of the EB tresos AutoCore build environment.
- The EB tresos AutoCore `Platforms` module in the `plugin` folder of your EB tresos Studio installation. The `Platforms` module implements hardware-specific parts of the EB tresos AutoCore build environment.
- A target device to which you may transfer the resulting code of the application example. Ideally, the target should support a debug connection to verify it more thoroughly. An LED panel on the board is also useful.

For information on installing EB tresos Studio and EB tresos AutoCore modules, see the EB tresos installation guide.

4.2.2. File and directory structure

In your installed EB tresos AutoCore OS package, you find all application example-dependent files in the directory `$TRESOS_BASE\demos\AutoCore_OS\os_demo_<target>_<version>`[1]. During the setup of the application example, this directory will be copied into your workspace directory (e.g. `$TRESOS_BASE\workspace`).

`$TRESOS_BASE` is the directory into which you have installed EB tresos Studio, e.g. `C:\EB\tresos`.

4.2.2.1. Location of the makefiles

The makefiles of the application example `os_demo` are located in the directory `util`.

[1]The actual name of the directory depends on the target platform and `Os` version. It may look like the following: `$TRESOS_BASE\demos\AutoCore_OS\os_multicore_demo_TC277_6.0.54`

4.2.2.2. Location of the configuration file

The configuration file `Os.xdm` of the application example `os_demo` is an XDM file, which is located in the directory `config\Os.xdm`.

4.2.2.3. Location of the source files

The C source files used for the application example consist of:

- the main common application file `source\demo.c`, which contains the implementation of all tasks and ISR routines, and
- a bundle of board-specific files implementing board-specific functions and macros located in the `source\boards` directory.

4.2.2.4. Debug files and workspaces

Some EB tresos AutoCore OS plugins are delivered with additional files such as workspaces or projects for the specific toolchain environments or debugger script files. If such files are available for your plugin, they are located in the `source\boards` directory.

4.2.3. Functional behavior of the application example

The application example consists of:

- the five tasks:
  - `InitTask`
  - `Loop`
  - `Cyclic`
  - `Task_St1`
  - `Task_St2`
- the two alarms:
  - `AlarmActCyclic`
  - `SysCounterIncrementer`
- the resource `Res_CounterVar`
- one software counter,
- one hardware counter,
- one schedule table, and

- two applications to which all other objects are assigned to.

The auto-started task `InitTask` activates the cyclic alarm `AlarmActCyclic` and switches to the task `Loop`. This task performs an endless loop, which continuously takes and releases the resource `Res_CounterVar`.

The auto-started alarm `SysCounterIncrementer` increments the software counter, which is linked with the alarm `AlarmActCyclic`, at each alarm event.

At the appearance of an alarm event by the `AlarmActCyclic`, the task `Cyclic` is activated whose priority is higher than the priority of the task `Loop`. As soon as task `Cyclic` is activated and the resource is no longer occupied, the task `Loop` will be interrupted and the task `Cyclic` runs.

In task `Cyclic`, a variable is incremented that is used as output value by some target-specific output module.

Time units of the counter are typically dimensioned so that the task `Cyclic` is activated once per second.

Parallel to the above described behavior, a schedule table is started. This schedule table has two expiry points, each with one task activation for the task `Task_St1` and for the task `Task_St2`. Both tasks are configured with a higher priority than the `Loop` task, thus this task is interrupted over and over.

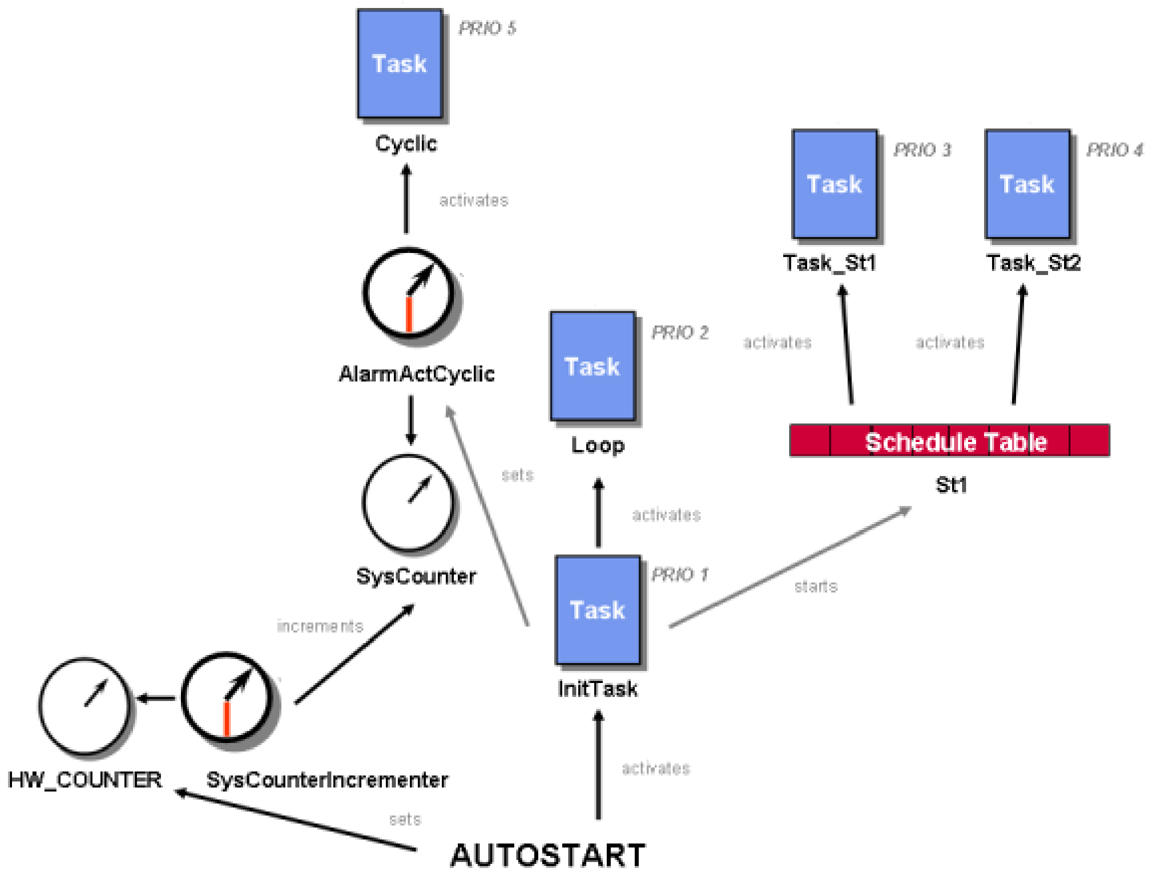


Figure 4.1. Interaction of OS objects

4.3. Setting up the application example

4.3.1. Importing the application example

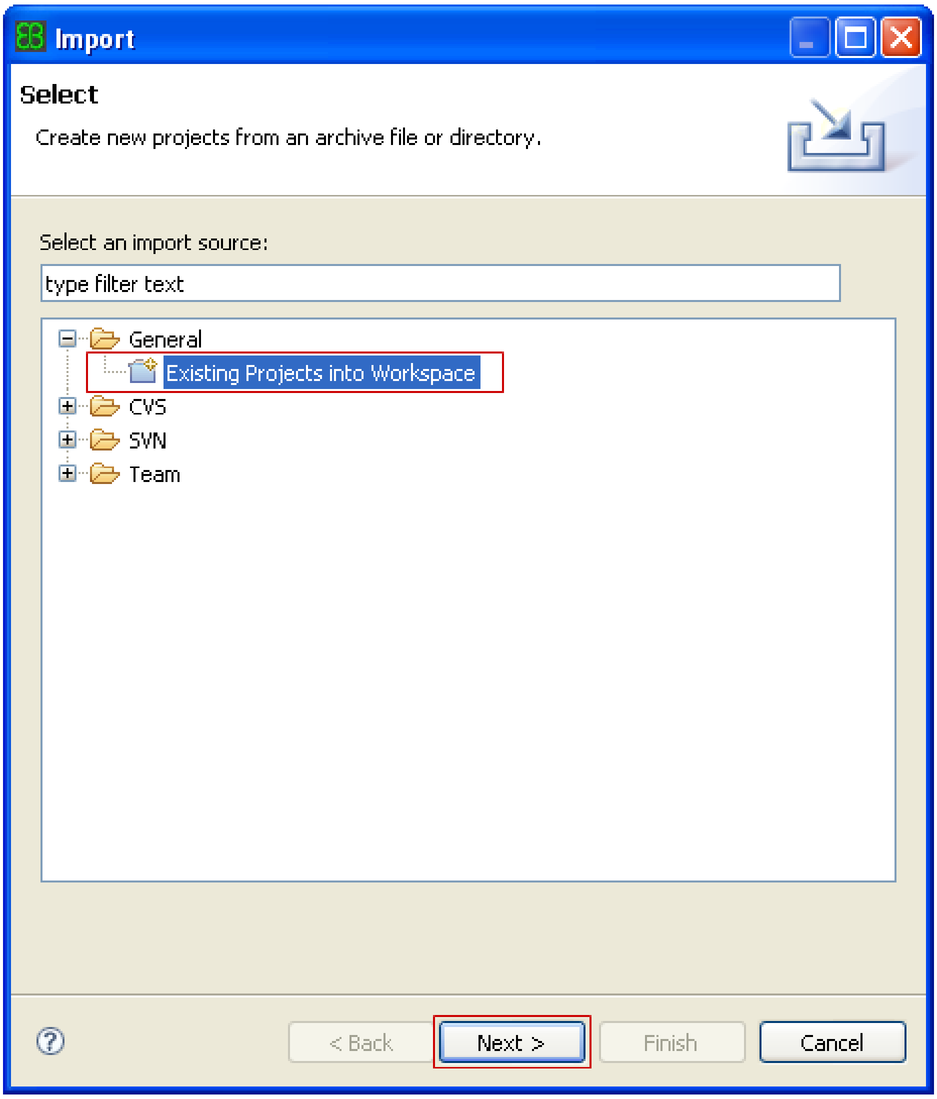
The application examples are delivered as EB tresos Studio project. You need to import the project into your EB tresos Studio workspace, e.g. at `$TRESOS_BASE\workspace`. `$TRESOS_BASE` is the directory into which you have installed EB tresos Studio, e.g. `C:\EB\tresos`.

To import the application example into your workspace:

- Locate `$TRESOS_BASE\bin\tresos_gui.exe`.
- Double-click `tresos_gui.exe`.

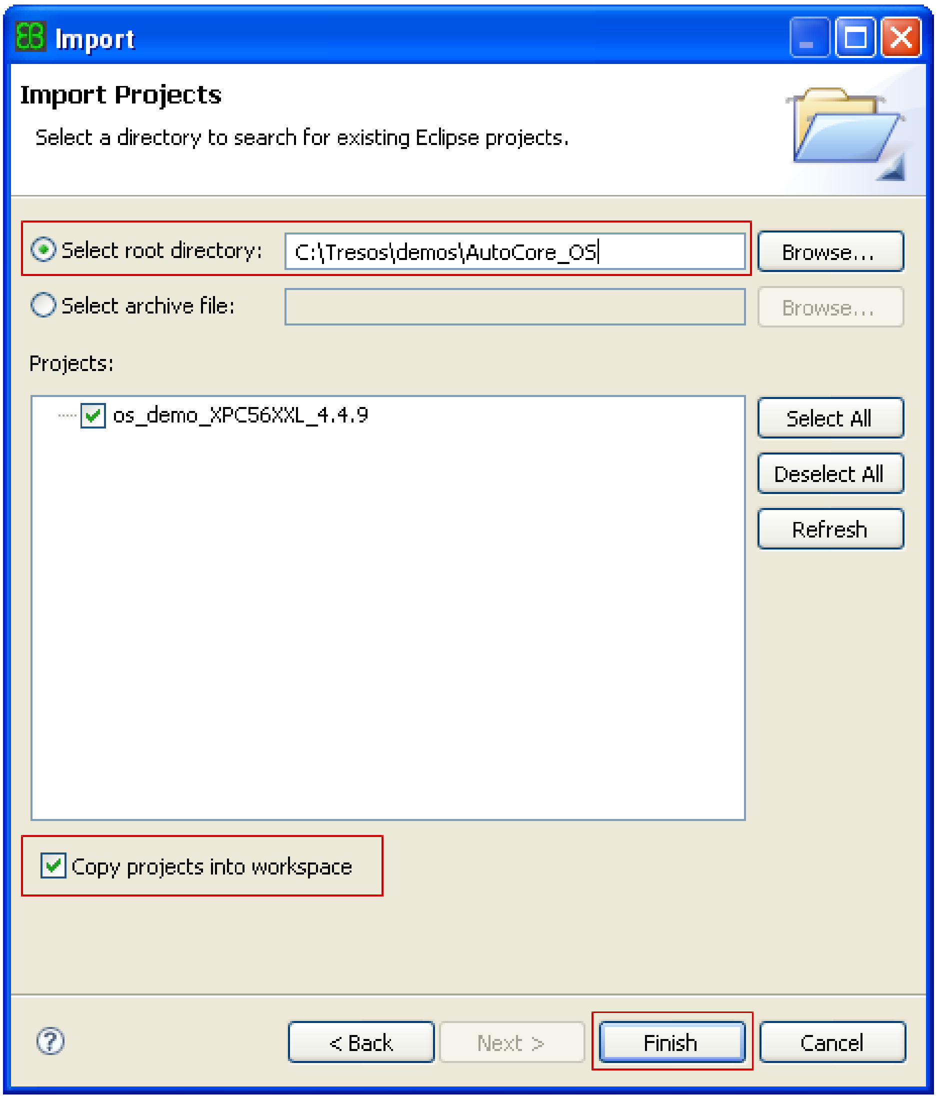
  EB tresos Studio opens up.
- In the **File** menu, select **Import**.

  The **Import** dialog opens up.



- Select **Existing Projects into Workspace**.
- Click **Next**.
- Select **Select root directory** and browse to `$TRESOS_BASE\demos\AutoCore_OS`.
- Click **OK**.

The project appears in the **Projects** window of the **Import** dialog.



The actual name of the project depends on the target platform and the `Os` version. It may look like the following: `os_multicore_demo_TC277_6.0.54`.

- Select **Copy projects into workspace**.

  The project is now copied to the default workspace at `$TRESOS_BASE\workspace`.

- Click **Finish**.

  You are done.

In the **Project Explorer** view you now see the project:

- Open it by double-clicking on the project name.
- Open the configuration by clicking the gray chip symbol .

4.3.2. Adapting the build environment

NOTE

For EB tresos WinCore, no compiler has to be configured

EB tresos WinCore is delivered with the MinGW development environment, which is installed automatically when you build the application example. The MinGW development environment also contains a compiler. Therefore, you only need to follow these instructions when you want to configure a different compiler.

To change the settings or the path of the compiler, follow the steps below:

1. Locate the project folder in the workspace directory, e.g. `$TRESOS_BASE\workspace\os_demo_$DEMO`. Place-holder `$DEMO` contains the version of your release and the name of the target hardware of this demo.
2. Open the file `util\launch.bat` in a text editor.
3. Set the environment variable `TRESOS_BASE` to the directory, into which you have installed EB tresos Studio. For example add "`SET TRESOS_BASE=C:\EB\tresos`".
4. Use variable `PLUGINS_BASE` to specify the `plugins` folder you want to use. For example add "`SET PLUGINS_BASE=%TRESOS_BASE%\plugins`".
5. Save file `util\launch.bat`.
6. Open the file `util\<target>_<derivative>_<compiler>_cfg.mak` in a text editor.
7. Change the variable `TOOLPATH_COMPILER` to the actual compiler installation path. For example: `TOOLPATH_COMPILER ?= C:\WindRiver\diab\5.5.1.0`.
8. The variable `CC_OPT` in the same file contains the compiler options. If you need any specific settings, adjust `CC_OPT`.

4.3.3. Changing the compiler

If your architecture supports multiple compilers, you can change the compiler. To change the compiler:

1. Locate the project folder in the workspace directory, e.g. `$TRESOS_BASE\workspace\os_demo`.
2. Open the file `util\<target>_<derivative>_Makefile.mak` in a text editor.
3. Set the variable `TOOLCHAIN` to the name of the toolchain, e.g. `TOOLCHAIN = dcc`.
4. Set the variable `COMPILER` to the compiler, e.g. `COMPILER = PA_XPC5777M_dcc`.

4.3.4. Changing the board settings

If you want to use a different board, you need to change the board settings.

The directory `source\boards\<board name>` in your project folder contains board-specific makefiles and source files.

In order to use a different board, follow the steps below:

1. Locate the project folder in your workspace directory, e.g. `$TRESOS_BASE\workspace\os_demo`.
2. Open the file `util\<target>_<derivative>_Makefile.mak` in a text editor.
3. Change the variable `BOARD`.

   The value of the variable has to be the same as the name of the board directory, e.g. `BOARD = EvaXPC5777M`.

4.3.5. Configuring the `Os` module

For possible adaptations of the `Os` module, see Section 5.3, "Configuring and using `Os` objects". For information on target-dependent adaptations of makefiles, refer to the EB tresos AutoCore documentation and to the EB tresos AutoCore OS architecture notes.

4.4. Building the application example

In order to build an EB tresos AutoCore project, you need to perform the following three steps in the order they are described:

1. Generate the project.
2. Create the project dependencies.
3. Build the project.

To build the application example, make sure that EB tresos Studio is not running anymore. Then follow these steps:

1. In the `$TRESOS_BASE\workspace\os_demo_$DEMO\util` directory, double-click the file `launch.bat`. The first start of `launch.bat` takes some time. Place-holder `$DEMO` contains the version of your release and the name of the target hardware of this demo.
2. Type **make generate** and hit the **Enter** key.

   The project is being generated.
3. Type **make depend** and hit the **Enter** key.

   The project dependencies are being created. This target may be obsolete in newer build environments.

4. Type **make** and hit the **Enter** key.

   Your application example is being built.

   If you work with EB tresos WinCore, the MinGW development environment is being installed with the build of the application example.

You find the resulting binary file in the `$TRESOS_BASE\workspace\os_demo_$DEMO\output\bin` directory.

4.5. Checking whether your code was built correctly

After the code of the application example was built, transfer it to your target and check whether the code was built correctly. There are two ways to check whether your code was built correctly:

- by running it on your target board, or
- with a debugging software.

4.5.1. Checking the sample code on the hardware board

For many targets the application example is customized in a way that it controls an LED panel on the board. With this LED panel you may control whether the tasks of the sample `Os` are activated correctly and how the resource is used:

- Four LEDs indicate the value of a count variable that is incremented in the `Cyclic` task.
- A fifth LED shows whether the resource `Res_CounterVar` is taken or not.

If the program is running on your target, counter-LEDs are incrementing each second and the resource LED is flashing.

NOTE

The flashing rate depends on the CPU frequency
The flashing rate of the LEDs inform you about the CPU frequency: The resource is taken and released using a delay loop. Thus, the faster the lights blink, the faster this delay loop is run.

If you use a target board the `Os` module does not support directly, you may have to adapt the board macros `LEDS_INIT` and `LEDS_SET` in the file `board.h` residing in the `source\boards` directory for your board.

For information on whether or not the `Os` module supports your board, see the respective architecture notes.

4.5.2. Checking the sample code with a debugging software

To check your sample code with a debugging software, you must load this code into the debugging software.

TIP

To start the debugging program, use the make debug command

Some target implementations support the **make debug** command that is useful for starting the debugging program, setting up the debug environment, etc. To set up your debugger with this command, double-click the file `launch.bat` in the `util` directory, type **make debug**, and press **Enter**.

For information on whether your target supports this command, see the respective architecture notes.

To check whether your code runs correctly:

1. Check whether the `Cyclic` task runs by using a breakpoint: Check whether the last 4 bits of the `counter` variable of this task are incremented each second. If they are, your code was built correctly.
2. If you use the variables `task_St1_counter` and `task_St2_counter`, you may check whether the schedule table runs correctly: If the variables `task_St1_counter` and `task_St2_counter` are continuously incremented, and the value of `task_St1_counter` is always higher than the value of `task_St2_counter`, the schedule table works correctly.

   Note that whether the value of `task_St1_counter` is higher than `task_St2_counter` is calculated independent from the data range of the counter variables. In case of an overflow, there might be situations in which the value of `task_St1_counter` is smaller (i.e. zero or negative).

When the demo is running correctly, you are ready to begin adapting it to your needs.

5. User's guide

5.1. Overview

This chapter contains instructions for how to work with EB tresos AutoCore OS. The structure of the user's guide is according to the workflow when you create your AUTOSAR Os from scratch:

- In Section 5.3, “Configuring and using `Os` objects” you find instructions for configuring `Os` objects in EB tresos Studio and for using `Os` objects in the source code.
- In Section 5.4, “Generating the code of the `Os` module” you find instructions for verifying and generating the source code of your OS.
- In Section 5.5, “Creating a linker script” you find instructions for mapping memory sections to OS sections.

In Section 5.2.2, “The directory structure” you find a description of the directory structure of your EB tresos AutoCore OS installation.

In Section 5.2.1, “Development workflow” you find a short summary of the workflow when you want to develop an AUTOSAR `Os` from scratch.

If you are an advanced user, there is additional information for you:

- Section 5.6.1, “Importing OIL/EPC files”: instructions for importing your old configuration files.
- Section 5.6.2, “Optimizing your `Os` module”: instructions for shrinking your OS.
- Section 5.6.3, “Compiling EB tresos AutoCore OS in custom build environments”: instructions for generating the EB tresos AutoCore OS in your own build environment.

5.2. Background information

5.2.1. Development workflow

To develop a complete operating system using EB tresos AutoCore OS, the following workflow applies:

- Create a configuration file in OIL, ASC or XDM format for the Generator.
- Use the Os Generator to create the source and header files that configure the kernel.
- Create source files containing all tasks, ISRs etc that are configured in the configuration file.
- Compile all source files to relocatable object files.

- Create a linker script.
- Link the object files and libraries to produce the system binary image.

The order given above is only an illustration; some of the activities can be carried out in parallel.

5.2.2. The directory structure

EB tresos AutoCore OS is intended to be used with the EB tresos Studio environment and uses by default the following directory structure. The structure can be modified by an experienced user if needed. The base directory is located at `$TRESOS_BASE\`, which is the directory into which you have installed EB tresos Studio.

`$TRESOS_BASE\bin`
: This directory contains some executables needed for the Os Generator and the make tool used to build sample applications.

`$TRESOS_BASE\doc`
: Contains the user documentation in PDF format.

`$TRESOS_BASE\plugins\Make_<variant_string>`
: This directory contains the `Make` plugin for EB tresos Studio. A compatible `Make` plugin is necessary for using the standard customer build environment.

`$TRESOS_BASE\plugins\Platforms_<variant_string>`
: This directory contains the `Platforms` plugin for EB tresos Studio. A compatible `Platforms` plugin is necessary for using the standard customer build environment.

`$TRESOS_BASE\plugins\Os_<variant_string>`
: This directory contains the `Os` plugin and is referred to as `$OS_BASE` in the following descriptions.

`$TRESOS_BASE\plugins\Os_<variant_string>_<release_suffix>`
: These directories contains release specific parts of the `Os` plugin.

`$OS_BASE\data`
: This directory contains subdirectories which hold data files for the `Os` Generator.

`$OS_BASE\make`
: The OS-specific parts of the standard customer build environment can be found in this directory. This is used to build the examples.

`$OS_BASE\src`
: Some source files are always compiled by the user because they depend on the configuration. These source files can be found in this directory. Architecture-specific source files can be found in the architecture subdirectory of this directory.

`$OS_BASE\lib_src`
: If you have a source-code license (for example in order to build optimized kernel libraries), the library source files can be found in this directory. The files are further divided into kernel files, user files, error-table files and architecture-specific files in appropriately-named subdirectories.

`$OS_BASE\include`
: The include subdirectory contains header files which are either referenced by the generated kernel or board specific files.

`$OS_BASE\application_template`
: Contains some example-code. These application examples are a good starting point when learning how to use EB tresos AutoCore OS. In the source directory of every example some board files are included as starting point.

There may be some additional directories under `$OS_BASE`, depending on the specific architecture.

EB tresos AutoCore OS does not support directories with names containing spaces. You should always ensure that the installation directories of EB tresos AutoCore OS and supporting tools (including the compiler toolchain) do not have spaces in their names.

5.2.3. The Os Generator

The Os Generator is the main tool of the development process. It is integrated in EB tresos Studio, but runs on an invisible level. The Os Generator is a tool which permits a system to be configured according to a set of demands. Configurations are specified using either the standard AUTOSAR configuration language or the XDM format.

Os configuration files can be created using a normal programmer's text editor, but the recommended method is to use EB tresos Studio, a graphical configurator plug-in. Detailed information on EB tresos Studio is available in the EB tresos Studio documentation, which you find in your folder `$TRESOS_BASE\doc\2.0_EB_tresos_Studio`.

5.3. Configuring and using `Os` objects

To configure the `Os` with EB tresos Studio, you need to add the `Os` module to your EB tresos Studio project. Instructions for adding modules to your EB tresos Studio project are available in the EB tresos Studio documentation, chapter `EB tresos Studio user's guide/Editing module configurations`.

TIP

Parameter descriptions are available

Parameter descriptions are available:

- as context sensitive help in the **Element Description** view on the lower right corner of EB tresos Studio
- as a list at Chapter 7, “References“

5.3.1. Configuring and using general parameters

5.3.1.1. Configuring general parameters in EB tresos Studio

When you start configuring a new operating system, it is recommended to start with the configuration of the general parameters of the `Os` module. To configure the general parameters of the `Os` module:

1. Open the **OsOS** tab of the **OS (Os)** editor:

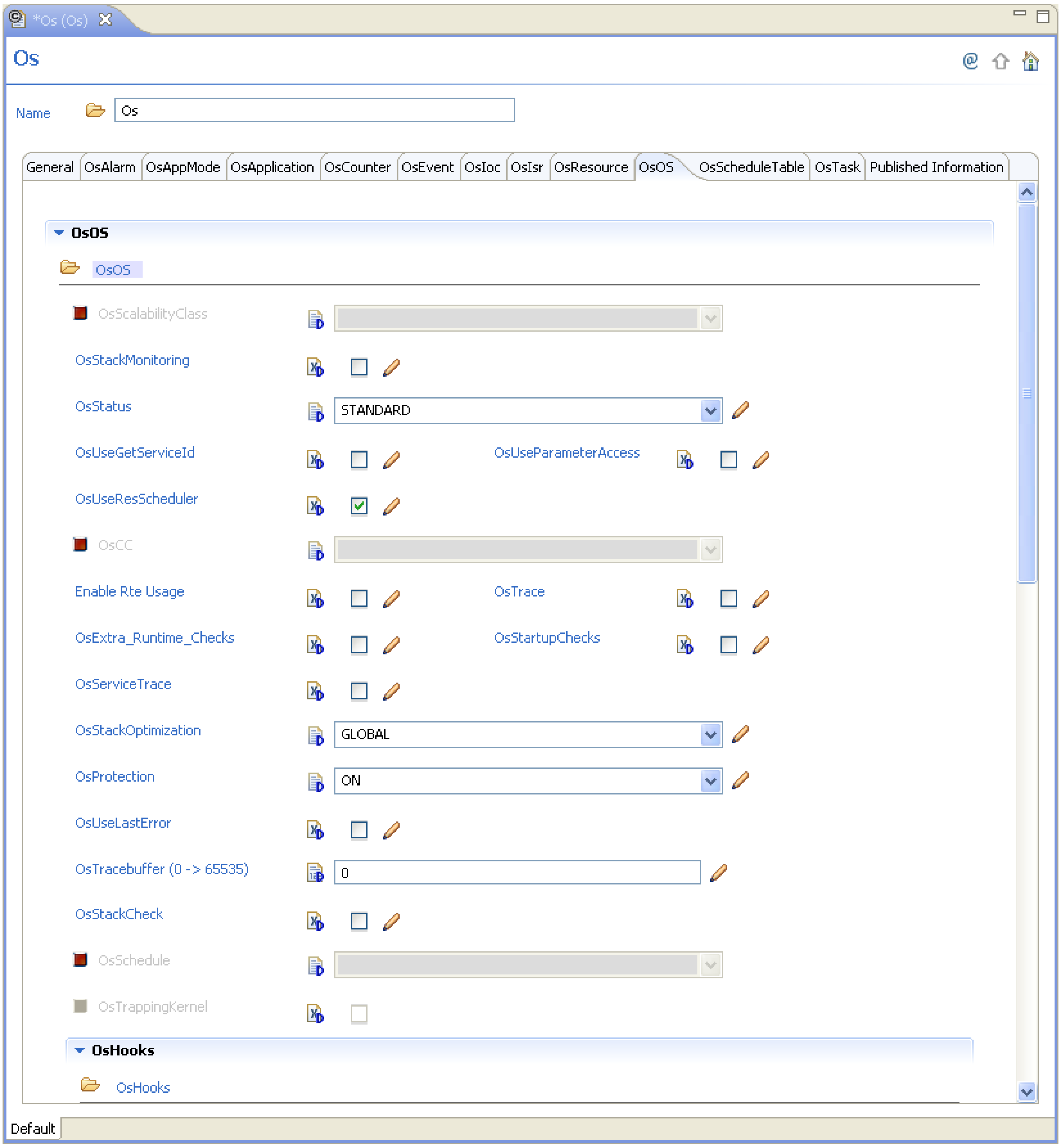


Figure 5.1. Configuring parameters of the `OsOS` container

2. Configure the parameters to your needs. For information about the single parameters, see Section 7.1, “EB tresos AutoCore OS Configuration Language”.

   Detailed information about the parameters of the `OsAutosarCustomization` container is available at Section 5.6.2.2, “Enabling kernel customizations”.

5.3.2. Adding `Os` objects in EB tresos Studio

For each type of `Os` object, there is a corresponding list in the configuration. To add `Os` objects to your configuration:

- Add an entry in the corresponding list.
- Fill in the parameters for the entry.

For example, to add a task to your `Os` configuration:

- Open the **OsTask** tab of the **OS (Os)** editor:

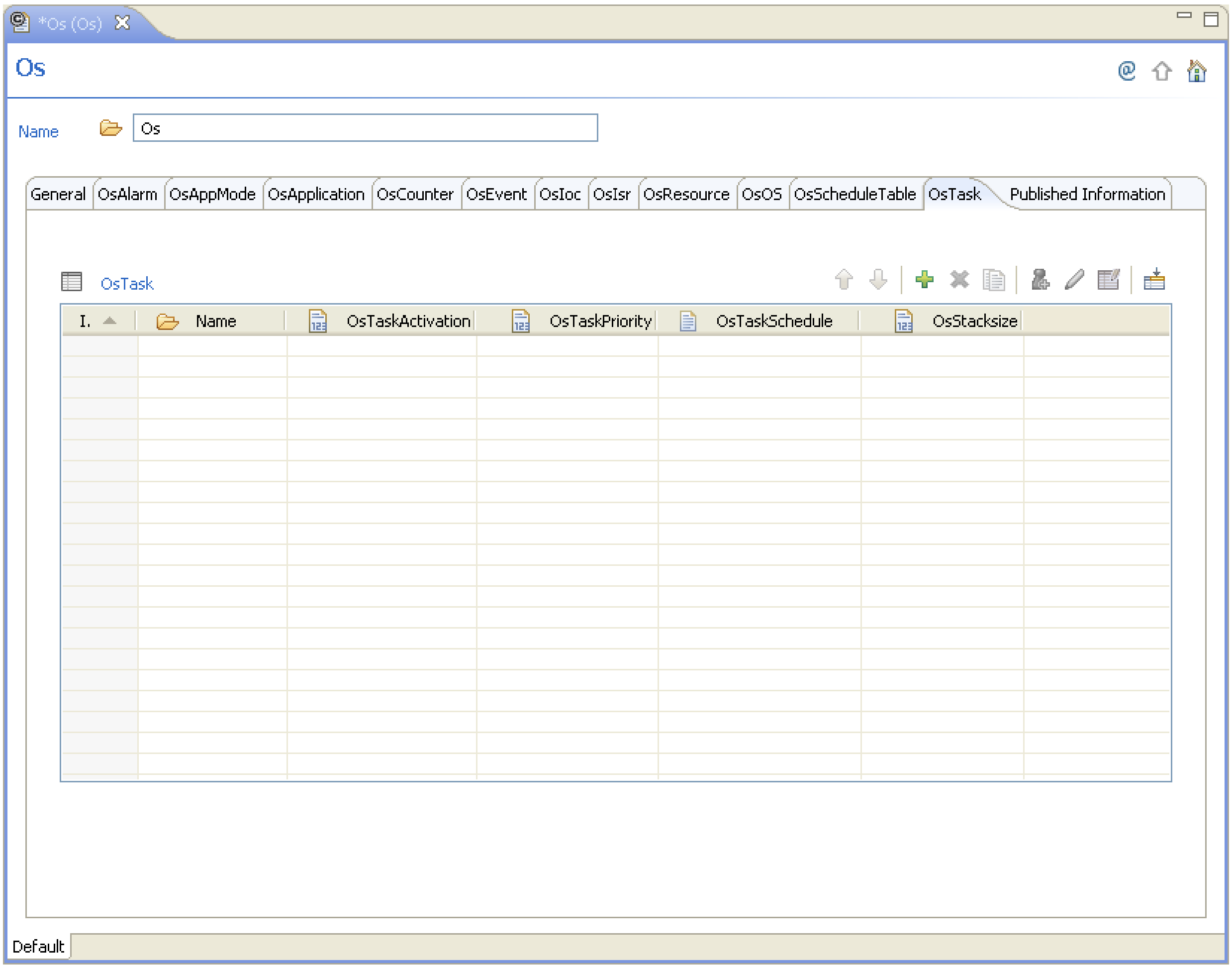


Figure 5.2. Adding tasks to your `Os` configuration

- To add a task with default values, click .

  A new task is added to the **OsTask** table in the **Os (OS)** editor.
- To configure the most important parameters for the task, use the list view:

- In the **Name** column, edit the name of the task.
- In the **OsTaskActivation** column, edit the maximum number of activations this task can have at one point in time.
- In the **OsTaskPriority** column, assign a priority to the task.
- In the **OsTaskSchedule** column, select whether the task shall be preemptive or non-preemptive.

  In the drop-down list box, select:

  - `FULL` for preemptive tasks.
  - `NON` for non-preemptive tasks.
- In the **OsStackSize** column, edit the stack size of the task.

- To configure other task parameters, double-click in the **Index** column of the task.

  The **OsTask** editor for the task is started. In this editor, you can change all available configuration parameters according to your needs. To find out the semantics for each parameter:

  - Use the context sensitive help in the **Element Description** view on the lower right corner of EB tresos Studio.
  - Use the list at Chapter 7, “References“.

5.3.3. Configuring the Os for use with Microkernel

If your Os release was built to support acting as QM-OS, forming the EB tresos Safety OS together with the Microkernel (module `MicroOs`), the **OS (Os)** editor contains a tab called **OsMicrokernel**. It is used to configure the general parameters of the `MicroOs` module. Code generation for the `MicroOs` module is disabled by default. To enable code generation and to configure the general parameters of the `MicroOs` module:

1. Open the **OsMicrokernel** tab of the **OS (Os)** editor.
2. Enable the **OsMicrokernel** container.

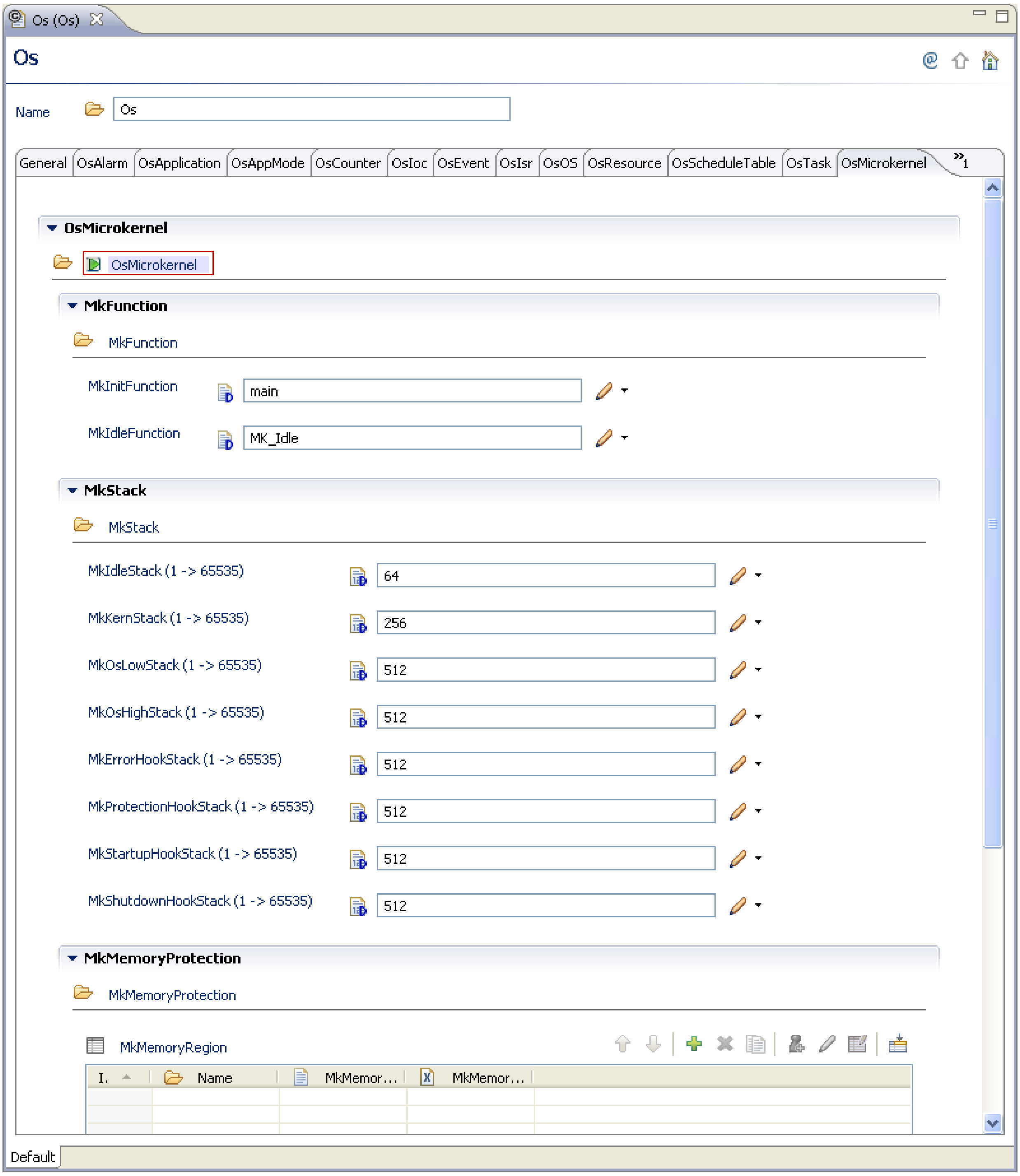


Figure 5.3. Enabling the `OsMicrokernel` container

3. Configure the parameters to your needs. For information about the configuration of the Microkernel, see Section 7.1.3, “Configuration parameters for the Microkernel” and to the Microkernel Users's Guide and the Safety Manual.

5.4. Generating the code of the `Os` module

After you have configured your `Os` module, you need to generate the code out of it. You may either generate the code output with EB tresos Studio or with the command line. If you generate the code with EB tresos Studio, you can verify the code before generating it.

5.4.1. The location of the makefiles

The AUTOSAR standard build environment uses a set of two configuration makefiles for each module. Additionally, the shipment contains a set of compiler plugins for the supported toolchains and a set of plugins for the configuration environment EB tresos Studio. The EB tresos AutoCore OS specific plugin files are located in `$OS_BASE\make`:

`Os_defs.mak`
: The `Os_defs.mak` file describes all files that need to be built, directories that must be created and where output files are placed. It defines all generic files that are part of the OS-libraries. Architecture- and derivative-specific files are included from this file. They define the extra files that are needed for each architecture and derivative.

`Os_rules.mak`
: The `Os_rules.mak` file contains rules concerning the OS, e.g. the **clean** rule. The generation rule is part of the EB tresos Studio plugin files.

In addition to these files, the following file is located in the project's `util` directory:

`Os_cfg.mak`
: The `Os_cfg.mak` file is part of the application and contains options for building the OS.

5.4.2. Generating code with EB tresos Studio

To generate or verify the code of your project with EB tresos Studio:

- in EB tresos Studio, select your project in the **Project Explorer** view.
- To verify the configuration of your project, click on the **Verify** button in the toolbar of EB tresos Studio.

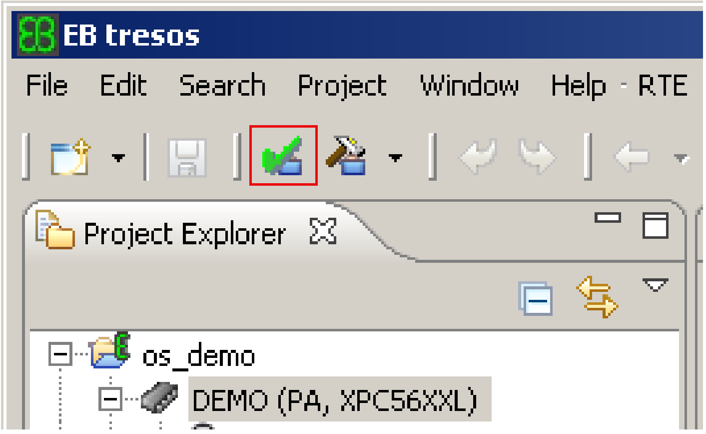


Figure 5.4. Verifying the configuration in EB tresos Studio

- To generate code for your project, click on the **Generate** button in the toolbar of EB tresos Studio.

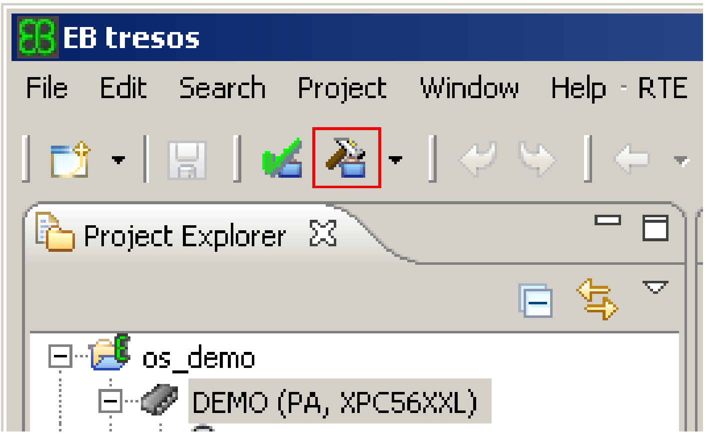


Figure 5.5. Generating code in EB tresos Studio

Per default, the code is generated into the folder `<INSTALL_PATH>\workspace\<project_name>\output`.

For further information on generating code with EB tresos Studio, see the EB tresos Studio documentation.

5.5. Creating a linker script

The linker script defines the placement in memory of all text and data sections, and defines symbols that are used by the kernel and startup code to locate the sections for initialization and memory protection purposes.

If the system you are developing does not need to use memory protection, the standard linker script for the board and toolchain can be used. This linker script places all `.text` sections and `.rodata` sections together in

ROM and all `.data` sections and `.bss` sections together in RAM. The `.data` and `.bss` sections are initialized at startup by the board-specific startup code.

For systems using memory protection it is necessary to create a custom linker script. The script will gather together the code and data sections that belong to applications, and will define symbols that the kernel needs in order to find these sections. You may either generate such a linker script using a supplied Perl program, or create the linker script by hand. This is described in the following sections. To optimize your linker script, you may want to have a look at some hints for tuning the linker script in Section 5.5.4, "Tuning the linker script for memory protection"

5.5.1. Generating the linker script with Perl

The EB tresos AutoCore build environment already includes rules to generate a linker script for projects using memory protection. These rules are automatically enabled as soon as the appropriate parameters for memory protection are configured. Linking, especially the mapping between protected Os objects and their memory regions, is based on the object file names.

To inform the linker script generator about your implementation, you need to map your object files to Os objects by providing the following variables in your `Makefile`:

`CC_FILES_TO_BUILD`
: holds all files to build for your project. Here, the source files for your Os objects must be added.

`OBJS_`*xxx*
: holds the names of all object files holding code or data for Os object *xxx*.

For example, let's assume you have configured a non-trusted Os application called `App`, consisting of two tasks named `FooTask` and `BarTask`. For each task you have a corresponding C file containing its code and the task-private data. You want both tasks to share some application-private data; the C definition of the corresponding variables is in the file `Appdata.c`. In your `Makefile`, you would add the following section:

```
CC_FILES_TO_BUILD += $(PROJECT_ROOT)\source\Appdata.c \
                     $(PROJECT_ROOT)\source\FooTask.c \
                     $(PROJECT_ROOT)\source\BarTask.c

                     OBJS_App = Appdata.$(OBJ_FILE_SUFFIX)
                     OBJS_FooTask = FooTask.$(OBJ_FILE_SUFFIX)
                     OBJS_BarTask = BarTask.$(OBJ_FILE_SUFFIX)
```

Then you'd call `make`, and the Perl linker script generator would be invoked. Its output is put in your project's output directory, usually at `$(PROJECT_ROOT)/output/generated`.

5.5.2. Creating a linker script by hand

This section assumes that you are familiar with your toolchain and have knowledge about writing linker scripts. It describes the linker symbols expected by EB tresos AutoCore OS. This description features commonly used symbols; see your architecture notes for hardware specific features.

In all of the following descriptions, a *start* address of a region specifies the first address of that region and an *end* address specifies the first address (greater than or equal to the *start* address) that does not belong to the region.

The kernel and the associated startup code provided with the kernel expects the linker script to define the following symbols:

`__STARTDATA`
: marks the start of the global data section.

`__ENDDATA`
: marks the end of the global data section.

`__STARTBSS`
: marks the start of the global bss section.

`__ENDBSS`
: marks the end of the global bss section.

`__INITDATA`
: marks the start of the rom image for the global data section.

The above symbols are only used by the default startup code in `board.c`. If you are not using the default startup code these symbols do not need to be defined. If you use the default startup code but nevertheless provide your own memory initialization code (or use the compiler's startup code), set these symbols to NULL (0) or in case of the KEIL ARM® toolchain, create empty execution regions with `0x0` as start address.

The following symbols are required, when OsOS/OsTrappingKernel is enabled:

`__GLBL_TEXT_START`
: marks the start of the program text section.

`__GLBL_TEXT_END`
: marks the end of the program text section.

The above symbols are used by the kernel to set up the code protection registers for all non-trusted applications. All memory between the two symbols is executable, the rest is non-executable.

`__GLBL_RODATA_START`
: marks the start of the read-only data (constants, C strings etc.) in ROM.

`__GLBL_RODATA_END`
: marks the end of the read-only data.

`__GLBL_DATA_START`
: marks the start of the variable data and bss.

`__GLBL_DATA_END`
: marks the end of the variable data and bss.

The above symbols are used by the kernel to grant non-trusted applications read-only access to ROM data and data belonging to other applications. All memory between the two symbols in each pair is readable. On processors such as TriCore that have a limited number of regions all memory between the lesser of the two START symbols and the greater of the two END symbols delimit a single read-only region for all non-trusted applications.

`__DATA_`*xxx*
: marks the start of the private data and bss belonging to the non-trusted APPLICATION, TASK or ISR named *xxx*.

`__DEND_`*xxx*
: marks the end of the private data and bss belonging to the non-trusted APPLICATION, TASK or ISR named *xxx*.

The above symbols mark the private data belonging to the named object. The data belonging to an APPLICATION is readable and writeable by all TASKs and ISRs in that application. The data belonging to a TASK or ISR is readable and writeable only by that TASK or ISR. Other TASKs, ISRs and APPLICATIONS gain read-only access through the global region (see above). The linker must define these symbols for each APPLICATION, and for each TASK and ISR. Setting these symbols to NULL (0) indicates that the named object has no private data.

Note, that the kernel does not use these symbols for trusted APPLICATIONs and for each TASK and ISR that belongs to a trusted APPLICATION. For trusted applications, their TASKs and ISRs, you should define these symbols to NULL (0).

`__IDAT_`*xxx*
: marks the start of the initialization data for the non-trusted APPLICATION, TASK or ISR named *xxx*.

`__IEND_`*xxx*
: marks the end of the initialization data for the non-trusted APPLICATION, TASK or ISR named *xxx*.

These symbols are used by the code that initializes the private data areas during `StartOS()`. The ROM image from `__IDAT_`*xxx* to `__IEND_`*xxx* is copied to the addresses `__DATA_`*xxx* and so on. The RAM data region must therefore be bigger than or equal in size to the ROM image. Any remaining portion of the `__DATA_`*xxx* region is set to zero. It is therefore assumed that the linker places all `.data` sections belonging to the object into the area, followed by all `.bss` sections. Setting these symbols to NULL (0) has no special meaning. If the symbols are equal, no data will be copied, but the entire `__DATA_` area will be set to zero. If you wish to provide your own initialization code or use that provided by the compiler, it is therefore necessary to override the kernel's initialization of private data areas. This can be achieved by overriding the kernel's initialization function with an empty stub, i.e. by linking the following code:

```
void OS_InitApplicationData(void)
{
}
```

The Perl scripts provided with EB tresos AutoCore OS create a linker script with a default memory layout and define all these symbols accordingly. However, the default layout will not suit all systems. You can write your own layout program based on the scripts provided, or simply create a linker script manually.

5.5.3. Support for the KEIL ARM® toolchain

To support the KEIL ARM® toolchain within the EB tresos AutoCore OS a slightly different approach was implemented. This subchapter describes how the needed symbols can be set using the `armlink` linker and what rules should be followed to satisfy the EB tresos AutoCore OS.

Because of the lack to set additional global symbols within the `armlink` linker script, the auto generated linker symbols, for set execution regions, must be used. Therefore the EB tresos AutoCore OS relies on a corresponding naming of created execution regions inside of the linker script file.

If the provided startup code is used, following names for the global data and bss section must be used:

`data_DATA`
: contains all global data objects

`bss_DATA`
: contains all global bss objects

The linker automatically creates the following symbols for the above mentioned execution regions which are used within the startup code to copy the data and initialize the bss section:

`Image$$data_DATA$$Base`
: marks the start of the global data section

  also available as preprocessor define: `OS_TOOL_STARTDATA`

`Image$$data_DATA$$ZI$$Limit`
: marks the end of the global data section

  also available as preprocessor define: `OS_TOOL_ENDDATA`

`Load$$data_DATA$$Base`
: marks the start of the rom image for the global data section

  also available as preprocessor define: `OS_TOOL_INITDATA`

`Image$$bss_DATA$$Base`
: marks the start of the global bss section

  also available as preprocessor define: `OS_TOOL_STARTBSS`

`Image$$bss_DATA$$ZI$$Limit`
: marks the end of the global bss section

  also available as preprocessor define: `OS_TOOL_ENDBSS`

For the OsOS/OsTrappingKernel support the symbols already mentioned in Section 5.5.2, "Creating a linker script by hand" must be provided. At the KEIL ARM® toolchain this can be achieved by creating empty excution regions at appropriate spots. Since the symbols are only used to setup the memory protection and not to copy any data, this is totally sufficient.

Therefore empty execution sections with the following names must be created to generate the expected symbols which are named below.

`__GLBL_TEXT_START`
: empty execution section used to mark the start of the text section

`__GLBL_TEXT_END`
: empty execution section used to mark the end of the text section

`__GLBL_RODATA_START`
: empty execution section used to mark the start of the rodata section

`__GLBL_RODATA_END`
: empty execution section used to mark the end of the rodata section

`__GLBL_DATA_START`
: empty execution section used to mark the start of the variable data and bss sections

`__GLBL_DATA_END`
: empty execution section used to mark the end of the variable data and bss sections

From the above specified execution sections the linker creates the following symbols which are used within the EB tresos AutoCore OS:

`Load$$__GLBL_TEXT_START$$Base`
: marks the start of the program text section

  also available as preprocessor define: `OS_TOOL_TEXT_START`

`Load$$__GLBL_TEXT_END$$Base`
: marks the end of the program text section

  also available as preprocessor define: `OS_TOOL_TEXT_END`

`Load$$__GLBL_RODATA_START$$Base`
: marks the start of the read-only data (constants, C strings etc.) in ROM

  also available as preprocessor define: `OS_TOOL_RODATA_START`

`Load$$__GLBL_RODATA_END$$Base`
: marks the end of the read-only data

also available as preprocessor define: `OS_TOOL_RODATA_END`

`Image$$__GLBL_RAM_START$$Base`
: marks the start of the variable data and bss

  also available as preprocessor define: `OS_TOOL_RAM_START`

`Image$$__GLBL_RAM_END$$Base`
: marks the end of the variable data and bss

  also available as preprocessor define: `OS_TOOL_RAM_END`

Also the naming of execution sections for non-trusted APPLICATIONs, TASKs, and ISRs must follow a specific structure. In the case the data and bss regions are split in two sections for each of the provided functions. Nevertheless it does not mean that the sections can be located separately in memory. The bss section must always follow the data section.

To achieve the right initialization of this section, the following name scheme must be used:

`data_`*xxx*
: the name of the data section must be prefixed by the word `data_` following the name of the APPLICATION, TAKS or ISR (*xxx*)

`bss_`*xxx*
: the name of the bss section must be prefixed by the word `bss_` following the name of the APPLICATION, TAKS or ISR (*xxx*)

From this sections the linker creates the following symbols which are used to copy the data and to initialize the bss section:

`Image$$data_`*xxx*`$$Base`
: marks the start of the private data and bss belonging to the non-trusted APPLICATION, TASK or ISR named*xxx*.

`Image$$bss_`*xxx*`$$ZI$$Limit`
: marks the end of the private data and bss belonging to the non-trusted APPLICATION, TASK or ISR named *xxx*.

`Load$$data_`*xxx*`$$Base`
: marks the start of the initialization data for the non-trusted APPLICATION, TASK or ISR named *xxx*.

`Load$$data_`*xxx*`$$ZI$$Limit`
: marks the end of the initialization data for the non-trusted APPLICATION, TASK or ISR named *xxx*.

TIP

load and execution address

Take care of the difference between the load (prefix `Load`, normally the address at ROM) and execution address (prefix `Image`, normally the address at RAM) to achieve a successful copy process within the EB tresos AutoCore OS.

5.5.4. Tuning the linker script for memory protection

The memory layout generated by the Perl program works and provides near-optimal protection. However, this comes at the cost of potentially leaving large areas of memory unused and unusable.

In most real applications it is therefore necessary to tune the linker script generation process, or hand-tune the linker script, to provide the best possible protection within the limitations of the processor or board. The following paragraphs give a few hints on how to optimize the configuration of the OS and the linking process without seriously compromising the protection.

5.5.4.1. Sharing data within an application

If it is not really necessary to protect tasks and ISRs within an application from each other, the private data for all the tasks and ISRs in an application can be placed in the same page as the application's own private data. This can be achieved by two methods:

- either list all the files that belong to the tasks and ISRs as belonging to the application, or
- modify the linker script and place all the task and ISR data sections in the same page as the application's data.

5.5.4.2. Restricting the task and ISR stack-sharing

If the Os Generator shares stacks among all objects that do not preempt each other, the stacks need to be placed in their own pages. This is necessary to prevent a task or ISR from gaining write access to another application's data through its stack permissions. If, however, sharing is restricted to within applications using the configuration option, the task and ISR stacks for applications can be placed in the same page as the application's data. This reduces the effectiveness of the stack-overflow protection, but this can be mitigated somewhat by placing the stack at the bottom of the page. In any case, stack over- and underflow will only affect the application and not the whole system.

5.5.4.3. Placing the hook stacks all in the same page

The kernel allocates two stacks for the hook functions of non-trusted applications: one for startup and shutdown hooks, and one for error hooks. On analysis, it will be clear that these can never be used simultaneously by 2 applications, so it is safe to place them both in the same page.

5.5.4.4. Placing the kernel stack and data in the same page

The kernel stack is best placed at the bottom of the available RAM so that if a kernel stack overflow occurs, the processor will trap to an exception handler. The kernel's data can be placed in the same page as the kernel's stack. However, many linkers process their linker script serially, so the selection of data from remaining files must appear at the end of the script. Without knowing beforehand how much data is used outside non-trusted applications, it is impossible to reserve a number of pages for the data of trusted code, so the linker script places kernel stack and kernel data together at the upper end of the allocated memory.

When the characteristics of the system are better known, it should be possible to place the kernel stack and non-trusted data lower in memory.

5.5.4.5. Taking care of the alignment of memory regions

If a memory region covers 4 or more minimum-size pages (i.e. if it is bigger than 12 kB), the number of Translation Look-aside Buffer (TLB) entries required can change depending on the alignment of the region. A region of between 12 and 16 kB aligned on a 16 kB boundary only needs one TLB entry. If it is aligned on an odd 4 kB boundary or an odd 8 kB boundary it will require 4 TLB entries. Thus larger memory regions should be aligned with care.

5.5.5. Mapping the memory using `Memmap.h`

EB tresos AutoCore OS supports the standard AUTOSAR memory mapping mechanism provided by `MemMap.h`. Since placing the various sections of the OS to specific memory regions is a very crucial task on many architectures (i.e. for CPUs which have protection mechanisms or which use banked memory), the use of `MemMap.h` is disabled by default.

NOTE

Mapping via `Memmap.h` usually not necessary

For normal use cases, mapping the memory using `Memmap.h` is not necessary. For example, in a protected system using a linker script like explained above, it is sufficient to map the memory based on the names of the object files.

To enable the `MemMap.h` support, compile EB tresos AutoCore OS with the macro `OS_MEMMAP` set to `1`.

5.6. Advanced configuring

5.6.1. Importing OIL/EPC files

Instead of configuring your `Os` module from scratch, you may also import your existing OIL or EPC files. To import an OIL or EPC file into your EB tresos Studio project:

- Open EB tresos Studio.
- Load your project you want to import to or create a new project.
- Right-click the gray chip symbol of your project.

  A context menu opens up.

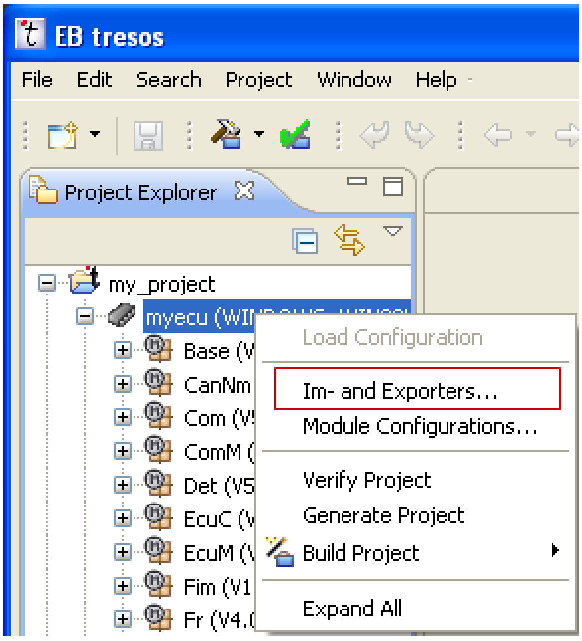


Figure 5.6. Importing an OIL/EPC file into your EB tresos Studio project

- In the context menu, select **Im- and Exporters...**.

  The **Create, manage and run im- and exporters** dialog opens up.

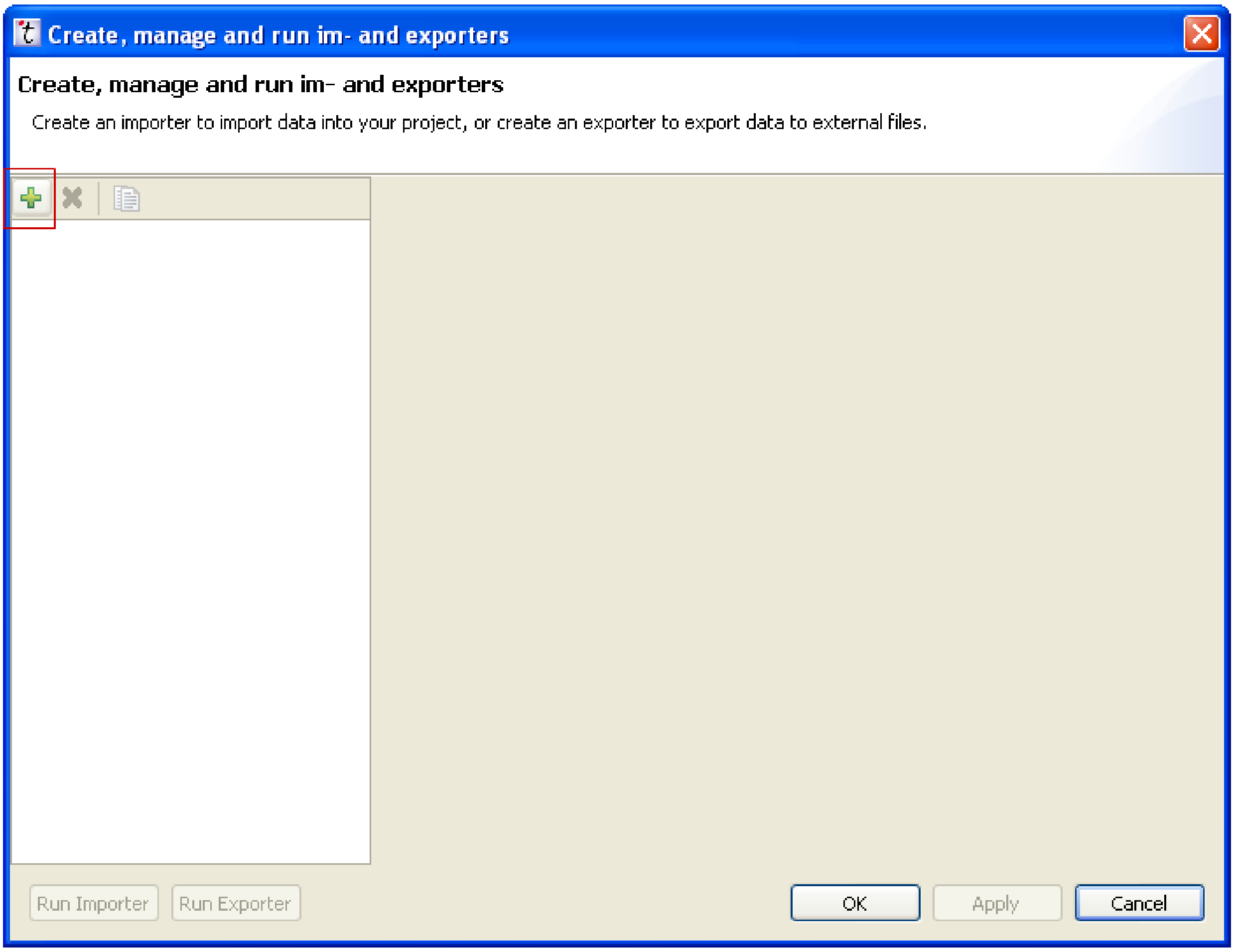


Figure 5.7. Opening the **New Importer/Exporter** dialog

- To add an importer, click the  button.

  The **New Importer/Exporter** dialog opens up.

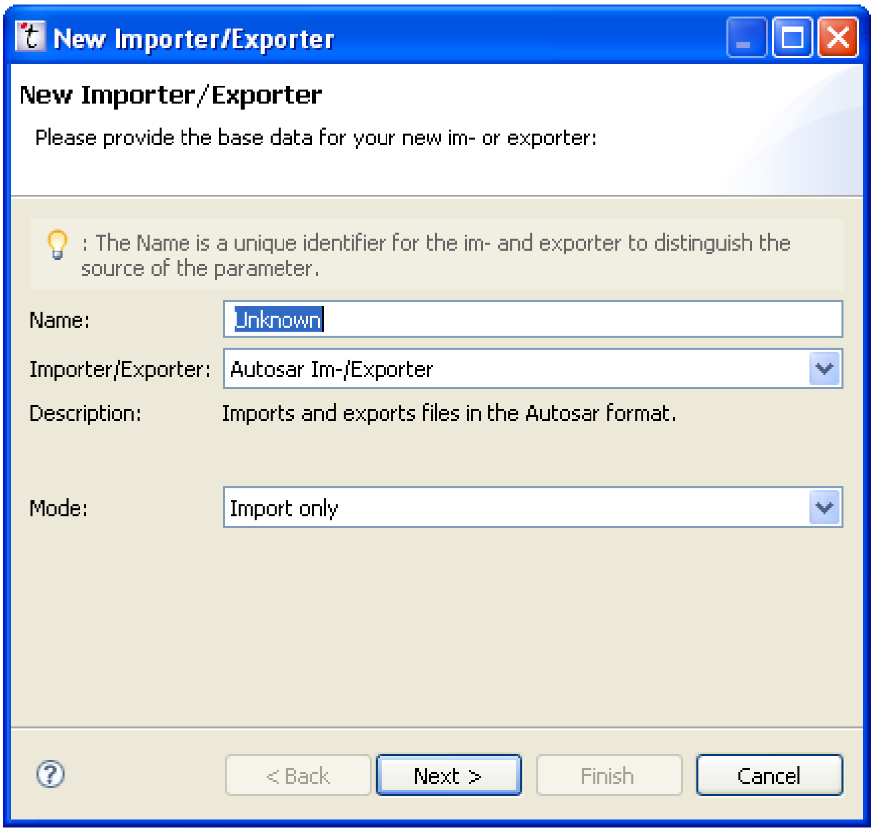


Figure 5.8. Configuring your importer: file format and the importer's name

- In the **Name** text box, enter a name of your new importer.
- In the **Importer/Exporter** drop-down list box:
  - To import an EPC file, select **Autosar Im-/Exporter**.
  - To import an OIL file, select **Oil Importer**.
- Click **Next**.

  The **<fileformat> Importer Options** page of the **New Importer/Exporter** dialog opens up.

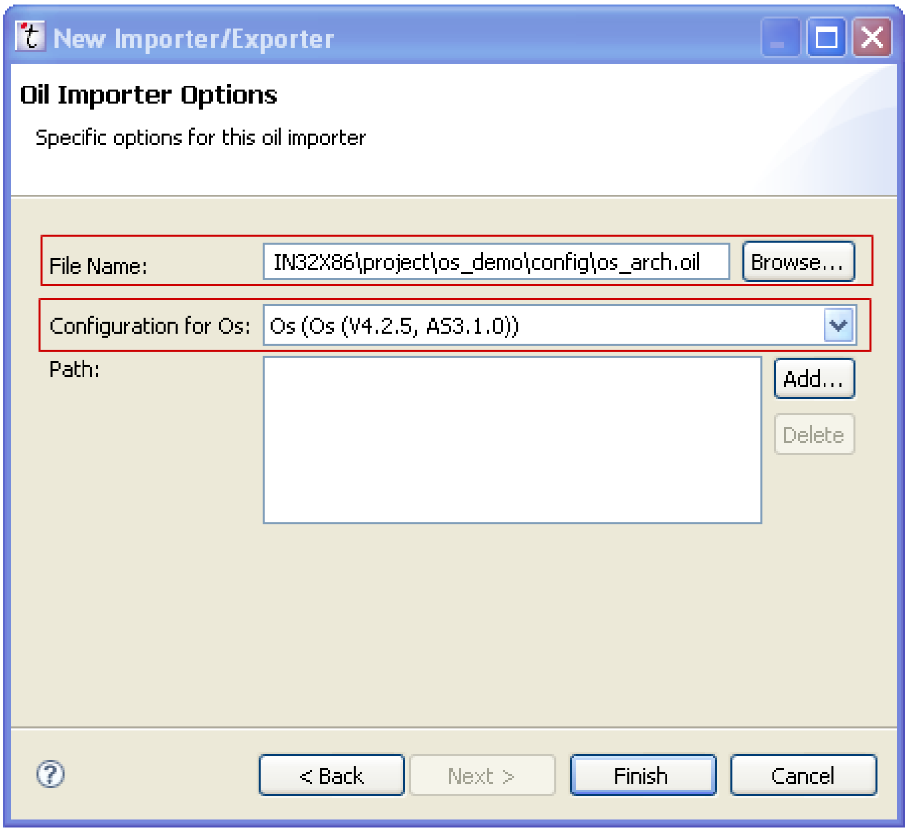


Figure 5.9. Configuring your importer options: selecting the EPC/OIL file and AUTOSAR version

- In the **File Name** text box, browse to your EPC/OIL file.
- In the **Configuration for Os** drop-down list box, select the AUTOSAR version of your Os configuration.
- Click **Finish**.

  You importer is being created.

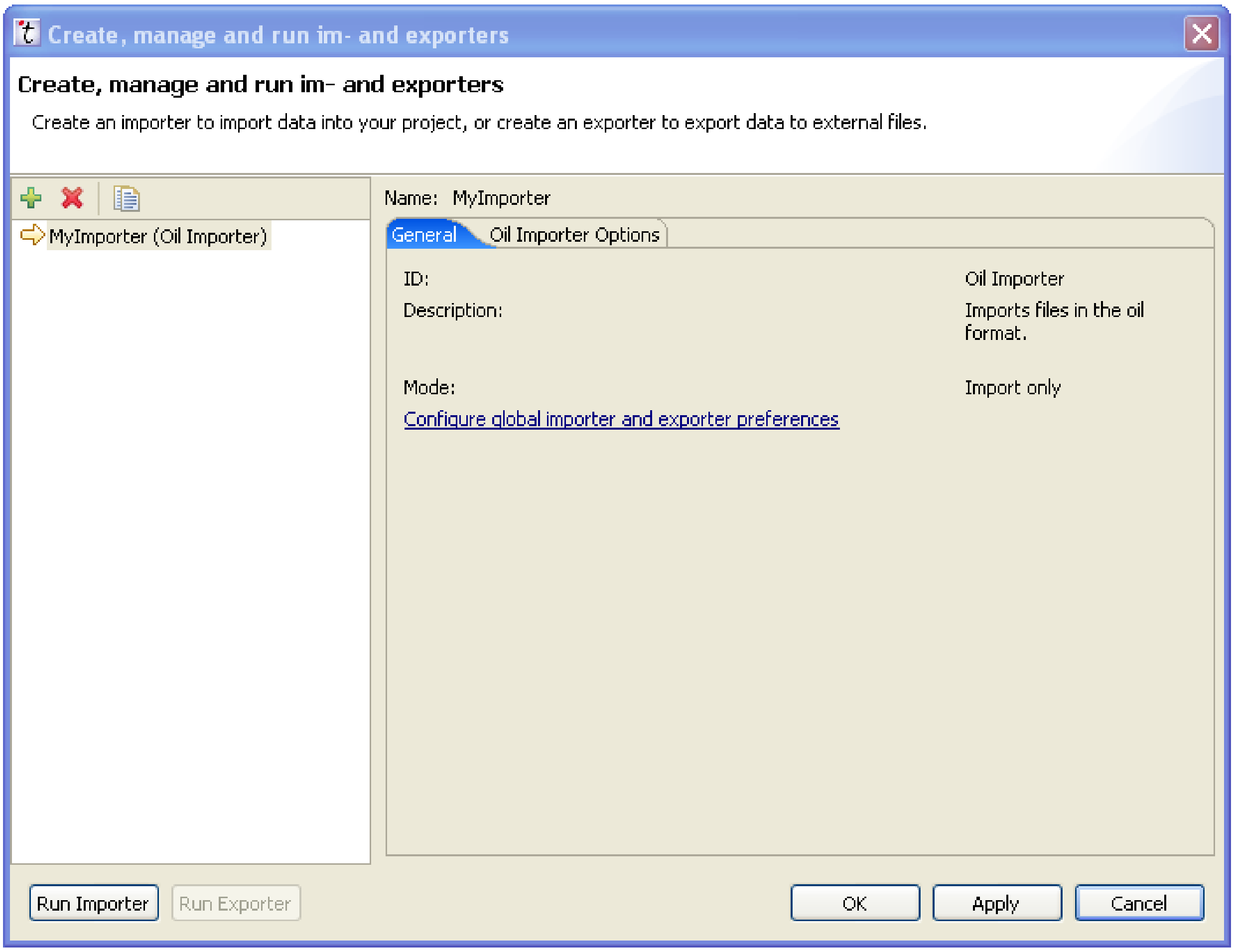


Figure 5.10. Finishing your importer configurations

- Click **Run Importer**.

  If you have not saved your importer configuration yet, a confirmation dialog opens up asking you whether you want to save the changes of your importer configuration.

  The importer runs and imports your selected file.

- If errors occur during the import, a feedback dialog that informs you about these errors opens up.

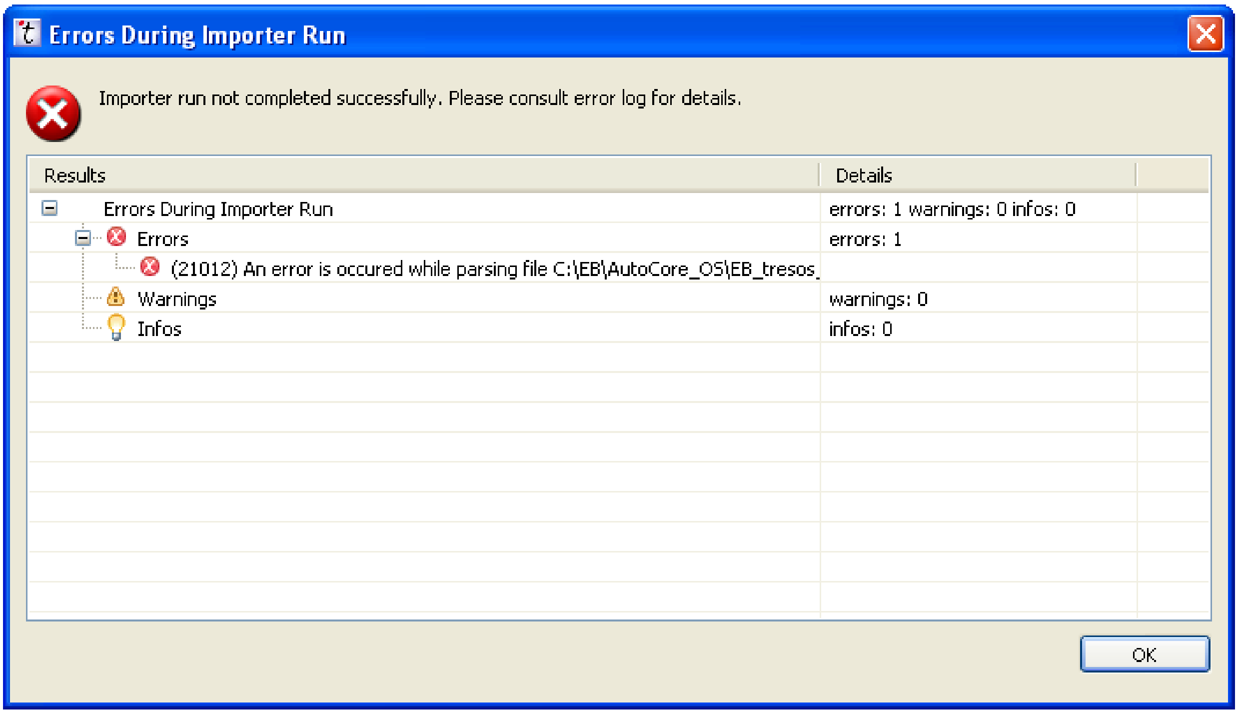


Figure 5.11. Getting feedback from the import process

5.6.2. Optimizing your `Os` module

EB tresos AutoCore OS is highly configurable. Even when using the standard library, all the standard functionality of the standard AUTOSAR Os, right up to scalability class 4, is available. The disadvantage of this approach is that the kernel is often much bigger and slower than it needs to be for a given ECU, even with all the techniques that are used in the kernel to avoid linking unnecessary functions and data into the final binary. As a countermeasure, it is possible to build customized libraries tailored to your configuration.

There are three ways to optimize your `Os` module:

- Section 5.6.2.1, “Optimizing the library”: make your kernel smaller and thus faster.
- Section 5.6.2.2, “Enabling kernel customizations”: activate optimization options in the configuration.

5.6.2.1. Optimizing the library

The EB tresos AutoCore OS is built as a library. This means that functions and data that are not referenced will normally not be linked into the final binary. The configurability is achieved by making extensive use of decisions based on external (ROM) constants, and function pointers that can be redirected to null (empty) functions. This method of construction means that we can minimize the size of the kernel in case the operating system is only compiled once and is used as a generic library, but it means that at each decision there is code for both the true and false cases even though the decision always follows the same branch for a given configuration. For function pointers, the overhead of calling the null function and returning from it is still present. So you can eliminate all the unnecessary decisions and unneeded function calls from the final kernel. The way you achieve this is by building a customized library that is exactly tailored to a given configuration. The optimizations are determined from the OS configuration. Many of them come from the standard configuration, but there is a set of OS customization options that can result in a smaller, faster kernel with the possible loss of some AUTOSARconformance.

If you want the kernel to be smaller or faster, you need an optimized build. Depending on the target processor and the configuration, an optimized kernel can be as small as around 30% of the size of a standard kernel. There will also be performance gains, although not as dramatic.

If compile time is a problem, rather use a standard library in the early stages of a project, when the configuration is undergoing change. If optimization is used extensively while the configuration is changing, lots of customized libraries will be generated. The OS's library directory will fill up with the different library versions and the corresponding object files, but these can be deleted if disk space becomes a problem.

5.6.2.1.1. Building an optimized library with the EB tresos AutoCore OS build environment

To build an optimized library with the EB tresos AutoCore OS build environment:

- Open the **OsOS** tab of the **OS (Os)** editor.
- Check the **OsSourceOptimization** switch.
- Generate the project.

As a result, the make variable `OS_BUILD_OPTIMIZED_LIB_FROM_SOURCE` is set to `TRUE` in the file `Os_objects.make`. The C preprocessor macro `OS_USE_OPTIMIZATION_OPTIONS` is defined as `1` in the file `Os_libcfg.h`. This causes the definitions of the `OS_EXCLUDE_something` macros in the same file to be enabled. Both files are created by the Os Generator in the output directory.

By setting `OS_BUILD_OPTIMIZED_LIB_FROM_SOURCE` to `TRUE`, the name of the object-file directory and the kernel library get a library ID encoded into them. This ID identifies uniquely all the optimizations that affect the kernel library, so that if you change your configuration in a way that changes the optimization, a new library is automatically selected and, if necessary, compiled.

5.6.2.1.2. Building an optimized library with a custom build environment

If you are using your own build environment, you only have to check the configuration option **OsSourceOptimization** as described in the previous section to compile the optimized library correctly. You don't need to define any preprocessor macro yourself.

Using a library ID for the library and object files is not compulsory, but if the same name is already used, you must delete the libraries and object files and rebuild the library whenever the configuration changes significantly.

The generated header file `Os_libcfg.h` defines several macros, typically called `OS_EXCLUDE_something`, which tell the kernel that it can omit the code that performs the `something`. The macros are described in the following section.

5.6.2.1.3. Kernel optimization parameters

The following is a list of all the optimization options recognized by the kernel.

> **WARNING** **Do not define these macros manually**
>
> These optimizations are obtained directly from the OS configuration by the Os Generator. The macros are defined automatically to get the most optimizations for your configuration. Do not define these macros manually. Manual definition may result in compile-time, link-time and run-time errors.

`OS_EXCLUDE_APPLICATIONS`
: This macro is defined, if the configuration contains no OS applications. With this macro many application-related functions can be omitted.

`OS_EXCLUDE_CALLINGCONTEXTCHECK`
: This macro removes the explicit calling-context check from kernel API functions.

`OS_EXCLUDE_CAT2ISR`
: This macro excludes category 2 interrupt service routines (ISRs) from your module configuration. Some functions related to ISRs can be omitted.

`OS_EXCLUDE_CPULOAD`
: With this macro, the CPU load monitoring feature is omitted.

`OS_EXCLUDE_ERRORHANDLING`
: This macro omits the complete error handling code. Error codes are returned to the caller. No application specific hook functions are called.

`OS_EXCLUDE_ERRORHOOK`
: This macro excludes the code that calls the error hook from the Os configuration.

`OS_EXCLUDE_ERRORHOOK_APP`
: This macro omits the code that calls the application's error hook. This means that the processor mode switch is omitted, too.

`OS_EXCLUDE_EVENTS`
: This macro omits all functions related to events, such as `WaitEvent`, `SetEvent`, etc. A few other optimizations in `ActivateTask` are also possible.

`OS_EXCLUDE_EXCEPTIONS`
: This macro omits the standard exception handling. Instead of the standard exception handling, a user-provided function is called.

OS_EXCLUDE_EXTENDED
: This macro omits all error checking that takes place in EXTENDED status.

OS_EXCLUDE_EXTRA_CHECK
: This macro omits all code that is related to extra runtime checks.

OS_EXCLUDE_HWCOUNTER
: This macro omits all the functionality for hardware counters: initialization, starting and stopping during alarm processing.

OS_EXCLUDE_HW_FP
: This macro omits floating-point context switching. It only has an effect on architectures which offer hardware floating point support.

OS_EXCLUDE_INTSENABLEDCHECK
: This macro omits checking whether interrupts are enabled. these checks are required by AUTOSAR for almost all API functions.

OS_EXCLUDE_KILLABLE_APPEHOOK
: This macro calls error hooks belonging to applications directly without saving the context. This means that these error hooks run as trusted code and cannot be killed.

OS_EXCLUDE_KILLABLE_APPSHOOK
: This macro calls the startup and shutdown hooks belonging to applications directly without saving context. This means that these hooks run as trusted code and cannot be killed.

OS_EXCLUDE_KILLABLE_ISR
: This macro calls interrupt service routines (ISRs) directly without saving context. This means that the ISRs run as trusted code and cannot be killed.

OS_EXCLUDE_KILLALARM
: This macro omits the function for killing an alarm. This means an application with an active alarm cannot be properly terminated.

OS_EXCLUDE_KILLISR
: This macro omits the function for killing an interrupt service routine (ISR). This means an ISR can never be killed in response to a protection error.

OS_EXCLUDE_KILLTASK
: This macro omits the function for killing a task. This means a task cannot be killed in response to a protection error.

OS_EXCLUDE_MULTIPLE_ACTIVATIONS
: If there are no tasks with multiple activations, this macro omits the code to handle those in ActivateTask and TerminateTask.

OS_EXCLUDE_PARAMETERACCESS
: If the error hooks do not need to determine what API parameters cause the error, the code that passes the parameters through the error handling can be omitted. This affects all API functions.

OS_EXCLUDE_POSTISRHOOK
: This macro omits the code that calls the PostISRHook.

OS_EXCLUDE_POSTTASKHOOK
: This macro omits the code that calls the PostTaskHook.

OS_EXCLUDE_PREISRHOOK
: This macro omits the code that calls the PreISRHook.

OS_EXCLUDE_PREEMPTION
: This macro simplifies the possible context switch at the end of the interrupt handler.

OS_EXCLUDE_PRETASKHOOK
: This macro omits the code in the error handler that calls the PreTaskHook.

OS_EXCLUDE_PROTECTION
: This macro omits all code that is related to memory protection.

OS_EXCLUDE_PROTECTIONHOOK
: The code in the error handler that calls the ProtectionHook is omitted.

OS_EXCLUDE_RATEMONITORS
: This code omits all the arrival rate limiting code in ActivateTask, SetEvent, WaitEvent and in the handling of category 2 ISRs.

OS_EXCLUDE_RESOURCEONISR
: This macro omits the code that adjusts the interrupt levels in GetResource and ReleaseResource.

OS_EXCLUDE_SHUTDOWNHOOK
: This macro omits the code that calls the shutdown hook.

OS_EXCLUDE_STACKCHECK
: This macro omits all software stack checking.

OS_EXCLUDE_STARTUPHOOK
: This macro omits the code that calls the StartupHook in StartOS.

OS_EXCLUDE_SWCOUNTER
: This macro omits all code related to software counters (including the IncrementCounter API).

OS_EXCLUDE_TIMINGPROTECTION
: This macro omits all the code that implements execution-time protection (execution budget, resource and interrupt lock timing).

OS_EXCLUDE_USERTASKRETURN
: This macro omits the code that handles a return from a task's main function. This means that if a task fails to call TerminateTask and simply returns from its main function, the result is undefined but will most likely result in the task entering an endless loop, which will lock out equal and lower priority tasks.

OS_EXCLUDE_AGGREGATELIMIT
: This macro is no longer used. The aggregate execution-time limit was removed in AUTOSAR version 3.0.

5.6.2.2. Enabling kernel customizations

Kernel customizations are further options that have been added by EB. These options must be explicitly enabled in the OS configuration, and can provide further reductions in size and runtime. However, many of them result in a kernel whose behavior is not strictly AUTOSAR-compliant, so these options must be used with care. In particular, extreme caution must be exercised if customized error handling is selected in a system with protection features enabled.

To use the kernel customizations:

- In EB tresos Studio, open your OS configuration in the **Os (OS)** editor.
- Open the **OsOS** tab.
- Enable the `OsAutosarCustomization` container.

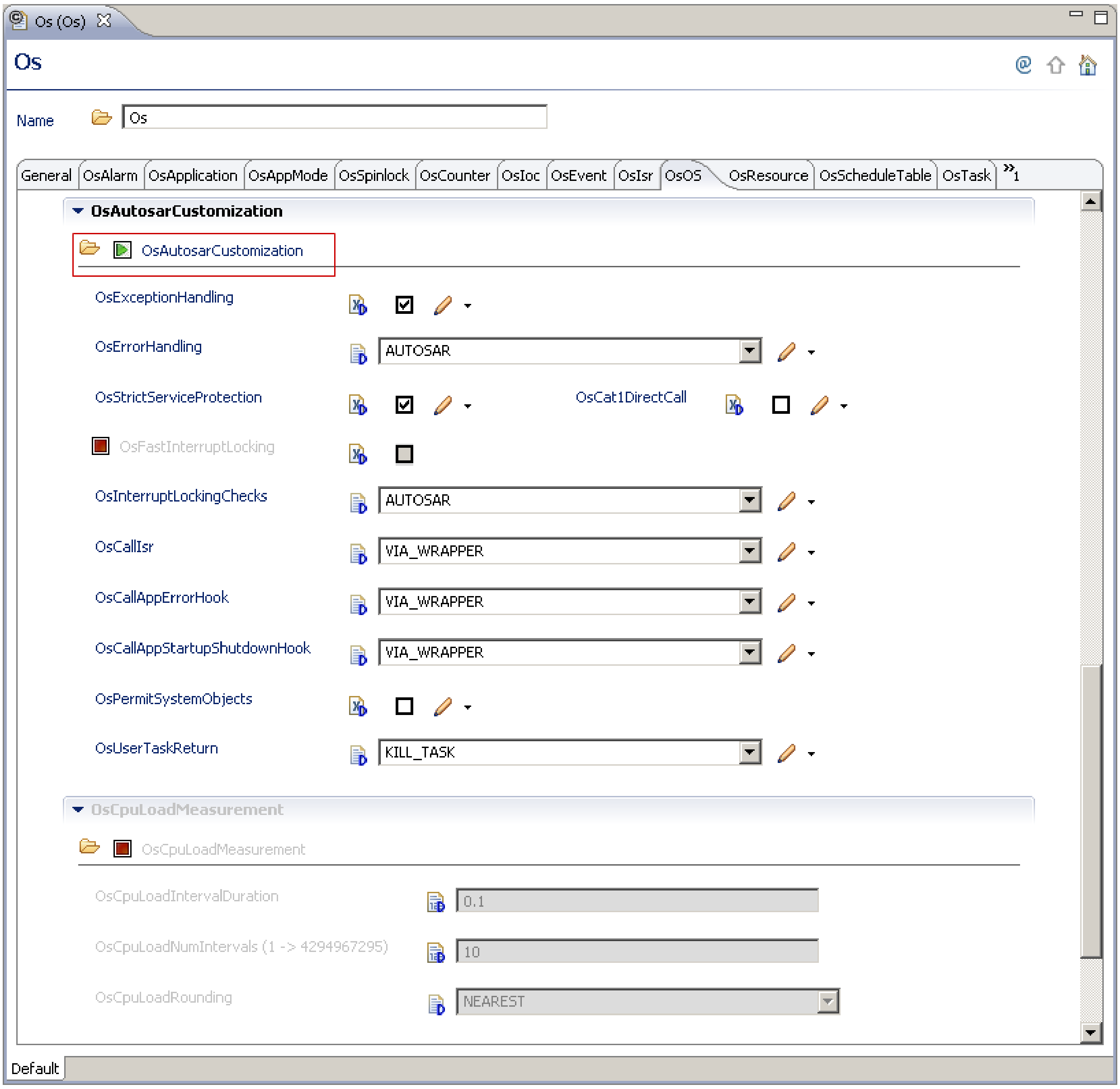


Figure 5.12. Enabling the `OsAutosarCustomization` container

▶ Configure the options inside the `OsAutosarCustomization` container as desired:

Parameter	Description

`OsErrorHandling`	`AUTOSAR`
	Select `AUTOSAR` to choose AUTOSAR-compliant error handling.
	`FULL`
	Select `FULL` to choose error handling in which errors in APIs that do not return `StatusType` are detected and handled. The `ErrorHook` is called and the default error action is performed, which could result in the calling Task,

Parameter	Description
	ISR or hook function being killed. If the action is to return an error code, the API silently fails to do its job. MINIMAL Select MINIMAL to choose error handling in which API functions return an error code if an error is detected. In EB tresos AutoCore OS versions 4.1.-7 and upwards, the ErrorHook() is called if configured, and parameter access is also supported. In older EB tresos AutoCore OS versions the error hook is not called, so internal errors are detected but not reported. With MINIMAL error handling all errors are reported the same way regardless of type, and the ProtectionHook and application-specific ErrorHooks are not supported. This option is not suitable for use with scalability classes SC3 and SC4. **NOTE** MINIMAL error handling will only be effective if OsSourceOptimization is enabled (see also Section 5.6.2.1, "Optimizing the library").
OsStrictServiceProtection	TRUE To set OsStrictServiceProtection to TRUE, select the check box. This is the default behavior. FALSE To set OsStrictServiceProtection to FALSE, clear the check box. If you set OsStrictServiceProtection to FALSE, the very strict calling-checks required by AUTOSAR are disabled. However, the implicit checks that are necessary for correct functioning of the kernel, such as TerminateTask being called from a task, are still present, so this does not affect the safety of the kernel. In the EB tresos AutoCore OS kernel, many APIs work correctly when they are called from a context that is forbidden by AUTOSAR. The functions ActivateTask and SetEvent work correctly when they are called from an alarm callback or from the ErrorHook.
OsInterruptLockingChecks	MINIMAL Select MINIMAL to only check the interrupt lock status when it affects the kernel's operation. The interrupt lock status affects the kernel's operation e.g. in the functions GetResource and ReleaseResource. EXTRACHECK Select EXTRACHECK to check the interrupt lock status in all API functions which may cause a task switch.

Parameter	Description
	`AUTOSAR` Select `AUTOSAR` to be fully compliant with the AUTOSAR specification. **NOTE** **EB tresos AutoCore OS tasks always start with interrupts enabled** In EB tresos AutoCore OS the interrupt lock status is considered to be part of the task's context. This means that each newly activated task starts with an interrupt enabled.

`OsCallIsr`	`DIRECTLY`
	Select `DIRECTLY` to always run ISRs as supervisor with kernel protection boundaries. If you select `DIRECTLY` ISRs cannot be killed: If a protection error occurs in an ISR, the only possible course of action is `SHUTDOWN`.
	
	`VIA_WRAPPER`
	Select `VIA_WRAPPER` to run ISRs inside an OS wrapper. In this case, the ISRs may run in user mode and can be killed in case of an error.

`OsCallAppErrorHook`	`DIRECTLY`
	Select `DIRECTLY` to always run application-specific error hooks as supervisor with kernel protection boundaries. If you select `DIRECTLY`, error hooks cannot be killed: If a protection error occurs in a hook function, the only possible course of action is `SHUTDOWN`.
	
	`VIA_WRAPPER`
	Select `VIA_WRAPPER` to run error hooks inside an OS wrapper. In this case, the hook functions may run in user mode and can be killed in case of an error.

`OsCallAppStartupShutdownHook`	`DIRECTLY`
	Select `DIRECTLY` to always run application-specific startup and shutdown hooks as supervisor with kernel protection boundaries. If you select `DIRECTLY`, application-specific startup and shutdown hooks cannot be killed: If a protection error occurs in a hook function, the only possible course of action is `SHUTDOWN`.
	
	`VIA_WRAPPER`
	Select `VIA_WRAPPER` to run startup and shutdown hooks inside an OS wrapper. In this case, the hook functions may run in user mode and can be killed in case of an error.

`OsPermitSystemObjects`	`TRUE` To set `OsPermitSystemObjects` to `TRUE`, select the check box. If `OsPermitSystemObjects` is set to `TRUE`, and if your system contains OS applications, OS objects are permitted to belong to the system itself and not

Parameter	Description
	to any particular OS application. Such objects can access other objects without restrictions. This feature is useful for objects such as schedule tables that control the scheduling of all applications in a system. `FALSE` To set `OsPermitSystemObjects` to `FALSE`, clear the check box. In this case, each OS object must belong to an OS application if OS applications are used.

`OsUserTaskReturn`	This option determines what happens when a task returns from its main function.
	`KILL_TASK`
	Select `KILL_TASK` to end the task after returning from its main function. `KILL_TASK` is AUTOSAR-compliant but requires the full error handling and error action to be enabled for correct functionality.
	`LOOP`
	Select `LOOP` to make a task that returns from its main function try to terminate. If termination fails, the task tries to shut down the OS. If shutting down the OS fails, the task enters an endless loop with the effect of locking out all tasks of equal or lower priority.
	NOTE **This option has no effect when return-from-task is caught by a special exception handler**
	On architectures such as Tricore on which return-from-task is caught using a special exception handler, this option has no effect.

5.6.3. Compiling EB tresos AutoCore OS in custom build environments

This section provides instructions as to how EB tresos AutoCore OS can be compiled outside the user build environment provided by EB. You may derive all the information necessary from the Makefiles provided by the demo application and the EB tresos Studio plugins it uses.

WARNING **Generation of non-executable or non-compilable code**

If you use another build environment than the one delivered with EB tresos AutoCore OS, your EB tresos AutoCore OS version is considered untested. This might lead to non-executable or non-compilable code.

To avoid non-executable or non-compilable code, do not use another build environment than the build environment integrated into EB tresos AutoCore OS.

5.6.3.1. Determining the source files and include paths

The list of source files that is necessary to build the EB tresos AutoCore OS is located in the OS plugin makefiles; these are files ending with `.mak` which are located in the `$TRESOS_BASE\plugins\Os_TS_T<a>D<b>M4I4R0_<release suffix>\make`[1] directory.

To find out the needed files, do the following:

- Provide an environment similar to the one in the demo application. Set the variables `SSC_ROOT`, `PROJECT_ROOT`, `PROJECT_OUTPUT_PATH` and `TOOLCHAIN` according to the makefiles in the demo application.
- Include the files `Os_defs.mak` and `Os_rules.mak` in that order.
- The makefiles define a set of variables which specify the libraries needed to build the OS and their needed source files:

  `LIBRARIES_TO_BUILD`
  : the names of the libraries needed to build the OS

  `<libname>_FILES`
  : for each library in `LIBRARIES_TO_BUILD`, a list of source files to build for that library

  `CC_INCLUDE_PATH`
  : all needed include directories to build C files

  `ASM_INCLUDE_PATH`
  : all needed include directories to build assembler files

- Use the variables set by the makefiles to determine the files to build. For example, you could use a makefile snippet like the following:

```
# list all needed source files in SOURCE_FILES
SOURCE_FILES := $(foreach lib,$(LIBRARIES_TO_BUILD),$($(lib)_FILES))
```

- Use the variables `CC_INLCUDE_PATH` and `ASM_INCLUDE_PATH` to determine all directories containing header files. Add these directories to your include path.

[1]The actual name of your installation directory depends on your OS variant, e.g. the target CPU and the AUTOSAR release. It may look like the following: `$TRESOS_BASE\plugins\Os_TS_T17D1M4I4R0_AS40\make`

NOTE

Assembler files must be preprocessed

The assembler files provided by EB tresos AutoCore OS include C preprocessor macros. If your assembler does not include a C preprocessor, feed the assembler files to the C preprocessor before running the assembler.

5.6.3.2. Determining the compiler options

5.6.3.2.1. Options influencing the build process

The compiler options used to build the module are located in the EB tresos AutoCore OS release notes.

In general, only the set of options described there has been validated to work correctly.

5.6.3.2.2. Options for defining preprocessor macros

The EB tresos AutoCore OS relies on some configuration-dependent preprocessor definitions during compilation.

To find out the set of preprocessor definitions needed for your configuration, do the following:

- Provide an environment similar to the one in the demo application. Set the variables `SSC_ROOT`, `PROJECT_ROOT`, `PROJECT_OUTPUT_PATH` and `TOOLCHAIN` according to the makefiles in the demo application.
- Include the files `Os_defs.mak` and `Os_rules.mak` in that order.
- The makefiles define a set of variables which specify the libraries needed to build the OS and their needed source files:

  `PREPROCESSOR_DEFINES`
  : a list of identifiers to distinguish each define in the Makefiles

  `<define>_KEY`
  : for each define in `PREPROCESSOR_DEFINES`, the key to use for the C preprocessor

  `<define>_VALUE`
  : for each define in `PREPROCESSOR_DEFINES`, the value to use for the C preprocessor

- Use the variables set by the makefiles to determine the compiler command line needed to set the corresponding defines. For example, if your compiler uses `-D<key>=<value>` to set the define `<key>` to `<value>`, you could use a makefile snippet like the following:

```
# get compiler command line snippet for preprocessor defines
```

```
PREPROC_OPTS := $(foreach d,$(PREPROCESSOR_DEFINES), -D$($(d)_KEY)=$($(d)_VALUE))
```

6. Multicore User's guide

6.1. Overview

This users guide contains information on how to use EB tresos AutoCore OS on a multicore microcontroller. You will find

- background information about the AUTOSAR multicore philosophy when using this OS
- information on configuring a multicore OS
- information on running the target software

All examples in this document refer - if not explicitly mentioned - to the PowerPC Leopard from Freescale/STM. This microcontroller has a symmetrical multicore architecture consisting of two e200z4 cores.

6.2. Background information

6.2.1. Overview

This chapter contains general information about the AUTOSAR philosophy for the support of multicore microcontrollers. Since EB tresos OS is based on AUTOSAR it is recommended to also read the related AUTOSAR specifications ("Specification of Multi-Core OS Architecture", V1.0.0; "Specification of Operating System", V4.-0.0) since they contain more details.

6.2.1.1. Static system

An AUTOSAR OS is a real-time operating system (RTOS) which maintains a collection of system objects. All these objects are predefined and an object is never created during runtime dynamically. This means that the user has to configure (or define) all these objects before. Objects can be Tasks, Alarms, Schedule Tables, and so forth. The same applies to the multicore variants of AUTOSAR OS. Here it is important to mention that also the mapping of objects to cores is fixed and does not change during runtime. The mapping is done via so called OS-Applications. Each OS object belongs to an OS-Application and each OS-Application is bound to one core. From the runtime perspective this means that the user has full control on which core a Task (or interrupt) is executed. Also maths can be used to show that a system is able (or not able) to manage its work in time, e.g. via RMA. The drawback of this static mapping is that it might happen that on one core no Task is ready to execute but on other cores more than one Task may be executable. A migration of Tasks (in this case to a idle core) is not possible.

6.2.1.2. Grouping and mapping of system objects

AUTOSAR OS offers OS-Applications which are a collection of different OS objects. Normally OS-Applications are only available in specific scalability classes (SC). For a single core OS they are only available in scalability classes 3 and 4 (SC3/SC4). In a multicore system also the scalability classes SC1 and SC2 allow the usage of OS-Applications since OS-Applications are used to map OS objects to cores. As a consequence all OS objects must belong to an OS-Application. The only exception are spinlocks which are used to synchronize access to shared data.

6.2.1.3. Communication between cores

The multicore OS offers different methods for communication between cores:

- Use ActivateTask() / SetEvent() to either activate a task on another core or set an event of an extended task.
- Use own data area and synchronize access via spinlocks.
- Use the inter OS-Application communication (IOC).

6.2.1.3.1. Usage of ActivateTask / SetEvent

It is possible to use a task activation (e.g. via ActivateTask()) or the setting of an event to start activities on a different core. The caller's OS-Application must have access rights to the Task which is either activated or receives the event. Note that this may cause a rescheduling on the core where the receiving / activated Task is placed. This mechanism is quite useful e.g. to start an action on a different core.

6.2.1.3.2. Usage of own data structures

It is possible to exchange date / information via shared memory if it can be accessed by the communicating cores. At least one core needs write access to the area, for the other cores it might be enough to have read access. Specific care should be taken regarding hardware effects, like cache coherency problems or the visibility of the shared memory on different addresses. The method is quite effective to send / receive data blocks. Since all the synchronization primitives must be managed by the application - e.g. by proper use of GetSpinlock() and ReleaseSpinlock() it can be a quite fast method.

6.2.1.3.3. Usage of IOC

The IOC offers communication primitives between OS-Applications. These mechanism can be used to send and receive signals between different OS-Application running of different cores. In AUTOSAR the IOC is used by the RTE. Also complex device drivers may use the IOC to communicate.

The functionality of the IOC can be described best as a message box or black board service. Message boxes can be used to send single messages to a receiver. if the receiver has read (consumed) the message it will be removed from the message box. The user can specify how many messages can be send to a message box. Black boards in comparison contain always a valid message value which can be read by the receivers (and which is not consumed but replaced if a newer message is send). Normally the sending (or receiving) of messages has no impact on other cores. This means e.g. that there is no sort of automatic notification on the receiver side if a message is send. These notifications are normally done by the application, e.g. by using ActivateTask() or SetEvent().

The advantage of using the IOC instead of managing own data areas as mentioned before is that it takes care about hardware and other constraints. This means e.g. that it is able to handle cache coherency or to cross memory protection boundaries.

6.2.1.4. Startup

The startup of an AUTOSAR multicore system can be splitted into two phases: before and after the call to StartOS().

- During the first phase the startup code does some low-level initialization and afterwards calls the compiler startup to initialize the data sections. Afterwards the other cores may be started and on each started core StartOS() is called.
- In the second phase - after StartOS() was called on each used core - the OS performs further internal initializations and autostarts possible alarms and schedule tables and activates task depending on the application mode. The application mode is given to StartOS() as a parameter. Afterwards the cores are synchronized and call the global StartupHook() and the application specific StartupHooks if configured. When all these hooks have finished the OS again synchronizes the cores then the scheduler starts scheduling tasks on each core.

Since the real steps to initialize the hardware, registers, etc. are very MCU depnedend, the following list gives an overview about this procedure on the Leopard. To improve the understanding of what is executed on the different cores the perfixes MC (master core) and SC (slave core) are used.

1. [MC] After a reset one core (the "master") starts executing code. Since the Leopard has by default only limited access to memory (only one 4 kB page via MMU) the MMU is configured to allow access to other memory areas (RAM, Flash, IO). Afterwards PLL or other clocks are configured.
2. [MC] The compiler startup code initializes data (C variables) and clears the bss area. Note that if a SC3 or SC4 system with memory protection is used the initialization takes typically only place for the real global data / bss. Application specific data is initialized during StartOS(). Since the Leopard uses a shared RAM for all cores the global initialization does only happen on the master core.
3. [MC] The main() function is called. Here the second core is started via StartCore(). Afterwards the master core calls StartOS(). Within StartOS() the OS waits for the arrival (call of StartOS()) of the second core.
4. [SC] The second core starts and does also configure its MMU to have access to all needed memory areas. Afterwards it calls main()
5. [SC] In the main() function the core calls StartOS().
6. [MC] and [SC] After both cores have called StartOS() the OS will internally call the StartupHook() (if configured) on both cores. Within the StartupHook() the code can check the core where it is executed by calling GetCoreID(). Afterwards the application specific startup hooks are called. An application specific startup hook is only called on the core to which the application is bound.
7. [MC] and [SC] After the calls to the application startup hooks have returned the cores synchronize each other and then each core starts scheduling.

6.2.1.5. Shutdown

The shutdown of an AUTOSAR multicore OS can be realized either global by shuting down all cores or local by shuting down only one core.

6.2.1.5.1. Shutdown of a single core

The shutdown of an single core is done by calling ShutdownOS(). Afterwards all OS-Applications of the core where the service was called are terminated and the application specific shutdown hooks of those OS-Applications are called. After this the core will wait for the other cores to shut down. If this happens the global ShutdownHook() is called synchronously on all cores. If this call returns, the OS enters an endless loop (typically a watchdog reset will occure).

6.2.1.5.2. Shutdown of all cores

The shutdown of all cores is done by calling ShutdownAllCores(). This service will signal the shutdown to all cores. On each of the cores all OS-Applications are terminated and the application specific shutdown hooks of those OS-Applications are called. At the end the cores synchronize and ShutdownHook() is called on each core. If this call returns, the OS enters an endless loop (on this core) and typically a watchdog will reset the MCU.

6.3. Configuring multicore systems

6.3.1. MultiCore configuration

This version of the OS does only support the standardized AUTOSAR configuration parameters as described in "Specification of Multi-Core OS Architecture" from AUTOSAR R4.0rev1 chapter 10.

6.4. API

6.4.1. MultiCore API

This version of the OS does only support the standardized AUTOSAR API as described in "Specification of Multi-Core OS Architecture" from AUTOSAR R4.0rev1.

6.4.1.1. GetCoreID()

NAME	**GetCoreID**
SYNOPSIS	Get the ID of the callers core
SYNTAX	`CoreIdType GetCoreID( void )`
DESCRIPTION	This service returns the unique number identifier of the core where the caller is executing.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.

NAME	**GetCoreID**
RETURNS	Core number of the core where the service was called
CONFORMANCE	Autosar MultiCore API

6.4.1.2. GetNumberOfActivatedCores()

NAME	**GetNumberOfActivatedCores**
SYNOPSIS	Get the number of cores activated via StartCore()
SYNTAX	`uint32 GetNumberOfActivatedCores( void )`
DESCRIPTION	This service returns the number of cores activated via StartCore.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	Number of cores where StartCore() was called
CONFORMANCE	Autosar MultiCore API

6.4.1.3. GetSpinlock()

NAME	**GetSpinlock**
SYNOPSIS	Get a spinlock
SYNTAX	`StatusType GetSpinlock( SpinlockIdType SpinlockId )`
DESCRIPTION	GetSpinlock tries to occupy a spin-lock variable. The service will actively loop until the spin-lock variable can be taken.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
E_OK	Success
E_OS_ID	Parameter is not a valid spin-lock

NAME	**GetSpinlock**
E_OS_STATE	Spin-lock is already locked by the caller
E_OS_INTERFER-ENCE_DEADLOCK	Spin-lock is already locked by a Task on the same core as the caller
E_OS_NESTING_-DEADLOCK	Locking of spin-lock may cause a deadlock
E_OS_ACCESS	Spin-lock can not be accessed
CONFORMANCE	Autosar MultiCore API

6.4.1.4. TryToGetSpinlock()

NAME	**TryToGetSpinlock**
SYNOPSIS	Try to get a spinlock
SYNTAX	`StatusType TryToGetSpinlock( SpinlockIdType SpinlockId, TryToGetSpinlockType* Success )`
DESCRIPTION	TryToGetSpinlock tries to occupy a spin-lock variable. The success (TRY-TOGETSPINLOCK_SUCCESS or TRYTOGETSPINLOCK_NOSUCCESS) will be reported via the out parameter.
AVAILABILITY	See Table 6.1, "Permitted calling context for Multicore API".
RETURNS	
E_OK	Success
E_OS_ID	Parameter is not a valid spin-lock
E_OS_STATE	Spin-lock is already locked by the caller
E_OS_INTERFER-ENCE_DEADLOCK	Spin-lock is already locked by a Task on the same core as the caller
E_OS_NESTING_-DEADLOCK	Locking of spin-lock may cause a deadlock
E_OS_ACCESS	Spin-lock can not be accessed

NAME	TryToGetSpinlock
CONFORMANCE	Autosar MultiCore API

6.4.1.5. ReleaseSpinlock()

NAME	ReleaseSpinlock
SYNOPSIS	Get a spinlock
SYNTAX	`StatusType ReleaseSpinlock( SpinlockIdType SpinlockId )`
DESCRIPTION	ReleaseSpinlock releases a spin-lock variable that was occupied before. Before terminating a task all spinlock variables that have been occupied with GetSpinlock() or TryToGetSpinlock() shall be released. The same applies to a call to WaitEvent().
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
E_OK	Success
E_OS_ID	Parameter is not a valid spin-lock
E_OS_STATE	Spin-lock is already not locked by the caller
E_OS_NOFUNC	Before the given spin-lock can be released, another occupied spin-lock has to be released
E_OS_ACCESS	Spin-lock can not be accessed
CONFORMANCE	Autosar MultiCore API

6.4.1.6. ShutdownAllCores()

NAME	ShutdownAllCores
SYNOPSIS	Shutdown all cores in a multicore system
SYNTAX	`void ShutdownAllCores( StatusType Error )`

NAME	ShutdownAllCores
DESCRIPTION	This serives init a shutdown of all cores in a multicore system. The function will never return.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	-
CONFORMANCE	Autosar MultiCore API

6.4.1.7. StartCore()

NAME	StartCore
SYNOPSIS	Start the given core
SYNTAX	`void StartCore( CoreIdType CoreID, StatusType* Status )`
DESCRIPTION	This service starts the given core and returns status information via the Status parameter. Valid calls can only be executed before StartOS() on the callers core is called. E_OK is returned if core was succesfully started, E_OS_ID indicates a wrong core ID, E_OS_STATE indicates that the core was already activated. E_OS_ACCESS is returned if the service is called after StartOS().
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	-
CONFORMANCE	Autosar MultiCore API

6.4.1.8. StartNonAutosarCore()

NAME	StartNonAutosarCore
SYNOPSIS	Start core which is not used by autosar
SYNTAX	`void StartNonAutosarCore( CoreIdType CoreID, StatusType* Status )`

NAME	**StartNonAutosarCore**
DESCRIPTION	This service starts the given core and returns status information via the Status parameter. E_OK is returned if core was succesfully started, E_OS_ID indicates a wrong core ID, E_OS_STATE indicates that the core was already activated.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	-
CONFORMANCE	Autosar MultiCore API

6.4.1.9. IocEmptyQueue_%IocId%()

NAME	**IocEmptyQueue_%IocId%**
SYNOPSIS	Empty a queued communciation
SYNTAX	`Std_ReturnType IocEmptyQueue_%IocId%( void )`
DESCRIPTION	In case of queued communication identified by the %IocId% in the function name, the content of the IOC internal communication queue shall be deleted
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Content of the queue was successfully deleted
CONFORMANCE	Autosar MultiCore API

6.4.1.10. IocReadGroup_%IocId%()

NAME	**IocReadGroup_%IocId%**
SYNOPSIS	Read multiple signals (data semantic)

NAME	**IocReadGroup_%IocId%**
SYNTAX	`Std_ReturnType IocReadGroup_%IocId%( %Data1% OUT1, %Data2% OUT2, ... )`
DESCRIPTION	Performs an "explicit" sender-receiver reception of data elements with a "data" semantic for a unidirectional 1:1 communication between OS-Applications located on the same or on different cores. This API involves a group of data elements which values are specified in parameter. %IocId% is a unique identifier that references a unidirectional 1:1 communication involving many data elements.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Success
CONFORMANCE	Autosar MultiCore API

6.4.1.11. IocRead_%IocId%()

NAME	**IocRead_%IocId%**
SYNOPSIS	Read a signal (data semantic)
SYNTAX	`Std_ReturnType IocRead_%IocId%( %Data1% OUT )`
DESCRIPTION	Performs an "explicit" sender-receiver reception of data elements with "data" semantic for a unidirectional communication between OS-Applications located on the same or on different cores. %IocId% is a unique identifier that references a unidirectional 1:1 or N:1 communication.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Success

NAME	**IocRead_%IocId%**
CONFORMANCE	Autosar MultiCore API

6.4.1.12. IocReceiveGroup_%IocId%()

NAME	**IocReceiveGroup_%IocId%**
SYNOPSIS	Read multiple signals (event semantic)
SYNTAX	`Std_ReturnType IocReceiveGroup_%IocId%( %Data1% OUT1, %Data2% OUT2, ... )`
DESCRIPTION	Performs an "explicit" sender-receiver reception of data elements with "event" semantic for a unidirectional 1:1 communication between OS-Applications located on the same or on different cores. This API involves a group of data elements which values are specified in parameter. %IocId% is a unique identifier that references a unidirectional 1:1 communication involving many data elements.
AVAILABILITY	See Table 6.1, "Permitted calling context for Multicore API".
RETURNS	
IOC_E_OK	Success
IOC_E_NO_DATA	No data available for receiption
IOC_E_OK	Data is received but some data were lost (buffer overflow on sender side)
CONFORMANCE	Autosar MultiCore API

6.4.1.13. IocReceive_%IocId%()

NAME	**IocReceive_%IocId%**
SYNOPSIS	Receive a signal (event semantic)
SYNTAX	`Std_ReturnType IocReceive_%IocId%( %Data1% OUT )`

NAME	IocReceive_%IocId%
DESCRIPTION	Performs an "explicit" sender-receiver reception of data elements with "event" semantic for a unidirectional communication between OS-Applications located on the same or on different cores.. %IocId% is a unique identifier that references a unidirectional 1:1 or N:1 communication.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Success
IOC_E_NO_DATA	No data available for receiption
IOC_E_OK	Data is received but some data were lost (buffer overflow on sender side)
CONFORMANCE	Autosar MultiCore API

6.4.1.14. IocSendGroup_%IocId%()

NAME	IocSendGroup_%IocId%
SYNOPSIS	Send a group of signals to another application
SYNTAX	`Std_ReturnType IocSendGroup_%IocId%( %Data1% IN1, %Data2% IN2, ... )`
DESCRIPTION	Performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 communication between OS-Applications located on the same or on different cores. This API involves a group of data elements which values are specified in parameter. IocId is a unique identifier that references a unidirectional 1:1 communication involving many data elements.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Success
IOC_E_LIMIT	buffer is full

NAME	**IocSendGroup_%IocId%**
CONFORMANCE	Autosar MultiCore API

6.4.1.15. IocSend_%IocId%[_%SenderId%]()

NAME	**IocSend_%IocId%[_%SenderId%]**
SYNOPSIS	Send a signal to another application
SYNTAX	`Std_ReturnType IocSend_%IocId%[_%SenderId%]( %Data% IN )`
DESCRIPTION	Performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores. IocId is a unique identifier that references a unidirectional 1:1 or N:1 communication. SenderId is used only in N:1 communication. Together with IocId, it uniquely identifies the sender. It is separated from IocId with an underscore. In case of 1:1 communication, it shall be omitted.
AVAILABILITY	See Table 6.1, “Permitted calling context for Multicore API”.
RETURNS	
IOC_E_OK	Success
IOC_E_LIMIT	buffer is full
CONFORMANCE	Autosar MultiCore API

6.4.1.16. IocWriteGroup_%IocId%()

NAME	**IocWriteGroup_%IocId%**
SYNOPSIS	Send a group of signals to another application
SYNTAX	`Std_ReturnType IocWriteGroup_%IocId%( %Data1% IN1, %Data2% IN2, ... )`

NAME	**IocWriteGroup_%IocId%**
DESCRIPTION	Performs an "explicit" sender-receiver transmission of data elements with "data" semantic for a unidirectional 1:1 communication between OS-Applications located on the same or on different cores. This API involves a group of data elements which values are specified in parameter. %IocId% is a unique identifier that references a unidirectional 1:1 communication involving many data elements.
AVAILABILITY	See Table 6.1, "Permitted calling context for Multicore API".
RETURNS	
IOC_E_OK	Success
CONFORMANCE	Autosar MultiCore API

6.4.1.17. IocWrite_%IocId%[_%SenderId%]()

NAME	**IocWrite_%IocId%[_%SenderId%]**
SYNOPSIS	Send a signal to another application
SYNTAX	`Std_ReturnType IocWrite_%IocId%[_%SenderId%]( %Data% IN )`
DESCRIPTION	Performs an "explicit" sender-receiver transmission of data elements with "data" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores. IocId is a unique identifier that references a unidirectional 1:1 or N:1 communication. SenderId is used only in N:1 communication. Together with IocId, it uniquely identifies the sender. It is separated from IocId with an underscore. In case of 1:1 communication, it shall be omitted.
AVAILABILITY	See Table 6.1, "Permitted calling context for Multicore API".
RETURNS	
IOC_E_OK	Success

NAME	**IocWrite_%IocId%[_%SenderId%]**
CONFORMANCE	Autosar MultiCore API

6.4.2. Permitted calling context

This section contains the permitted calling context for the Multicore API functions.

Service	Task	Cat1ISR	Cat2ISR	ErrorHook	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	AlarmCallback	ProtectionHook
GetNumberOfActivatedCores	Y		Y							
GetCoreID	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
StartCore										
StartNonAutosarCore										
GetSpinlock	Y		Y							
ReleaseSpinlock	Y		Y							
TryToGetSpinlock	Y		Y							
ShutdownAllCores	Y		Y	Y			Y			

Table 6.1. Permitted calling context for Multicore API

7. References

7.1. EB tresos AutoCore OS Configuration Language

The EB tresos AutoCore OS Generator supports OSEK Implementation Language (OIL) version 3.0, AUTOSAR ECU Parameter Configuration (EPC) and the XML Data Model (XDM).

OIL's grammar and syntax are beyond the scope of this manual. See the appropriate OSEK documentation for full details.

This chapter describes the configuration of standard objects and attributes and the architecture-independent extensions implemented by EB tresos AutoCore OS.

7.1.1. Common configuration parameters

Containers included		
Container name	**Multiplicity**	**Description**
OsAlarm	0..n	An **OsAlarm** may be used to asynchronously inform or activate a specific task. It is possible to start alarms automatically at system start-up depending on the application mode.
OsAppMode	1..255	**OsAppMode** objects are used to define which tasks, alarms, etc. will be started automatically when the kernel is first started. In a valid AUTOSAR OS configuration the **CPU** must contain at least one **OsAppMode** object. Plain OIL defines no attributes for the APPMODE object. An OsAppMode called OSDEFAULTAPPMODE must always be present for OSEK compatibility. [source: OSEK OIL Spec. 2.5]
OsApplication	0..n	An **AUTOSAR OS** must be capable of supporting a collection of OS objects (tasks, interrupts, alarms, hooks etc.) that form a cohesive functional unit. This collection of objects is termed an OS-Application. All objects which belong to the same OS-Application have access to each other. Access means to allow to use these objects within API services. Access by other applications can be granted separately.

Containers included		
OsCounter	0..n	A COUNTER is the mechanism by which Alarms are triggered.
OsEvent	0..n	**OsEvent** objects are used to provide inter-task coordination. Events are represented by their event masks.
OsSpinlock	0..n	An OsSpinlock object is used to co-ordinate concurrent access by TASKs/ISR2s on different cores to a shared resource.
OsIoc	0..1	Configuration of the IOC (Inter OS Application) module.
OsIsr	0..n	**OsIsr** objects are used to represent interrupt service routines. All ISRs should be declared in the application code using the *ISR()* API. The attributes of the ISR object are defined in the following section
OsResource	0..n	An **OsResource** object is used to co-ordinate the concurrent access by tasks and ISRs to a shared resource, e.g. the scheduler, any program sequence, memory or any hardware area.
OsOS	1..1	The OS object defines the existence of, and configuration for, the AUTOSAR OS kernel. In a valid AUTOSAR OS configuration the **CPU** must contain exactly one OS object.
OsScheduleTable	0..n	An OsScheduleTable addresses the synchronization issue by providing an encapsulation of a statically defined set of alarms that cannot be modified at runtime.
OsTask	0..n	**TASK** objects are used to define which tasks are present in the system. The attributes of the **TASK** object are defined in the following sections.

Parameters included	
Parameter name	**Multiplicity**
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Config Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile
Range	VariantPreCompile

7.1.1.1. OsAlarm

Containers included		
Container name	**Multiplicity**	**Description**
OsAlarmAction	1..1	The **OsAlarmAction** attribute is a parametrized enum value specifying what shall happen when the alarm expires. The values are: ▶ ACTIVATETASK ▶ SETEVENT ▶ ALARMCALLBACK ▶ INCREMENTCOUNTER The parameters are: ▶ TASK: The task that shall be activated or have an event set ▶ EVENT: The event that shall be set for the task ▶ ALARMCALLBACKNAME:the name of the alarm callback function to be called. The function should be declared using the *ALARMCALLBACK(name)* API.
OsAlarmAutostart	0..1	**OsAlarmAutostart** is a boolean attribute whose value specifies whether the alarm shall be started automatically when the kernel starts. If the value is **TRUE**, the **OsAlarmAppModeRef** sub-attribute specifies in which application modes the task shall be automatically started, and the sub-attributes **OsAlarmAlarmTime** and **OsAlarmCycleTime** specify the first and subsequent relative values of the counter at which the alarm shall expire.

Parameters included	
Parameter name	**Multiplicity**
OsAlarmAccessingApplication	0..n
OsAlarmCounterRef	1..1

Parameter Name	OsAlarmAccessingApplication
Description	Reference to applications which have an access to this object.
Multiplicity	0..n

Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAlarmCounterRef**
Description	The **OsAlarmCounterRef** attribute specifies the Counter with which the Alarm is associated. Each Alarm must be associated with exactly one Counter.
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.2. OsAlarmAction

Containers included		
Container name	**Multiplicity**	**Description**
OsAlarmActivateTask	1..1	This container specifies the parameters to activate a task.
OsAlarmCallback	1..1	This container specifies the parameters to call a callback OS alarm action.
OsAlarmIncrement-Counter	1..1	This container specifies the parameters to increment a counter.
OsAlarmSetEvent	1..1	This container specifies the parameters to set an event

7.1.1.3. OsAlarmActivateTask

Parameters included	
Parameter name	**Multiplicity**
OsAlarmActivate-TaskRef	1..1

Parameter Name	**OsAlarmActivateTaskRef**
Description	Reference to the task that will be activated by that alarm action
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.4. OsAlarmCallback

Parameters included	
Parameter name	**Multiplicity**
OsAlarmCallbackName	1..1

Parameter Name	OsAlarmCallbackName
Description	Name of the function that is called when this alarm callback is triggered.
Multiplicity	1..1
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC

7.1.1.5. OsAlarmIncrementCounter

Parameters included	
Parameter name	**Multiplicity**
OsAlarmIncrement-CounterRef	1..1

Parameter Name	OsAlarmIncrementCounterRef
Description	Reference to the counter that will be incremented by that alarm action
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.6. OsAlarmSetEvent

Parameters included	
Parameter name	**Multiplicity**
OsAlarmSetEventRef	1..1
OsAlarmSetEvent-TaskRef	1..1

Parameter Name	OsAlarmSetEventRef
Description	Reference to the event that will be set by that alarm action

Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAlarmSetEventTaskRef**
Description	Reference to the task that will be activated by that event
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.7. OsAlarmAutostart

Parameters included	
Parameter name	**Multiplicity**
OsAlarmAlarmTime	1..1
OsAlarmAutostartType	1..1
OsAlarmCycleTime	1..1
OsAlarmAppModeRef	1..n
OsTimeUnit	0..1

Parameter Name	**OsAlarmAlarmTime**
Description	The relative or absolute tick value when the alarm expires for the first time. Note that for an alarm which is RELATIVE the value must be at bigger than 0.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295 >=1
Origin	AUTOSAR_ECUC

Parameter Name	**OsAlarmAutostartType**
Description	This specifies the type of autostart for the alarm..
Multiplicity	1..1
Type	ENUMERATION

Default value	RELATIVE
Range	ABSOLUTE RELATIVE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAlarmCycleTime**
Description	Cycle time of a cyclic alarm in ticks. If the value is 0 than the alarm is not cyclic.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295 >=0
Origin	AUTOSAR_ECUC

Parameter Name	**OsAlarmAppModeRef**
Description	Reference to the application modes for which the AUTOSTART shall be performed
Multiplicity	1..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsTimeUnit**
Description	**OsTimeUnit** contains the time unit type used for this alarm.
Multiplicity	0..1
Type	ENUMERATION
Default value	TICKS
Range	NANOSECONDS TICKS
Origin	Elektrobit Automotive GmbH

7.1.1.8. OsAppMode

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsAlarmRef	0..n
OsScheduleTableRef	0..n
OsTaskRef	0..n

Parameter Name	OsAlarmRef
Description	Optional references to autostarted OS alarms. This configuration parameter is not supported by AutoCore OS, use the application mode reference in the alarm object instead.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsScheduleTableRef
Description	Optional references to autostarted OS schedule tables. This configuration parameter is not supported by AutoCore OS, use the application mode reference in the schedule table object instead.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskRef
Description	Optional references to autostarted OS tasks. This configuration parameter is not supported by AutoCore OS, use the application mode reference in the task object instead.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.9. OsApplication

Containers included		
Container name	**Multiplicity**	**Description**
OsApplicationHooks	1..1	This container structures the OS-Application-specific hooks.

Containers included		
OsApplicationTrusted-Function	0..n	The **OsApplicationTrustedFunction** attribute is a list of **BOOLEAN** attributes specifying trusted functions belonging to this application. If the value is **TRUE**, further sub-attributes specify the **NAME** of the trusted function. There are further implementation-specific sub-attributes. Trusted functions can be called by other applications using the *CallTrustedFunction* API.

Parameters included	
Parameter name	**Multiplicity**
OsTrusted	1..1
OsAppAlarmRef	0..n
OsAppCounterRef	0..n
OsAppIsrRef	0..n
OsAppResourceRef	0..n
OsAppSched-uleTableRef	0..n
OsAppTaskRef	0..n
OsRestartTask	0..1
OsApplicationCoreAs-signment	0..1
OsAppEcucPartitionRef	0..1

Parameter Name	OsTrusted
Description	**OsTrusted** is a boolean attribute that specifies whether Tasks, ISRs etc. associated with the application are to run with the kernel's Privileged or Non-Privileged protection parameters. Privileged applications have access to more of the CPU's resources than non-privileged applications. When **OsTrusted** is **TRUE**, the **TRUSTED_FUNCTION** sub-attributes are available.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	OsAppAlarmRef
Description	Specifies the OsAlarms that belong to the OsApplication.

Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppCounterRef**
Description	References the OsCounters that belong to the OsApplication.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppIsrRef**
Description	references which OsIsrs belong to the OsApplication
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppResourceRef**
Description	References the OsResources that belong to the OsApplication.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppScheduleTableRef**
Description	References the OsScheduleTables that belong to the OsApplication.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppTaskRef**
Description	references which OsTasks belong to the OsApplication
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsRestartTask**

Description	If it is set **OsRestartTask** specifies which task shall be automatically activated when the application is killed and restarted after a serious error.
Multiplicity	0..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsApplicationCoreAssignment**
Description	ID of the core onto which the OsApplication is bound.
Multiplicity	0..1
Type	INTEGER
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppEcucPartitionRef**
Description	Denotes which "EcucPartition" is implemented by this "OSApplication". **This reference is not used by the OS generator.**
Multiplicity	0..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.10. OsApplicationHooks

Parameters included	
Parameter name	**Multiplicity**
OsAppErrorHook	1..1
OsAppShutdownHook	1..1
OsAppStartupHook	1..1
OsAppErrorHookStack	0..1
OsAppShutdownHookStack	0..1
OsAppStartupHookStack	0..1

Parameter Name	**OsAppErrorHook**
Description	**OsAppErrorHook** is a boolean attribute that specifies whether this application has a private error-hook function. If the value is **TRUE**, the kernel calls the user-

	supplied *void ErrorHook_<application-name>(StatusType errorcode)* instead of the global error hook whenever an error is detected in the application, unless the error was caused within an error hook.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppShutdownHook**
Description	**OsAppShutdownHook** is a boolean attribute that specifies whether this application has a private shutdown-hook function. If the value is **TRUE**, the kernel calls the user-supplied *void ShutdownHook_<applicationname>(StatusType errorcode)* when the system has been shut down, before calling the global shutdown hook. The parameter is the value of the error code passed to *ShutdownOS()* Application-specific startup hooks must always return because the order of calling is not defined. Any final action such as restarting the system should take place in the global shutdown hook.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppStartupHook**
Description	The **OsAppStartupHook** attribute specifies whether the application has a private startup-hook function. If the value is **TRUE**, the kernel calls the user-supplied *void StartupHook_<application-name>(void)* immediately before starting the scheduler, after calling the global startup hook.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsAppErrorHookStack**
Description	**OsAppErrorHookStack** defines the stack size of the error hook in bytes.
Multiplicity	0..1
Type	INTEGER

Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsAppShutdownHookStack**
Description	**OsAppShutdownHookStack** defines the stack size of the shutdown hook in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsAppStartupHookStack**
Description	**OsAppStartupHookStack** defines the stack size of the startup hook in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

7.1.1.11. OsApplicationTrustedFunction

Parameters included	
Parameter name	**Multiplicity**
OsTrustedFunctionName	1..1
OsTrustedFunctionStacksize	0..1

Parameter Name	**OsTrustedFunctionName**
Description	Trusted function (as part of a trusted OS-Application) available to other OS-Applications. This also supersedes the OSEK OIL attribute TRUSTED in APPLICATION because the optionality of this parameter is describing that already.

Multiplicity	1..1
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC

Parameter Name	**OsTrustedFunctionStacksize**
Description	This attribute specifies the amount of stack required by the trusted function in bytes. **EB tresos AutoCore OS:** The kernel checks that the calling task or ISR has sufficient stack remaining before calling the trusted function. **It is vitally important that the stack size for trusted functions is set correctly. Too small a value means that the trusted function could overflow the task or ISR's stack region, and since it is trusted the overflow will not be caught by the memory protection mechanisms.**
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

7.1.1.12. OsCounter

Containers included		
Container name	**Multiplicity**	**Description**
OsDriver	0..1	This Container contains the information how a software counter can be incremented automatically without specifying an alarm. This configuration is only valid if the parameter Os-CounterType is set to SOFTWARE. If the container is disabled, the Os manages the counter and increments it, if configured by the user, with an alarm. If the container is enabled the OS can either use the Autosar GPT driver or a hardware module to automatically increment the counter. For the former variant, the user has to supply the GPT channel. For the latter, a hardware module has to be specified.
OsTimeConstant	0..n	Allows the user to define constants which can be e.g. used to compare time values with timer tick values. A time value will be converted to a timer tick value during generation and can be accessed later on via its OsConstName. The conversion is done by rounding time values to the nearest fitting tick value.

Parameters included	
Parameter name	**Multiplicity**
OsCounterMaxAllowed-Value	1..1
OsCounterMinCycle	1..1
OsCounterTicksPer-Base	1..1
OsCounterType	1..1
OsSecondsPerTick	0..1
OsCounterAccessin-gApplication	0..n

Parameter Name	OsCounterMaxAllowedValue
Description	Maximum possible allowed value of the system counter in ticks. When the counter reaches this value, the next advancement will cause it to restart from zero.
Multiplicity	1..1
Type	INTEGER
Origin	AUTOSAR_ECUC

Parameter Name	OsCounterMinCycle
Description	The MINCYCLE attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.
Multiplicity	1..1
Type	INTEGER
Origin	AUTOSAR_ECUC

Parameter Name	OsCounterTicksPerBase
Description	**OsCounterTicksPerBase** is a UINT32 value that specifies how many ticks of the counter represent a known unit of counting. The value of this attribute is not used by the kernel, but is made available for application purposes.
Multiplicity	1..1
Type	INTEGER
Range	<=4294967295 >=1
Origin	AUTOSAR_ECUC

Parameter Name	OsCounterType
Description	This parameter contains the natural type or unit of the counter.
Multiplicity	1..1
Type	ENUMERATION
Range	HARDWARE SOFTWARE
Origin	AUTOSAR_ECUC

Parameter Name	OsSecondsPerTick
Description	Time of one hardware tick in seconds.
Multiplicity	0..1
Type	FLOAT
Default value	0.1
Range	<=86400.0 >0.0
Origin	AUTOSAR_ECUC

Parameter Name	OsCounterAccessingApplication
Description	Reference to applications which have an access to this object.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.13. OsDriver

Containers included		
Container name	**Multiplicity**	**Description**
OsHwIncrementer	0..1	**OsHwIncrementer** specifies a hardware module that automatically increments the software counter. Specify the period of the incrementer module in the OsSecondsPerTick parameter.

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsGptChannelRef	0..1

Parameter Name	OsGptChannelRef
Description	Reference to the GPT channel.
Multiplicity	0..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.14. OsHwIncrementer

Parameters included	
Parameter name	**Multiplicity**
OsHwModule	1..1
OsIncrementerIrqLevel	1..1

Parameter Name	OsHwModule
Description	**OsHwModule** provides a list of supported hardware modules that can be used to increment a software counter.
Multiplicity	1..1
Type	ENUMERATION
Origin	Elektrobit Automotive GmbH

Parameter Name	OsIncrementerIrqLevel
Multiplicity	1..1
Type	INTEGER
Origin	Elektrobit Automotive GmbH

7.1.1.15. OsTimeConstant

Parameters included	
Parameter name	**Multiplicity**
OsConstName	1..1

Parameters included	
OsTimeValue	1..1

Parameter Name	OsConstName
Description	The name which is accessed by the application to get the OsTimeValue of the constant.
Multiplicity	1..1
Type	STRING
Origin	AUTOSAR_ECUC

Parameter Name	OsTimeValue
Description	This parameter contains the value of the constant in seconds.
Multiplicity	1..1
Type	FLOAT
Range	<=86400.0 >=0.0
Origin	AUTOSAR_ECUC

7.1.1.16. OsEvent

Parameters included	
Parameter name	Multiplicity
OsEventMask	0..1

Parameter Name	OsEventMask
Description	The **OsEventMask** attribute is a UINT64 attribute that specifies the set of bits to be associated with the event. The AUTOSAR OS kernel supports up to 32 events per task, therefore the event mask must be restricted to the lower 32 bits of the word.
Multiplicity	0..1
Type	INTEGER
Range	<=4294967295 >=1

Origin	AUTOSAR_ECUC

7.1.1.17. OsSpinlock

Parameters included	
Parameter name	**Multiplicity**
OsSpinlockAccessingApplication	1..n
OsSpinlockSuccessor	0..1
OsSpinlockLockMethod	1..1

Parameter Name	**OsSpinlockAccessingApplication**
Description	Reference to OsApplications that have an access to this object.
Multiplicity	1..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsSpinlockSuccessor**
Description	Reference to the next OsSpinlock object in the linked list. To check whether a spinlock can be occupied (in a nested way) without any danger of deadlock, a linked list of spinlocks can be defined. A spinlock can only be occupied in the order of the linked list. It is allowed to skip a spinlock. If no linked list is specified, spinlocks cannot be nested.
Multiplicity	0..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsSpinlockLockMethod**
Description	OsSpinlockLockMethod is an enumerated type whose value is one of: ▶ LOCK_NOTHING ▶ LOCK_ALL_INTERRUPTS ▶ LOCK_CAT2_INTERRUPTS ▶ LOCK_WITH_RES_SCHEDULER OsSpinlockLockMethod describes the lock method, which is additionally applied when a spinlock is taken. This method modifies priority and interrupt lock level of

	tasks, which hold this spinlock. If **LOCK_NOTHING** is chosen, taking the spinlock will not change the current task's priority or interrupt lock level. **LOCK_ALL_INTERRUPTS** will cause all interrupts to be disabled. **LOCK_CAT2_INTERRUPTS** will disable all category 2 ISRs while the spinlock is taken. **LOCK_WITH_RES_SCHEDULER** will prevent the task, holding this spinlock, from being preempted by another task. It is recommended to lock out all tasks and ISRs which could try to take a spinlock to prevent certain kinds of deadlocks.
Multiplicity	1..1
Type	ENUMERATION
Default value	LOCK_NOTHING
Range	LOCK_NOTHING LOCK_ALL_INTERRUPTS LOCK_CAT2_INTERRUPTS LOCK_WITH_RES_SCHEDULER
Origin	AUTOSAR_ECUC

7.1.1.18. OsIoc

Containers included		
Container name	**Multiplicity**	**Description**
OsIocDataTypeIncludeHeader	1..n	The container holds information about the header file to be included by the Ioc. The header file contains the definitions for the data types used by the Ioc.
OsIocCommunication	0..n	Representation of a 1:1 communication between software parts located in different OsApplications. The involved OsApplications are located on the same or on different cores.

7.1.1.19. OsIocDataTypeIncludeHeader

Parameters included	
Parameter name	**Multiplicity**
IncludeHeaderName	1..1

Parameter Name	**IncludeHeaderName**

Description	The name of the header file where data types which are used by the Ioc channels are defined in.
Multiplicity	1..1
Type	STRING

7.1.1.20. OsIocCommunication

Containers included		
Container name	**Multiplicity**	**Description**
OsIocDataType	1..n	Container holding information on the data type used for the Ioc channel. The container holds the data type name and two flags determining whether the data type is a complex type and if it is an array or string type.
OsIocCommunicationSemantics	1..1	Choice between queued and "last-is-best" semantics for this IOC communication.
OsIocReceiverProperties	1..n	Representation of receiver properties for one communication. For each OsIocCommunication (1:1 or N:1) one receiver has to be defined. This container should be instanciated within an OsIocCommunication.
OsIocSenderProperties	1..n	

Parameters included	
Parameter name	**Multiplicity**
OsIocDataTypeRef	0..n
OsIocSenderApiIsTrapping	0..1
OsIocIntraCoreLockType	0..1
OsIocUseInterCoreLock	0..1

Parameter Name	**OsIocDataTypeRef**
Description	This AUTOSAR standard parameter is not used and should not be edited.
Multiplicity	0..n
Type	FOREIGN-REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsIocSenderApiIsTrapping**
Description	If set to true, the IocSend/IocWrite functions of this channel are executed in kernel context. If set to false, these functions are executed in the context of the callers. If you use the Safety OS and the sender API is trapping, make sure you define a *READABLE* region for each application, which sends data over this IOC channel. These regions encompass those parts of the memory, from where the kernel may read the data elements passed to IocSend and IocWrite functions. They are usually defined in the linker script. The `IOC_RSA_READABLE_(application-name)` symbol defines the start address of such a range. The `IOC_RLA_READABLE_(application-name)` must point to the next location immediately following the last readable byte. In general, if a memory location needs to be protected from read accesses by certain applications or the kernel, it shall not appear in READABLE regions. Furthermore, the kernel must be able to read from READABLE regions without causing an exception. Note, the default values in the demo linker script are hardware-dependent and do not necessarily suit your application.
Multiplicity	0..1
Type	BOOLEAN
Default value	false

Parameter Name	**OsIocIntraCoreLockType**
Description	Selects the intra-core lock type that is used for write accesses to this channel. This lock type shall be used to avoid preemptions. This is recommended if multiple preemtive tasks in the sender application use this channel or in case of N:1 communication (i.e. senders on different cores) for reducing the coupling between cores. ▶ NO_LOCK ▶ INTERRUPT_LOCK
Multiplicity	0..1
Type	ENUMERATION
Default value	INTERRUPT_LOCK
Range	NO_LOCK INTERRUPT_LOCK
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsIocUseInterCoreLock**
Description	Enable or disable the use of a spin lock for inter-core synchronization. This is required in case of N:1 communication, i.e. when mutliple senders located on differ-

	ent cores exist. In single core configurations, this value should be set to false, otherwise it is ignored.
Multiplicity	0..1
Type	BOOLEAN
Default value	true
Origin	Elektrobit Automotive GmbH

7.1.1.21. OsIocDataType

Parameters included	
Parameter name	**Multiplicity**
DataTypeName	1..1
IsComplexType	1..1
IsArrayOrStringType	1..1
VariableLength	1..1

Parameter Name	**DataTypeName**
Description	Name of the data type used for this Ioc channel.
Multiplicity	1..1
Type	STRING
Default value	
Origin	Elektrobit Automotive GmbH

Parameter Name	**IsComplexType**
Description	This flag indicates, whether the data type is complex or primitive.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**IsArrayOrStringType**
Description	This flag indicates, whether the data has an array or string type.
Multiplicity	1..1

Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**VariableLength**
Description	This flag indicates, that the data type has a variable length. Primitive data types can't have variable length. The length, given at runtime, must not exceed the size of this type.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

7.1.1.22. OslocCommunicationSemantics

Containers included		
Container name	**Multiplicity**	**Description**
OslocLastIsBestCommunication	1..1	
OslocQueuedCommunication	1..1	

7.1.1.23. OslocLastIsBestCommunication

Parameters included	
Parameter name	**Multiplicity**
OslocInitValueSymbol	0..1

Parameter Name	**OslocInitValueSymbol**
Description	Symbolic value to be used as initializer for the data buffer. This symbol is generated by the RTE (e.g. as a macro), so that the IOC does not have to consider the data structure from the RTE, but can use the symbol in the generated C-Code as initializer.
Multiplicity	0..1
Type	LINKER-SYMBOL

Origin	AUTOSAR_ECUC

7.1.1.24. OsIocQueuedCommunication

Parameters included	
Parameter name	**Multiplicity**
OsIocBufferLength	1..1

Parameter Name	**OsIocBufferLength**
Description	This attribute defines the size of the IOC internal queue to be allocated for a queued communication. This configuration information shall allow the optimization of the needed memory for communications requiring buffers within the RTE and within the IOC.
Multiplicity	1..1
Type	INTEGER
Range	<=65535 >=1
Origin	AUTOSAR_ECUC

7.1.1.25. OsIocReceiverProperties

Parameters included	
Parameter name	**Multiplicity**
OsIocReceiverId	1..1
OsIocReceiverPullCB	1..1
OsIocReceivingOsApplicationRef	1..1

Parameter Name	**OsIocReceiverId**
Description	The receiver id may be used to identify a receiver. It is not evaluated by the OS generator for its internal operations and hence, its use and interpretation is not constraint by it in any way. Furthermore, it's not defined by AUTOSAR and therefore Elektrobit specific.
Multiplicity	1..1

Type	INTEGER
Range	<=255 >=0
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsIocReceiverPullCB**
Description	Callbacks are currently not supported.
Multiplicity	1..1
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC

Parameter Name	**OsIocReceivingOsApplicationRef**
Description	This attribute is a reference to the receiving OsApplication instance defined in the configuration file of the OS.
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.26. OsIocSenderProperties

Parameters included	
Parameter name	**Multiplicity**
OsIocSenderId	0..1
OsIocSendingOsApplicationRef	1..1

Parameter Name	**OsIocSenderId**
Description	Representation of a sender in a N:1 communication to distinguish between senders.
Multiplicity	0..1
Type	INTEGER
Range	<=255 >=0

Origin	AUTOSAR_ECUC

Parameter Name	**OsIocSendingOsApplicationRef**
Description	This attribute is a reference to the sending OS-Application instance defined in the configuration file of the OS.
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.27. OsIsr

Containers included		
Container name	**Multiplicity**	**Description**
OsIsrTimingProtection	0..1	**OsIsrTimingProtection** is a boolean attribute that specifies whether the kernel should apply timing protection to the ISR. When this attribute is **TRUE**, the sub-attributes **OsIsrExecutionBudget**, **OsIsrTimeFrame** and **OsIsrLockBudget** are available. They are described in the following sections.

Parameters included	
Parameter name	**Multiplicity**
OsIsrCategory	1..1
OsIsrResourceRef	0..n
OsMeasure_Max_Runtime	0..1
OsEnable_On_Startup	0..1
OsStacksize	1..1
OsIsrAccessingApplication	0..n

Parameter Name	**OsIsrCategory**
Description	**OsIsrCategory** is a UINT32 attribute that defines the IRS's Category. Only the values "CATEGORY_1" and "CATEGORY_2" are permitted.
Multiplicity	1..1
Type	ENUMERATION

Range	CATEGORY_1 CATEGORY_2
Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrResourceRef**
Description	This reference defines the resources accessed by this ISR.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsMeasure_Max_Runtime**
Description	**OsMeasure_Max_Runtime** is a boolean attribute that tells the kernel to record the longest-observed execution-time for this ISR. The value can be obtained by calling the function *OS_GetIsrMaxRuntime*.
Multiplicity	0..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsEnable_On_Startup**
Description	**OsEnable_On_Startup** is a boolean attribute that determines whether the kernel should automatically enable the interrupt source at startup. If this attribute is set to **FALSE**, the application code is responsible for enabling this interrupt source using `OS_EnableInterruptSource()` when needed.
Multiplicity	0..1
Type	BOOLEAN
Default value	true
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsStacksize**
Description	**OsStackSize** specifies the stack size of the ISR in bytes.
Multiplicity	1..1
Type	INTEGER
Range	<=2000000000

	>=0
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsIsrAccessingApplication**
Description	Reference to applications which have an access to this object.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

7.1.1.28. OsIsrTimingProtection

Containers included		
Container name	**Multiplicity**	**Description**
OsIsrResourceLock	0..n	This container contains a list of times the interrupt uses resources.

Parameters included	
Parameter name	**Multiplicity**
OsIsrAllInterruptLockBudget	0..1
OsIsrExecutionBudget	0..1
OsIsrOsInterruptLockBudget	0..1
OsIsrTimeFrame	0..1
OsIsrCountLimit	0..1

Parameter Name	**OsIsrAllInterruptLockBudget**
Description	This parameter contains the maximum time for which the ISR is allowed to lock all interrupts (via SuspendAllInterrupts() or DisableAllInterrupts()) (in seconds).
Multiplicity	0..1
Type	FLOAT
Range	<=86400.0 >0.0

Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrExecutionBudget**
Description	**OsIsrExecutionBudget** specifies, in seconds, the maximum execution time permitted for the ISR, from call to return. If the ISR is interrupted by a higher priority category 2 ISR, the interruption does not count towards the ISR's execution time. However, time spent in category 1 ISRs is counted in the time of the interrupted ISR.
Multiplicity	0..1
Type	FLOAT
Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrOsInterruptLockBudget**
Description	This parameter contains the maximum time for which the ISR is allowed to lock all Category 2 interrupts (via SuspendOSInterrupts()) (in seconds).
Multiplicity	0..1
Type	FLOAT
Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrTimeFrame**
Description	This parameter contains the minimum inter-arrival time between successive interrupts (in seconds).
Multiplicity	0..1
Type	FLOAT
Range	<=86400.0 >=0.0
Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrCountLimit**
Description	**OsIsrCountLimit** specifies the number of allowed interrupt arrivals within the time frame specified by OsIsrTimeFrame.
Multiplicity	0..1
Type	INTEGER
Default value	1
Range	>=0 <65536

Origin	Elektrobit Automotive GmbH

7.1.1.29. OsIsrResourceLock

Parameters included	
Parameter name	**Multiplicity**
OsIsrResourceLock-Budget	1..1
OsIsrResourceLockRe-sourceRef	1..1

Parameter Name	**OsIsrResourceLockBudget**
Description	This parameter contains the maximum time the interrupt is allowed to hold the giv-en resource (in seconds).
Multiplicity	1..1
Type	FLOAT
Range	<=86400.0 >0.0
Origin	AUTOSAR_ECUC

Parameter Name	**OsIsrResourceLockResourceRef**
Description	Reference to the resource the locking time is depending on
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.30. OsResource

Parameters included	
Parameter name	**Multiplicity**
OsResourceProperty	1..1
OsResourceAccessin-gApplication	0..n
OsResourceLinke-dResourceRef	0..1

Parameter Name	OsResourceProperty
Description	**RESOURCEPROPERTY** is an enumerated attribute that whose values are: <dl> <dt>STANDARD</dt> <dd>a normal resource that can be expicitly taken and released by application code</dd> <dt>LINKED</dt> <dd>a resource that is linked to another resource of type **STANDARD** or **LINKED**. The sub-attribute **LINKEDRESOURCE** specifies the resource to which it is linked.</dd> <dt>INTERNAL</dt> <dd>a resource that cannot be expicitly taken and released by application code. The resource is automatically given to a task whenever the task enters the running state.</dd> </dl>
Multiplicity	1..1
Type	ENUMERATION
Default value	STANDARD
Range	INTERNAL LINKED STANDARD
Origin	AUTOSAR_ECUC

Parameter Name	OsResourceAccessingApplication
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsResourceLinkedResourceRef
Description	The link to the resource. Must be valid if OsResourceProperty is LINKED. If OsResourceProperty is not LINKED the value is ignored.
Multiplicity	0..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.31. OsOS

Containers included		
Container name	**Multiplicity**	**Description**
OsHooks	1..1	Container to structure all hooks belonging to the OS

Containers included		
OsAutosarCustomization	0..1	The OsAutosarCustomization container and its attrributes can be used to can be use to fine-tune the OS to gain size, performance or other benefits. **Warning: Use of non-default values for these options means that the OS is not fully conformant with the Autosar specification.**
OsCpuLoadMeasurement	0..1	This container configures the CPU load measurement functionality in the AutosarOS kernel.
OsCoreConfig	0..n	

Parameters included	
Parameter name	**Multiplicity**
OsScalabilityClass	0..1
OsNumberOfCores	0..1
OsStackMonitoring	1..1
OsStatus	1..1
OsUseGetServiceId	1..1
OsUseParameterAccess	1..1
OsUseResScheduler	1..1
OsCC	0..1
OsTrace	1..1
OsExtra_Runtime_Checks	1..1
OsStartupChecks	1..1
OsServiceTrace	1..1
OsSourceOptimization	1..1
OsStackOptimization	1..1
OsProtection	1..1
OsUseLastError	1..1
OsTracebuffer	1..1
OsSchedule	0..1
OsSchedulingAlgorithm	0..1
OsTrappingKernel	0..1
OsGenerateSWCD	1..1

Parameters included	
OsInitCoreId	0..1
OsMaxNumberOfCores	1..1

Parameter Name	OsScalabilityClass
Description	A scalability class for each System Object **OS** has to be selected. In order to customize the operating system to the needs of the user and to take full advantage of the processor features the operating system can be scaled according to the scalability classes. The value is one of: ▶ SC1 ▶ SC2 ▶ SC3 ▶ SC4
Multiplicity	0..1
Type	ENUMERATION
Range	SC1 SC2 SC3 SC4
Origin	AUTOSAR_ECUC

Parameter Name	OsNumberOfCores
Description	Maximum number of cores that are controlled by EB tresos AutoCore OS.
Multiplicity	0..1
Type	INTEGER
Default value	1
Origin	AUTOSAR_ECUC

Parameter Name	OsStackMonitoring
Description	The **OsStackMonitoring** attribute is a BOOLEAN attribute that specifies whether the kernel should perform software stack monitoring at runtime. If it is set to **TRUE**, the stack monitoring is enabled.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Origin	AUTOSAR_ECUC

Parameter Name	**OsStatus**
Description	STATUS is an enumerated type whose value is one of: ▶ STANDARD ▶ EXTENDED In AUTOSAR OS there is no possibility of the system entering an undefined state because of an error in the application code. Errors are always checked for and reported. The **STATUS** setting determines how the kernel handles the error. In **STANDARD** mode OSEK/VDX does not specify how the kernel should react. In this mode AUTOSAR OS's typical reaction to a static error is to quarantine the offending task or application. In **EXTENDED** mode OSEK/VDX specifies that the system services should return certain error codes when an error is detected. AUTOSAR OS complies with this requirement.
Multiplicity	1..1
Type	ENUMERATION
Default value	STANDARD
Range	EXTENDED STANDARD
Origin	AUTOSAR_ECUC

Parameter Name	**OsUseGetServiceId**
Description	In the precompiled AUTOSAR OS kernel the *OSErrorGetServiceID()* API is always available within the *ErrorHook()*. However, if you are compiling an optimized kernel from the source code, the **USEGETSERVICEID** attribute can be used to enable or disable the feature and could result in a smaller, faster kernel.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsUseParameterAccess**
Description	In the precompiled AUTOSAR OS kernel the *OSError_x1_x2()* APIs are always available within the *ErrorHook()*. However, if you are compiling an optimized kernel from the source code, the **USEPARAMETERACCESS** attribute can be used to enable or disable the feature and could result in a smaller, faster kernel.

Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsUseResScheduler**
Description	**OsUseResScheduler** is a boolean attribute. If it is **TRUE**, the Generator creates a resource called **RES_SCHEDULER** whose resource ID is typically 0. Any task is eligible to take this resource. The ceiling priority of this resource is at least as high as the highest task priority. The OSEK/VDX API *RES_SCHEDULER* is defined in terms of this resource.
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Origin	AUTOSAR_ECUC

Parameter Name	**OsCC**
Description	Choose automatic selection or one of the following conformance classes: ▶ BCC1 ▶ BCC2 ▶ ECC1 ▶ ECC2 The precompiled AUTOSAR OS kernel always supports an **ECC2** system, but the setting here is used to check that all the tasks satisfy the desired conformance class constraints. If an optimized kernel is compiled from the sources, a lower CC setting might result in a smaller, faster kernel.
Multiplicity	0..1
Type	ENUMERATION
Range	BCC1 BCC2 ECC1 ECC2
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTrace**

Description	**OsTrace** is a boolean attribute. If it is **TRUE**, the macro OS_USE_TRACE will be passed via the Make environment to the OS library code, which enables the trace hooks for the Debug&Trace module.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsExtra_Runtime_Checks**
Description	**OsExtra_Runtime_Checks** is a boolean attribute. If it is **TRUE**, the kernel makes a range of extra checks at specific points while the system is running. This is helpful during the development phase for debugging purposes.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsStartupChecks**
Description	**OsStartupChecks** is a boolean attribute. If it is **TRUE**, the kernel makes a range of extra checks at system startup. This is helpful during the development phase to detect possible configuration errors and to ensure a coherent system state after startup.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsServiceTrace**
Description	Check this if you want to trace system calls via ORTI
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsSourceOptimization**
Description	Check this if you want to build a library optimized according to the configuration.

Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsStackOptimization**
Description	**OsStackOptimization** is an enumerated attribute that controls how the Generator optimizes task stacks across tasks and applications. The values are: ▶ NO ▶ WITHIN_APPLICATIONS ▶ GLOBAL With NO optimization, each task gets its own stack area. This option uses the most RAM but is useful to determine how much stack each individual task really uses. Optimization WITHIN_APPLICATIONS allows tasks of the same application to share a stack when the tasks types and priorities permit. GLOBAL optimization allows tasks from different applications to share stacks. This option provides the most efficient RAM footprint, but might conflict with memory protection mechanisms on some architectures.
Multiplicity	1..1
Type	ENUMERATION
Default value	GLOBAL
Range	NO WITHIN_APPLICATIONS GLOBAL
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsProtection**
Description	On microcontrollers that support memory protection the presence of non-trusted applications in the configuration will cause memory protection to be enabled. On some microcontrollers this can cause problems with debugger breakpoints in the flash memory. On such processors the **OsProtection** attribute permits you to disable the memory protection features without changing the trust status of the applications. The possible values of the **OsProtection** attribute are ON and OFF. Note that the use of this attribute does not affect the trust status of applications, nor does it affect the CPU mode in which the tasks run, so if a task performs an action that is not permitted in the user mode of the CPU, the protection system will still

	detect it. **Setting the PROTECTION attribute to any value other than ON invalidates any safety certification of the OS. The Generator produces a warning for this.**
Multiplicity	1..1
Type	ENUMERATION
Default value	ON
Range	OFF ON
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsUseLastError**
Description	**OsUseLastError** is a boolean attribute. If it is **TRUE**, the last error is stored internally and can be accessed via ORTI.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTracebuffer**
Description	**OsTracebuffer** defines the size of the trace buffer for tracing. A value of 0 disables tracing.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	>=0 <65536
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsSchedule**
Description	▶ NON ▶ FULL ▶ MIXED **NON** means that all Tasks must have their **OsTaskSchedule** attribute set to **NON**. **FULL** means that all Tasks must have their **OsTaskSchedule** attribute set to

	FULL. **MIXED** means that a mixture of Task scheduling types is permitted. The precompiled AUTOSAR OS kernel always supports mixed scheduling, but the attribute allows the generator to check that all tasks satisfy the desired scheduling constraints. If an optimized kernel is compiled from the sources, a more restrictive **OsSchedule** setting might result in a smaller, faster kernel.
Multiplicity	0..1
Type	ENUMERATION
Default value	MIXED
Range	NON FULL MIXED
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsSchedulingAlgorithm**
Description	▶ CLZ_QUEUE ▶ LINKED_LIST **CLZ_QUEUE** selects a scheduling algorithm working with one task-queue for each priority. It should be of advantage in configurations with a big number of tasks with many different priorities. **LINKED_LIST** selects the scheduling algorithms based on one single priority queue.
Multiplicity	0..1
Type	ENUMERATION
Default value	CLZ_QUEUE
Range	CLZ_QUEUE LINKED_LIST
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTrappingKernel**
Description	**OsTrappingKernel** is an optional boolean attribute. If it is **TRUE**, the kernel is entered via a Systrap mechanism. This is necessary for memory protection. If it is **FALSE**, the kernel is entered via function calls. Memory protection is not available in this case. If the optional parameter is disabled, it will be automatically enabled if non-trusted applications are found. **Note:** This parameter is only available on architectures that allow a selection between Systrap and function calls.
Multiplicity	0..1
Type	BOOLEAN

Origin	Elektrobit Automotive GmbH

Parameter Name	**OsGenerateSWCD**
Description	**OsGenerateSWCD** is a boolean attribute. If it is enabled, the OS specific software component description (SWCD) files will be generated, exporting a subset of the OS API via RTE interfaces. **Note:** Enabling this parameter will result in bigger generation and compile times. You only need to enable it if you are using software components that access OS API via RTE calls, which is unlikely.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsInitCoreId**
Description	**OsInitCoreId** designates the processor core, which will control the OS startup. If this value is disabled, the generator will choose it by itself. It depends on the target hardware and the current configuration, which one of the cores is chosen automatically. If you want a certain core to control the OS startup, then enable OsInitCoreId.
Multiplicity	0..1
Type	INTEGER
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsMaxNumberOfCores**
Description	This is the number of cores provided by the hardware.
Multiplicity	1..1
Type	INTEGER
Default value	1
Origin	Elektrobit Automotive GmbH

7.1.1.32. OsHooks

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsErrorHook	1..1
OsPostTaskHook	1..1
OsPreTaskHook	1..1
OsProtectionHook	0..1
OsShutdownHook	1..1
OsStartupHook	1..1
OsPreISRHook	1..1
OsPostISRHook	1..1

Parameter Name	OsErrorHook
Description	**OsErrorHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void ErrorHook(StatusType errorcode)* whenever an error is detected, unless the error was caused within **ErrorHook()**.
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Origin	AUTOSAR_ECUC

Parameter Name	OsPostTaskHook
Description	**OsPostTaskHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void PostTask- Hook(void)* when a task is about to leave the running state.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	OsPreTaskHook
Description	**OsPreTaskHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void PreTaskHook(void)* just before task execution resumes, but after the incoming task has entered the running state.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Origin	AUTOSAR_ECUC

Parameter Name	**OsProtectionHook**
Description	**OsProtectionHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *ProtectionReturn-Type ProtectionHook(StatusType errorcode)* whenever a protection violation is detected, unless the error was caused within *Protection-Hook()*. The return value of the *ProtectionHook()* function can be one of: ▶ PRO_TERMINATETASKISR ▶ PRO_TERMINATEAPPL ▶ PRO_TERMINATEAPPL_RESTART ▶ PRO_SHUTDOWN ▶ PRO_IGNORE
Multiplicity	0..1
Type	BOOLEAN
Default value	true
Origin	AUTOSAR_ECUC

Parameter Name	**OsShutdownHook**
Description	**OsShutdownHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void Shutdown- Hook(StatusType errorcode)* when the system has been shut down. The parameter is the value of the error code passed to *ShutdownOS()*
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Origin	AUTOSAR_ECUC

Parameter Name	**OsStartupHook**
Description	**OsStartupHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void StartupHook(void)* immediately before starting the scheduler.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	**OsPreISRHook**

Description	**OsPreISRHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void PreIsrHook(os_isrid_t isrid)* just before an ISR is called. The parameter is the ID of the ISR. For each ISR, the Generator defines a macro whose name is the OIL name of the ISR and whose value is its ISR ID.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsPostISRHook**
Description	**OsPostISRHook** is a boolean attribute. If it is **TRUE**, the kernel calls the user-supplied *void PostIsrHook(os_isrid_t isrid)* just after an ISR returns. The parameter is the ID of the ISR. For each ISR, the Generator defines a macro whose name is the OIL name of the ISR and whose value is its ISR ID.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

7.1.1.33. OsAutosarCustomization

Parameters included	
Parameter name	**Multiplicity**
OsExceptionHandling	1..1
OsErrorHandling	1..1
OsStrictServiceProtection	1..1
OsCat1DirectCall	1..1
OsFastInterruptLocking	0..1
OsInterruptLockingChecks	1..1
OsCallIsr	1..1
OsCallAppErrorHook	1..1
OsCallAppStartupShutdownHook	1..1

Parameters included	
OsPermitSystemObjects	1..1
OsUserTaskReturn	1..1

Parameter Name	OsExceptionHandling
Description	This option can be used to disable the execption handling. If set to **FALSE**, a minimal exception vector table is used, which can be adapted if necessary. *The exact behaviour is architecture-dependent. On some architectures this option may have no effect because the OS relies on some exceptions to perform its duties.* **Setting the option to FALSE will remove the ability of the OS to detect and correctly react to protection errors.**
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Origin	Elektrobit Automotive GmbH

Parameter Name	OsErrorHandling
Description	This option can be used to restrict the amount of error handling that is performed by the OS. The permitted values are MINIMAL, AUTOSAR and FULL. If you choose **MINIMAL**, the error handler *OS_Error()* is never called, and the default error code is returned to the user. Choosing this option means that error and protection hooks cannot be called and the correct action after an error (such as killing a task) does not take place. If you choose **AUTOSAR**, the error handler *OS_Error()* will be called for all errors except those that occur in System Services that do not return **StatusType**. This is the Autosar-conformant option. If you choose **FULL**, the error handler *OS_Error()* will be called for all errors, including those that occur in System Services that do not return **StatusType**. It will also call the error hook for those errors.
Multiplicity	1..1
Type	ENUMERATION
Default value	AUTOSAR
Range	MINIMAL AUTOSAR FULL
Origin	Elektrobit Automotive GmbH

Parameter Name	OsStrictServiceProtection

Description	Setting this option to **FALSE** disables most of the calling-context checks in the System Services. The OS will then only check the calling context when it is necessary for the correct functioning of the OS.
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsCat1DirectCall**

Description	This parameter selects whether a category 1 ISR is called directly or via the operating system's category 1 interrupt handler. When this option is disabled, the operating system's category 1 handler is used as the entry in the interrupt vector table. This handler performs a context save, switches to the kernel stack (if applicable, depending on the architecture) and sets internal context data for operating system for service protection. This setting is conformant with the Autosar specification. If you enable this option, the configured ISR is entered directly into the interrupt vector table. This allows a fast entry into the ISR, but Autosar service protection fails. Furthermore, use of operating system APIs is not supported, because the APIs do not know that they have been called from a category 1 ISR and may not function correctly. This applies even to the interrupt locking APIs (SuspendAllInterrupts/ResumeAllInterrupts etc.). In this case it is recommended to enable also OsFastInterruptLocking, as these routines bypass the caller checks. Please note that on several architectures the ISR routine needs to be prefixed with an __interrupt keyword (check compiler documentation for further details) which saves the context prior to entering the ISR. You can pass the keyword to the ISR prototype by defining the macro OS_INTERRUPT_KEYWORD prior to including Os.h. For example, if the toolchain uses the keyword "__interrupt", use the following code:
	
	#define OS_INTERRUPT_KEYWORD __interrupt
	#include "Os.h"
	
	__interrupt ISR(foo)
	{
	  (...)
	}

Multiplicity	1..1
Type	BOOLEAN
Default value	false

Origin	Elektrobit Automotive GmbH

Parameter Name	**OsFastInterruptLocking**
Description	Enabling this option replaces the OS interrupt locking mechanism with a faster, but not Autosar compliant mechanism. It affects the API functions DisableAllInterrupts()/EnableAllInterrupts(), SuspendAllInterrupts()/ResumeAllInterrupts() and SuspendOsInterrupts()/ResumeOsInterrupts(). When enabled, the kernel bypasses all checks and syscall handlers and directly disables/enables the interrupts upon calling the API functions. **Caveat:** In memory-protected systems, the application using the fast locking functions must be trusted, i.e. have supervisor privileges. Calling the functions from a non-trusted application will lead to an exception.
Multiplicity	0..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsInterruptLockingChecks**
Description	▶ **MINIMAL**: Select **MINIMAL** to only check the interrupt lock status when it affects the kernel's operation. The interrupt lock status affects the kernel's operation e.g. in the functions `GetResource` and `ReleaseResource`. ▶ **EXTRACHECK**: Select **EXTRACHECK** to check the interrupt lock status in all API functions which may cause a task switch. ▶ **AUTOSAR**: Select **AUTOSAR** to be fully compliant with the AUTOSAR specification. **Tasks always start with interrupts enabled** The interrupt lock status is considered to be part of the task's context. This means that each newly activated task starts with interrupts enabled.
Multiplicity	1..1
Type	ENUMERATION
Default value	AUTOSAR
Range	MINIMAL EXTRACHECK AUTOSAR
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsCallIsr**

Description	Setting this option to **DIRECTLY** causes the OS to call all category 2 ISRs directly rather than via a wrapper function. This means that all ISRs (even non-trusted) run with the permissions of the OS, and ISRs cannot be killed if they cause a protection fault.
Multiplicity	1..1
Type	ENUMERATION
Default value	VIA_WRAPPER
Range	DIRECTLY VIA_WRAPPER
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsCallAppErrorHook**
Description	Setting this option to DIRECTLY causes the OS to call all application error hooks directly rather than via a wrapper function. This means that all error hooks (even those belonging to non-trusted applications) run with the permissions of the OS, and the error hooks cannot be killed if they cause a protection fault. **The global ErrorHook and ProtectionHook functions are always called directly.**
Multiplicity	1..1
Type	ENUMERATION
Default value	VIA_WRAPPER
Range	DIRECTLY VIA_WRAPPER
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsCallAppStartupShutdownHook**
Description	Setting this option to **DIRECTLY** causes the OS to call all application startup and shutdown hooks directly rather than via a wrapper function. This means that all these hooks (even those belonging to non-trusted applications) run with the permissions of the OS, and the hooks cannot be killed if they cause a protection fault. **The global StartupHook and ShutdownHook are always called directly.**
Multiplicity	1..1
Type	ENUMERATION
Default value	VIA_WRAPPER
Range	DIRECTLY VIA_WRAPPER
Origin	Elektrobit Automotive GmbH

Parameter Name	OsPermitSystemObjects
Description	Setting this option to **TRUE** inhibits the check that, if an OS-Application exists, all Tasks and ISRs must belong to an OS-Application. Objects that do not belong to an application are known as "system objects" and are always trusted.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	OsUserTaskReturn
Description	The *OS_MissingTerminateTask()* function is entered if a task returns from its main function without calling *TerminateTask()* or *ChainTask()*. This optimisation option controls how *OS_MissingTerminateTask()* handles the error. Setting this option to LOOP configures *OS_MissingTerminateTask()* to be a simple endless loop. If you know that none of your tasks can ever return from its main function you can select this option to save a little code space. However, if a task ever returns without calling *TerminateTask()*, or *TerminateTask()* returns unexpectedly (for example if the task still occupies a resource or has disabled interrupts) it will remain in an endless loop. Setting this option to KILL_TASK configures *OS_MissingTerminateTask()* to enter the kernel and execute *OS_KernTaskReturn()*. *OS_KernTaskReturn()* will either call the error handler to kill the task or, if error handling is disabled, kill the task itself. If this fails for any reason, *OS_MissingTerminateTask()* will try to shut down the OS. If this fails, too, there is still an endless loop to prevent the task from executing undefined code.
Multiplicity	1..1
Type	ENUMERATION
Default value	KILL_TASK
Range	KILL_TASK LOOP
Origin	Elektrobit Automotive GmbH

7.1.1.34. OsCpuLoadMeasurement

Parameters included	
Parameter name	**Multiplicity**
OsCpuLoadIntervalDuration	1..1

Parameters included	
OsCpuLoadNumInter-vals	1..1
OsCpuLoadRounding	1..1

Parameter Name	OsCpuLoadIntervalDuration
Description	The duration of the interval between the CPU load measurement points in [s]. Several of these intervals are combined to form a window in which the average and peak CPU load is calculated.
Multiplicity	1..1
Type	FLOAT
Default value	0.1
Range	<=86400.0 >0.0
Origin	Elektrobit Automotive GmbH

Parameter Name	OsCpuLoadNumIntervals
Description	The number of intervals that are used to calculate the average and peak CPU load.
Multiplicity	1..1
Type	INTEGER
Default value	10
Range	<=4294967295 >=1
Origin	Elektrobit Automotive GmbH

Parameter Name	OsCpuLoadRounding
Description	Specifies the way how the average is rounded in the integer CPU load calculation. DOWN will use the lower result, UP rounds up. NEAREST adds 1/2 to find a mean value.
Multiplicity	1..1
Type	ENUMERATION
Default value	NEAREST
Range	DOWN NEAREST

	UP
Origin	Elektrobit Automotive GmbH

7.1.1.35. OsCoreConfig

Parameters included	
Parameter name	**Multiplicity**
OsCoreId	1..1

Parameter Name	**OsCoreId**
Multiplicity	1..1
Type	INTEGER
Default value	0
Origin	Elektrobit Automotive GmbH

7.1.1.36. OsScheduleTable

Containers included		
Container name	**Multiplicity**	**Description**
OsScheduleTableAutostart	0..1	**OsScheduleTableAutostart** is a boolean attribute whose value specifies whether the alarm shall be started automatically when the kernel starts. If the value is **TRUE**, the **OsAppmode** sub-attribute specifies in which application modes the task shall be automatically started, and the sub-attribute OsScheduleTableOffset specifies the time at which the first event of the schedule shall take place. The OsScheduleTableOffset is specified in ticks or nanoseconds depending on the **UNIT** attribute of the schedule table.
OsScheduleTableExpiryPoint	1..n	The point on a Schedule Table at which the OS activates tasks and/or sets events
OsScheduleTableSync	0..1	This parameter specifies the synchronization parameters of the schedule table.

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsScheduleTableDuration	1..1
OsScheduleTableRepeating	1..1
OsSchTblAccessingApplication	0..n
OsScheduleTableCounterRef	1..1
OsTimeUnit	0..1

Parameter Name	OsScheduleTableDuration
Description	The **OsScheduleTableDuration** attribute specifies the length of time for which the schedule table runs, from start to finish. For periodic schedule tables, it is the period. The **OsScheduleTableDuration** sub-attribute is specified in nanoseconds or ticks depending on the UNIT attribute of the schedule table.
Multiplicity	1..1
Type	INTEGER
Default value	0
Origin	AUTOSAR_ECUC

Parameter Name	OsScheduleTableRepeating
Description	The **OsScheduleTableRepeating** attribute specifies whether the schedule table is periodic. <dl> <dt>TRUE</dt> <dd>periodic schedule tables repeat indefinitely until explicitly stopped</dd> <dt>FALSE</dt> <dd>the schedule table processing stops when the final expiry point is processed</dd> </dl>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	AUTOSAR_ECUC

Parameter Name	OsSchTblAccessingApplication
Description	Reference to applications which have an access to this object.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsScheduleTableCounterRef
Description	This parameter contains a reference to the counter which drives the schedule table.Each Schedule Table must be associated with exactly one Counter.
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsTimeUnit
Description	**OsTimeUnit** contains the time unit type used for this schedule table.
Multiplicity	0..1
Type	ENUMERATION
Default value	TICKS
Range	NANOSECONDS TICKS
Origin	Elektrobit Automotive GmbH

7.1.1.37. OsScheduleTableAutostart

Parameters included	
Parameter name	**Multiplicity**
OsScheduleTableAutostartType	1..1
OsScheduleTableAppModeRef	1..n
OsScheduleTableStartValue	1..1

Parameter Name	OsScheduleTableAutostartType
Description	This specifies the type of the autostart for the schedule table.
Multiplicity	1..1
Type	ENUMERATION
Default value	RELATIVE
Range	ABSOLUTE RELATIVE

	SYNCHRON
Origin	AUTOSAR_ECUC

Parameter Name	**OsScheduleTableAppModeRef**
Description	Reference in which application modes the schedule table should be started during startup
Multiplicity	1..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsScheduleTableStartValue**
Description	Value depending on **OsScheduleTableAutostartType**: ▶ ABSOLUTE: Absolute autostart tick value when the schedule table starts. ▶ RELATIVE: Relative offset in ticks when the schedule table starts.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295 >=0
Origin	AUTOSAR_ECUC

7.1.1.38. OsScheduleTableExpiryPoint

Containers included		
Container name	**Multiplicity**	**Description**
OsScheduleTableEventSetting	0..n	Event that is triggered by that schedule table.
OsScheduleTableTaskActivation	0..n	Task that is triggered by that schedule table.
OsScheduleTblAdjustableExpPoint	0..1	Adjustable expiry point

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsScheduleTblExpPointOffset	1..1

Parameter Name	OsScheduleTblExpPointOffset
Description	The offset from zero (in ticks) at which the expiry point is to be processed.
Multiplicity	1..1
Type	INTEGER
Origin	AUTOSAR_ECUC

7.1.1.39. OsScheduleTableEventSetting

Parameters included	
Parameter name	**Multiplicity**
OsScheduleTableSetEventRef	1..1
OsScheduleTableSetEventTaskRef	1..1

Parameter Name	OsScheduleTableSetEventRef
Description	Reference to event that will be set by action
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	OsScheduleTableSetEventTaskRef
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.40. OsScheduleTableTaskActivation

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsScheduleTableActivateTaskRef	1..1

Parameter Name	OsScheduleTableActivateTaskRef
Description	Reference to task that will be activated by action
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.41. OsScheduleTblAdjustableExpPoint

Parameters included	
Parameter name	**Multiplicity**
OsScheduleTableMaxLengthen	1..1
OsScheduleTableMaxShorten	1..1

Parameter Name	OsScheduleTableMaxLengthen
Description	The maximum positive adjustment that can be made to the expiry point offset specified in nanoseconds or ticks depending on the UNIT attribute of the schedule table.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295 >=0
Origin	AUTOSAR_ECUC

Parameter Name	OsScheduleTableMaxShorten
Description	The maximum negative adjustment that can be made to the expiry point offset specified in nanoseconds or ticks depending on the UNIT attribute of the schedule table.

Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<=4294967295 >=0
Origin	AUTOSAR_ECUC

7.1.1.42. OsScheduleTableSync

Parameters included	
Parameter name	**Multiplicity**
OsScheduleTblExplicit-Precision	0..1
OsScheduleTblSyncS-trategy	1..1

Parameter Name	**OsScheduleTblExplicitPrecision**
Description	**OsScheduleTblExplicitPrecision** defines the deviation threshold for consider-ing a schedule table to be "synchronous". This parameter is only needed if explicit synchronisation is used.
Multiplicity	0..1
Type	INTEGER
Range	<=4294967295 >=0
Origin	AUTOSAR_ECUC

Parameter Name	**OsScheduleTblSyncStrategy**
Description	AUTOSAR OS provides support for synchronisation in two ways: explicit and im-plicit. ▶ **EXPLICIT**: The schedule table is driven by an OS counter but processing needs to be synchronized with a different counter which is not an OS counter object. The API function **SyncScheduleTable()** provides the synchronization count to the schedule table. Expiry points with **OsScheduleTblAdjustable-ExpPoint** configuration are used to adjust the schedule table to the synchro-nization count.

	▶ **IMPLICIT**: The counter driving the schedule table is the counter with which synchronisation is required. ▶ **NONE**: No support for synchronisation. (default)
Multiplicity	1..1
Type	ENUMERATION
Default value	NONE
Range	EXPLICIT IMPLICIT NONE
Origin	AUTOSAR ECUC

7.1.1.43. OsTask

Containers included		
Container name	**Multiplicity**	**Description**
OsTaskAutostart	0..1	**OsTaskAutostart** is a boolean attribute whose value specifies whether the task shall be started automatically when the kernel starts. If the value is **TRUE**, the **OsTaskAppModeRef** sub-attribute specifies in which application modes the task shall be automatically started.
OsTaskTimingProtection	0..1	OsTaskTimingProtection is a boolean attribute that specifies whether the kernel should apply timing protection to the task. When this attribute is **TRUE**, the sub-attributes EXECUTIONBUDGET, TIMEFRAME and LOCKINGTIME are available.

Parameters included	
Parameter name	**Multiplicity**
OsTaskActivation	1..1
OsTaskPriority	1..1
OsMeasure_Max_Runtime	0..1
OsTaskAccessingApplication	0..n
OsTaskEventRef	0..n
OsTaskResourceRef	0..n

Parameters included	
OsTaskUse_Hw_Fp	0..1
OsTaskCallScheduler	0..1
OsTaskType	0..1
OsStacksize	1..1
OsTaskSchedule	1..1

Parameter Name	OsTaskActivation
Description	**ACTIVATION** is a UINT32 attribute whose value defines the maximum number of activations that a task can have at any one time.
Multiplicity	1..1
Type	INTEGER
Default value	1
Range	<=255 >=1
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskPriority
Description	**OsTaskPriority** is a UINT32 attribute whose value defines the relative base priority of the task. The lowest priority is zero; larger values correspond to higher priorities. The values given for the **OsTaskPriority** attribute only specify a relative ordering. The actual values configured for the kernel by the Generator can be different from those specified.
Multiplicity	1..1
Type	INTEGER
Range	<=255 >=0
Origin	AUTOSAR_ECUC

Parameter Name	OsMeasure_Max_Runtime
Description	**OsMeasure_Max_Runtime** is a boolean attribute that tells the kernel to record the longest-observed executiontime for this task. The value can be obtained by calling the function *OS_GetTaskMaxRuntime*.
Multiplicity	0..1
Type	BOOLEAN

Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTaskAccessingApplication**
Description	Reference to applications which have an access to this object.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsTaskEventRef**
Description	This reference defines the list of events the extended task may react on.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsTaskResourceRef**
Description	This reference defines a list of resources accessed by this task.
Multiplicity	0..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

Parameter Name	**OsTaskUse_Hw_Fp**
Description	**OsTaskUse_Hw_Fp** is a boolean attribute that tells the kernel whether to provide a full floating-point environment for the task. The implementation of floating-point environments is architecture-dependent. Please refer to your Architecture Supplement.
Multiplicity	0..1
Type	BOOLEAN
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTaskCallScheduler**
Description	The **OsTaskCallScheduler** attribute informs the generator whether the task calls the *Schedule()* service. If **OsTaskCallScheduler** is set to **NO**, the generator assumes that *Schedule()* is never called by the task. If it is set to **YES** or to **DONTKNOW**, the generator assumes that *Schedule()* may be called. This information is used to determine which tasks are able to preempt each other.

Multiplicity	0..1
Type	ENUMERATION
Range	DONTKNOW YES NO
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTaskType**
Description	**OsTaskType** is an enumerated type whose value is one of: ▶ BASIC ▶ EXTENDED BASIC specifies that the task is a basic task. EXTENDED specifies that the task is an extended task.
Multiplicity	0..1
Type	ENUMERATION
Range	BASIC EXTENDED
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsStacksize**
Description	**OsStacksize** specifies the stack size of the task in bytes. Note that the generator adds an overhead for saving the task context on the stack during task switches, depending on the task and Os configuration.
Multiplicity	1..1
Type	INTEGER
Range	<=2000000000 >=0
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTaskSchedule**
Description	OsTaskSchedule is an enumerated type whose value is one of: ▶ NON ▶ FULL

	FULL specifies that the task is preemptable. **NON** specifies that the task is not preemptable.
Multiplicity	1..1
Type	ENUMERATION
Default value	FULL
Range	FULL NON
Origin	AUTOSAR_ECUC

7.1.1.44. OsTaskAutostart

Parameters included	
Parameter name	**Multiplicity**
OsTaskAppModeRef	1..n

Parameter Name	**OsTaskAppModeRef**
Description	Reference to application modes in which that task is activated on startup of the OS
Multiplicity	1..n
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.1.45. OsTaskTimingProtection

Containers included		
Container name	**Multiplicity**	**Description**
OsTaskResourceLock	0..n	This parameter contains the worst case time between getting and releasing a given resource (in seconds).

Parameters included	
Parameter name	**Multiplicity**
OsTaskAllInterruptLock-Budget	0..1

Parameters included	
OsTaskExecutionBudget	0..1
OsTaskOsInterruptLockBudget	0..1
OsTaskTimeFrame	0..1
OsTaskCountLimit	0..1

Parameter Name	OsTaskAllInterruptLockBudget
Description	This parameter contains the maximum time for which the task is allowed to lock all interrupts (via SuspendAllInterrupts() or DisableAllInterrupts()) (in seconds).
Multiplicity	0..1
Type	FLOAT
Range	<=86400.0 >=0.0
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskExecutionBudget
Description	**OsTaskExecutionBudget** specifies, in seconds, the maximum execution time permitted for the task, from activation to termination. If the task is interrupted by a higher priority task or a category 2 ISR, the interruption does not count towards the task's execution time. However, time spent in category 1 ISRs is counted in the time of the interrupted task. An extended task's execution timer is stopped when it enters the WAITING state, and is restarted from the beginning when the event occurs. Waiting for an event that is already pending also restarts the execution timer from the beginning.
Multiplicity	0..1
Type	FLOAT
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskOsInterruptLockBudget
Description	This parameter contains the maximum time for which the task is allowed to lock all Category 2 interrupts (via SuspendOSInterrupts()) (in seconds).
Multiplicity	0..1
Type	FLOAT
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskTimeFrame
Description	The minimum inter-arrival time between activations and/or releases of a task (in seconds).
Multiplicity	0..1
Type	FLOAT
Range	<=86400.0 >=0.0
Origin	AUTOSAR_ECUC

Parameter Name	OsTaskCountLimit
Description	**OsTaskCountLimit** specifies the number of allowed task arrivals within the time frame specified by OsTaskTimeFrame.
Multiplicity	0..1
Type	INTEGER
Default value	1
Range	>=0 <65536
Origin	Elektrobit Automotive GmbH

7.1.1.46. OsTaskResourceLock

Parameters included	
Parameter name	**Multiplicity**
OsTaskResourceLock-Budget	1..1
OsTaskResourceLock-ResourceRef	1..1

Parameter Name	OsTaskResourceLockBudget
Description	This parameter contains the maximum time the task is allowed to lock the resource (in seconds)
Multiplicity	1..1
Type	FLOAT
Range	<=86400.0

	>0.0
Origin	AUTOSAR_ECUC

Parameter Name	**OsTaskResourceLockResourceRef**
Description	Reference to the resource used by the task
Multiplicity	1..1
Type	REFERENCE
Origin	AUTOSAR_ECUC

7.1.2. Configuration parameters for AUTOSAR release 4.0

7.1.3. Configuration parameters for the Microkernel

Containers included		
Container name	**Multiplicity**	**Description**
OsApplication	0..n	
OsIsr	0..n	
OsTask	0..n	
OsMicrokernel	0..1	Configuration of the micro kernel.
OsScheduleTable	0..n	

7.1.3.1. OsApplication

Containers included		
Container name	**Multiplicity**	**Description**
OsApplicationTrusted-Function	0..n	

Parameters included	
Parameter name	**Multiplicity**

Parameters included	
OsAppMkPermitShut-downOS	1..1
OsAppMkPermitShut-downAllCores	1..1
OsAppMkCreateMemo-ryRegion	1..1
OsAppMkMemoryRe-gionRef	0..n
OsAppAccessingAppli-cation	0..n

Parameter Name	OsAppMkPermitShutdownOS
Description	This parameter configures whether this application is allowed to call the Shut-downOS() service.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	OsAppMkPermitShutdownAllCores
Description	This parameter configures whether this application is allowed to call the Shutdow-nAllCores() service.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	OsAppMkCreateMemoryRegion
Description	If this is checked, the Generator creates a memory region which is shared with all runnable elements (TASKS/ISRs) of this application. For this memory region, a set of references to symbols is generated using the name of the application as stem and prefixing it with ▶ `MK_RSA_` for the start address, ▶ `MK_RLA_` for the limit address, ▶ `MK_BSA_` for the start of implicitly-initialized data, ▶ `MK_RDA_` for the initialization data image.
Multiplicity	1..1

Type	BOOLEAN
Default value	true

Parameter Name	**OsAppMkMemoryRegionRef**
Description	This reference defines the list of memory regions associated to this Application.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsAppAccessingApplication**
Description	Reference to applications which have an access to this object.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

7.1.3.2. OsApplicationTrustedFunction

Parameters included	
Parameter name	**Multiplicity**
OsTfMkExcludeAppRegions	1..1
OsTfMkMemoryRegionRef	0..n
OsTfMkThreadModeOverride	0..1
OsTfUse_Hw_Fp	0..1

Parameter Name	**OsTfMkExcludeAppRegions**
Description	If this is checked, this trusted function will not have access to the memory regions of the application it belongs to.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**OsTfMkMemoryRegionRef**

Description	This reference defines the list of memory regions associated to this trusted function.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTfMkThreadModeOverride**
Description	Processor mode in which this trusted function's thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTfUse_Hw_Fp**
Description	**OsTfUse_Hw_Fp** is a boolean attribute that tells the microkernel whether to provide a full floating-point environment for the trusted function.
Multiplicity	0..1
Type	BOOLEAN
Origin	Elektrobit Automotive GmbH

7.1.3.3. OsIsr

Parameters included	
Parameter name	**Multiplicity**
OsIsrMkCreateMemoryRegion	1..1

Parameters included	
OsIsrMkExcludeAppRegions	1..1
OsIsrMkMemoryRegionRef	0..n
OsIsrMkThreadModeOverride	0..1
OsIsrUse_Hw_Fp	0..1

Parameter Name	OsIsrMkCreateMemoryRegion
Description	If this is checked, the generator creates a memory region for exclusive use by this ISR. For this memory region, a set of references to symbols is generated using the name of the ISR as stem and prefixing it with ▶ `MK_RSA_` for the start address, ▶ `MK_RLA_` for the limit address, ▶ `MK_BSA_` for the start of implicitly-initialized data, ▶ `MK_RDA_` for the initialization data image.
Multiplicity	1..1
Type	BOOLEAN
Default value	true

Parameter Name	OsIsrMkExcludeAppRegions
Description	If this is checked, this ISR will not have access to the memory regions of the application it belongs to. If it is not part of an application, this setting has no effect.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	OsIsrMkMemoryRegionRef
Description	This reference defines the list of memory regions associated to this ISR.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

Parameter Name	OsIsrMkThreadModeOverride

Description	Processor mode in which this ISR's thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsIsrUse_Hw_Fp**
Description	**OsIsrUse_Hw_Fp** is a boolean attribute that tells the microkernel whether to provide a full floating-point environment for the ISR.
Multiplicity	0..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

7.1.3.4. OsTask

Parameters included	
Parameter name	**Multiplicity**
OsTaskMkCreateMemoryRegion	1..1
OsTaskMkExcludeAppRegions	1..1
OsTaskMkMemoryRegionRef	0..n
OsTaskMkThreadModeOverride	0..1

Parameter Name	**OsTaskMkCreateMemoryRegion**

Description	If this is checked, the generator creates a memory region for exclusive use by this task. For this memory region, a set of references to symbols is generated using the name of the task as stem and prefixing it with ▶ `MK_RSA_` for the start address, ▶ `MK_RLA_` for the limit address, ▶ `MK_BSA_` for the start of implicitly-initialized data, ▶ `MK_RDA_` for the initialization data image.
Multiplicity	1..1
Type	BOOLEAN
Default value	true

Parameter Name	**OsTaskMkExcludeAppRegions**
Description	If this is checked, this task will not have access to the memory regions of the application it belongs to. If it is not part of an application, this setting has no effect.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**OsTaskMkMemoryRegionRef**
Description	This reference defines the list of memory regions associated to this task.
Multiplicity	0..n
Type	REFERENCE
Origin	Elektrobit Automotive GmbH

Parameter Name	**OsTaskMkThreadModeOverride**
Description	Processor mode in which this TASK's thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION

Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

7.1.3.5. OsMicrokernel

Containers included		
Container name	**Multiplicity**	**Description**
MkFunction	0..1	Function configurations.
MkStack	0..1	Stack configurations.
MkMemoryProtection	0..1	Memory protection configurations.
MkOptimization	0..1	Optimization options.
MkThreadCustomization	1..1	Customization options for microkernel internal threads.

7.1.3.6. MkFunction

Parameters included	
Parameter name	**Multiplicity**
MkInitFunction	0..1
MkIdleFunction	0..1
MkShutdownFunction	0..1
MkPanicStopFunction	0..1

Parameter Name	**MkInitFunction**
Description	**MkInitFunction** denotes the function that is called at system initialisation.
Multiplicity	0..1
Type	FUNCTION-NAME
Default value	main
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkIdleFunction**

Description	**MkIdleFunction** denotes the idle function. This function is executed when the system is Idle. It shall never return.
Multiplicity	0..1
Type	FUNCTION-NAME
Default value	MK_Idle
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkShutdownFunction**
Description	**MkShutdownFunction** denotes the shutdown function. This function is executed when the system shuts down. It shall never return.
Multiplicity	0..1
Type	FUNCTION-NAME
Default value	MK_Idle
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkPanicStopFunction**
Description	**MkPanicStopFunction** denotes the user startup panic stop function. This function is executed if a fatal error occurs during startup when a normal shutdown is not possible. The function must accept one parameter of type `mk_panic_t`. It shall never return.
Multiplicity	0..1
Type	FUNCTION-NAME
Origin	Elektrobit Automotive GmbH

7.1.3.7. MkStack

Parameters included	
Parameter name	**Multiplicity**
MkInitializeStacks	1..1
MkIdleStack	0..1
MkKernStack	0..1
MkOsStack	0..1
MkErrorHookStack	0..1
MkProtectionHookStack	0..1

Parameters included	
MkShutdownHookStack	0..1

Parameter Name	MkInitializeStacks
Description	**MkInitializeStacks** is a boolean attribute that tells the generator whether to initialize the stacks with a magic value or not. Initializing the stacks with a magic pattern allows to check the worst-observed stack usage at runtime, at the expense of a longer startup due to the time required to initialize the stacks.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

Parameter Name	MkIdleStack
Description	**MkIdleStack** defines the stack size of the idle thread in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

Parameter Name	MkKernStack
Description	**MkKernStack** defines the stack size of the kernel in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000 >=1
Origin	Elektrobit Automotive GmbH

Parameter Name	MkOsStack
Description	**MkOsStack** defines the stack size of all QM-OS threads in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000

	>=1
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkErrorHookStack**
Description	**MkErrorHookStack** defines the stack size used for the error hook in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000
	>=1
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkProtectionHookStack**
Description	**MkProtectionHookStack** defines the stack size used for the protection hook in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000
	>=1
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkShutdownHookStack**
Description	**MkShutdownHookStack** defines the stack size used for the shutdown hook in bytes.
Multiplicity	0..1
Type	INTEGER
Range	<=2000000000
	>=1
Origin	Elektrobit Automotive GmbH

7.1.3.8. MkMemoryProtection

Containers included		
Container name	**Multiplicity**	**Description**

Containers included		
MkMemoryRegion	0..n	**MkMemoryRegion** objects are used to provide user defined memory regions. These memory regions can be referenced by tasks, ISRs and applications. For each memory region, a set of references to symbols is generated using the name of the region as stem and prefixing it with ▶ `MK_RSA_` for the start address, ▶ `MK_RLA_` for the limit address, ▶ `MK_BSA_` for the start of implicitly-initialized data, ▶ `MK_RDA_` for the initialization data image. The symbols beginning with `MK_BSA_` and `MK_RDA_` are only generated if memory region initialization is selected by checking **MkMemoryRegionInitialize**.

7.1.3.9. MkMemoryRegion

Parameters included	
Parameter name	**Multiplicity**
MkMemoryRegionFlags	0..1
MkMemoryRegionInitialize	1..1
MkMemoryRegionGlobal	1..1
MkMemoryRegionInitThreadAccess	1..1
MkMemoryRegionIdleThreadAccess	1..1
MkMemoryRegionOsThreadAccess	1..1
MkMemoryRegionErrorHookAccess	1..1
MkMemoryRegionProtHookAccess	1..1
MkMemoryRegionShutdownHookAccess	1..1

Parameters included	
MkMemoryRegionShutdownAccess	1..1
MkMemoryRegionKernelAccess	1..1
MkMemoryRegionInitializePerCore	1..1

Parameter Name	MkMemoryRegionFlags
Description	Access restriction flags for each memory region have to be selected. The value is one of: ▶ READ ▶ READ_WRITE ▶ READ_EXECUTE ▶ EXECUTE **READ** means that only read access to this memory region is allowed. **READ_WRITE** means that read and write access to this memory region is allowed. **READ_EXECUTE** means that read access and code execution is allowed for this memory region. **EXECUTE** means that only execute access to this memory region is allowed. READ permissions may be required to execute code, because compilers embed constants in code. If EXECUTE is not applicable on your target hardware, the generator will report an error.
Multiplicity	0..1
Type	ENUMERATION
Default value	READ_WRITE
Range	READ READ_WRITE READ_EXECUTE EXECUTE
Origin	Elektrobit Automotive GmbH

Parameter Name	MkMemoryRegionInitialize
Description	Flag determining whether the memory region contains data that has to be initialized at startup. Checking this flag will result in generating refrences to the symbols `MK_BSA_name` and `MK_RDA_name` (where `name` is the name of the memory region) which are used for memory region initialization.

Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionGlobal**
Description	Flag determining whether the memory region is globally available to all threads.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionInitThreadAccess**
Description	Flag determining whether the memory region can be accessed from the initial thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionIdleThreadAccess**
Description	Flag determining whether the memory region can be accessed from the idle thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionOsThreadAccess**
Description	Flag determining whether the memory region can be accessed from the QM-OS thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionErrorHookAccess**
Description	Flag determining whether the memory region can be accessed from the error hook thread.
Multiplicity	1..1
Type	BOOLEAN

Default value	false

Parameter Name	**MkMemoryRegionProtHookAccess**
Description	Flag determining whether the memory region can be accessed from the protection hook thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionShutdownHookAccess**
Description	Flag determining whether the memory region can be accessed from the shutdown hook thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionShutdownAccess**
Description	Flag determining whether the memory region can be accessed from the shutdown thread.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionKernelAccess**
Description	Flag determining whether the memory region can be accessed from within the mi-crokernel.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Parameter Name	**MkMemoryRegionInitializePerCore**
Description	Flag determining whether this memory region is initialized by each core. This might be useful, if there's a core-local view of this region, so each core addresses different memory by accessing the same address in this region.
Multiplicity	1..1
Type	BOOLEAN

Default value	false

7.1.3.10. MkOptimization

Parameters included	
Parameter name	**Multiplicity**
MkFastInterruptLocking	1..1

Parameter Name	**MkFastInterruptLocking**
Description	**MkFastInterruptLocking** enables a special interrupt locking mechanism in the QM-OS part of the Safety Os. It will only yield gains in performance if at least all of the counters use the same IRQ priority level, which must be the highest IRQ level used in the whole configuration. If other ISRs make calls the the QM-OS kernel, too, they should also use this IRQ level. Note that optimizing interrupt locking performance using this method might lead to a slightly higher interrupt latency.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Origin	Elektrobit Automotive GmbH

7.1.3.11. MkThreadCustomization

Parameters included	
Parameter name	**Multiplicity**
MkInitThreadMode	0..1
MkIdleThreadMode	0..1
MkShutdownThreadMode	0..1
MkOsThreadMode	0..1
MkErrorHookMode	0..1
MkProtectionHookMode	0..1
MkShutdownHookMode	0..1
MkNonPreemptiveISRs	1..1

Parameter Name	**MkInitThreadMode**

Description	Processor mode in which the initial thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkIdleThreadMode**
Description	Processor mode in which the idle thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkShutdownThreadMode**
Description	Processor mode in which the shutdown thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1

Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkOsThreadMode**
Description	Processor mode in which the QM-OS thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkErrorHookMode**
Description	Processor mode in which the error hook thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkProtectionHookMode**

Description	Processor mode in which the protection hook thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkShutdownHookMode**
Description	Processor mode in which the shutdown hook thread is executed. The value is one of: ▶ USER ▶ SUPERVISOR **USER** means that the thread is run in the unpriviliged mode of the CPU. **SUPERVISOR** means that the thread is run in the priviliged mode of the CPU.
Multiplicity	0..1
Type	ENUMERATION
Default value	USER
Range	USER SUPERVISOR
Origin	Elektrobit Automotive GmbH

Parameter Name	**MkNonPreemptiveISRs**
Description	**MkNonPreemptiveISRs** raises the running priority and interrupt lock level of each ISR that is configured or implicitly generated for QM-OS to the maximum of all ISRs in its category.
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Origin	Elektrobit Automotive GmbH

7.1.3.12. OsScheduleTable

Parameters included	
Parameter name	**Multiplicity**
OsScheduleTableIsSimple	1..1

Parameter Name	**OsScheduleTableIsSimple**
Description	With this option, a schedule table becomes a *simple schedule table*. This means, that the schedule table handling is implemented in the safety-related portion of the SafetyOS. Enable this option, if **safety-relevant** functionality depends on this schedule table. If you enable `OsScheduleTableIsSimple` the following restrictions for the schedule table and its referenced counter apply: ▶ Not all hardware incrementers may be available for the referenced counter for use with the microkernel. ▶ You can only attach a single simple schedule table to the referenced counter. ▶ You can configure the referenced counter (and therefore the simple schedule table) only in ticks. ▶ The schedule table's duration (`OsScheduleTableDuration`) must be the same as the referenced counter's modulus, which is `OsCounterMaxAllowedValue` + 1. ▶ The referenced counter's `MaxAllowedValue` must be at most 2³⁰ - 1 ticks. ▶ You cannot attach alarms to the referenced counter. ▶ The schedule table does not support synchronization (`OsScheduleTblSyncStrategy` is NONE). ▶ The schedule table is repeating (`OsScheduleTableRepeating` is set). ▶ The schedule table cannot be started automatically (`OsScheduleTableAutostart` is disabled). In order to use the referenced counter, you can either increment the counter with the function `MK_UsrSstAdvanceCounter()` or you can select a hardware incrementer from the counter's `OsDriver` section.
Multiplicity	1..1
Type	BOOLEAN

Default value	false

7.1.4. Oil attribute naming translation

Since Autosar 3.0 Oil is no longer part of the SWS specification. Customers who want to use the oil import have to change the name of the file plugin_oil_unsupported.xml into plugin.xml in the plugin directory of the Os.

The following table is showing the corresponding oil name for each xdm attribute.

XDM schema name	OIL name
MkErrorHookStack	ERRORHOOKSTACK
MkFunction	FUNCTION
MkIdleFunction	IDLEFUNCTION
MkIdleStack	IDLESTACK
MkInitFunction	INITFUNCTION
MkKernStack	KERNSTACK
MkOsStack	OSSTACK
MkProtectionHookStack	PROTECTIONHOOKSTACK
MkShutdownHookStack	SHUTDOWNHOOKSTACK
MkStack	STACK
OsAlarmAccessingApplication	ACCESSING_APPLICATION
OsAlarmActivateTask	ACTIVATETASK
OsAlarmActivateTaskRef	TASK
OsAlarmAlarmTime	ALARMTIME
OsAlarmAppModeRef	APPMODE
OsAlarmAutostartType	AUTOSTARTTYPE
OsAlarmCallback	ALARMCALLBACK
OsAlarmCallbackName	ALARMCALLBACKNAME
OsAlarmCounterRef	COUNTER
OsAlarmCycleTime	CYCLETIME
OsAlarmIncrementCounter	INCREMENTCOUNTER
OsAlarmIncrementCounterRef	COUNTER
OsAlarmSetEvent	SETEVENT
OsAlarmSetEventRef	EVENT

XDM schema name	OIL name
OsAlarmSetEventTaskRef	TASK
OsAppAccessingApplication	ACCESSING_APPLICATION
OsAppAlarmRef	ALARM
OsAppCounterRef	COUNTER
OsAppErrorHook	ERRORHOOK
OsAppErrorHookStack	ERRORHOOKSTACK
OsAppIsrRef	ISR
OsAppMkPermitShutdownOS	MK_PERMIT_SHUTDOWN_OS
OsAppResourceRef	RESOURCE
OsAppScheduleTableRef	SCHEDULETABLE
OsAppShutdownHook	SHUTDOWNHOOK
OsAppShutdownHookStack	SHUTDOWNHOOKSTACK
OsAppStartupHook	STARTUPHOOK
OsAppStartupHookStack	STARTUPHOOKSTACK
OsAppTaskRef	TASK
OsApplicationCoreAssignment	COREASSIGNMENT
OsAutosarCustomization	AUTOSAR_CUSTOMIZATION
OsCC	CC
OsCallAppErrorHook	CALL_APP_ERRORHOOK
OsCallAppStartupShutdownHook	CALL_APP_STARTUP_SHUT-DOWN_HOOK
OsCallIsr	CALL_ISR
OsCoreConfig	CORE_CONFIG
OsCoreId	CORE_ID
OsCounterAccessingApplication	ACCESSING_APPLICATION
OsCounterMaxAllowedValue	MAXALLOWEDVALUE
OsCounterMinCycle	MINCYCLE
OsCounterTicksPerBase	TICKSPERBASE
OsCpuLoadIntervalDuration	CPULOAD_INTERVAL_DURATION
OsCpuLoadMeasurement	CPULOAD
OsCpuLoadNumIntervals	CPULOAD_NUM_INTERVALS
OsCpuLoadRounding	CPULOAD_ROUNDING

XDM schema name	OIL name
OsDriver	DRIVER
OsErrorHandling	ERRORHANDLING
OsErrorHook	ERRRORHOOK
OsEventMask	MASK
OsExceptionHandling	EXCEPTIONHANDLING
OsFastInterruptLocking	USE_FAST_LOCKING
OsGptChannelRef	GPTCHANNELNAME
OsHwIncrementer	HW_INCREMENTER
OsHwModule	INCREMENTER_MODULE
OsIncrementerIrqLevel	INCREMENTER_LEVEL
OsInitCoreId	INITCOREID
OsInterruptLockingChecks	INTERRUPT_LOCKING_CHECKS
OsIsrAccessingApplication	ACCESSING_APPLICATION
OsIsrAllInterruptLockBudget	ALLINTERRUPTLOCKTIME
OsIsrCategory	CATEGORY
OsIsrCountLimit	COUNTLIMIT
OsIsrExecutionBudget	EXECUTIONBUDGET
OsIsrInterruptLock	INTERRUPTLOCK
OsIsrMaxAllInterruptLockTime	MAXALLINTERRUPTLOCKTIME
OsIsrMaxOsInterruptLockTime	MAXOSINTERRUPTLOCKTIME
OsIsrOsInterruptLockBudget	OSINTERRUPTLOCKTIME
OsIsrResourceLock	RESOURCELOCK
OsIsrResourceLockBudget	RESOURCELOCKTIME
OsIsrResourceLockResourceRef	RESOURCE
OsIsrResourceRef	RESOURCE
OsIsrTimeFrame	TIMEFRAME
OsMicrocontroller	MICROCONTROLLER
OsNumberOfCores	NUMBER_OF_CORES
OsPermitSystemObjects	PERMIT_SYSTEM_OBJECTS
OsPostISRHook	POSTISRHOOK
OsPostTaskHook	POSTTASKHOOK

XDM schema name	OIL name
OsPreISRHook	PREISRHOOK
OsPreTaskHook	PRETASKHOOK
OsProtection	PROTECTION
OsProtectionHook	PROTECTIONHOOK
OsResourceAccessingApplication	ACCESSING_APPLICATION
OsResourceLinkedResourceRef	LINKEDRESOURCE
OsScalabilityClass	SCALABILITYCLASS
OsSchTblAccessingApplication	ACCESSING_APPLICATION
OsSchedule	SCHEDULE
OsScheduleTableActivateTaskRef	TASK
OsScheduleTableAppModeRef	APPMODE
OsScheduleTableAutostartType	AUTOSTARTTYPE
OsScheduleTableCounterRef	COUNTER
OsScheduleTableDuration	DURATION
OsScheduleTableEventSetting	SETEVENT
OsScheduleTableMaxLengthen	MAXADVANCE
OsScheduleTableMaxShorten	MAXRETARD
OsScheduleTableOffset	OFFSET
OsScheduleTableRepeating	REPEATING
OsScheduleTableSetEventRef	EVENT
OsScheduleTableStartValue	ABSVALUE
OsScheduleTableSyncMaxCor	MAX_CORRECTION
OsScheduleTableSyncMaxCorAsync	MAX_CORRECTION_ASYNC
OsScheduleTableSyncMaxDec	MAX_DECREASE
OsScheduleTableSyncMaxDecAsync	MAX_DECREASE_ASYNC
OsScheduleTableSyncMaxInc	MAX_INCREASE
OsScheduleTableSyncMaxIncAsync	MAX_INCREASE_ASYNC
OsScheduleTableTaskActivation	ACTIVATETASK
OsScheduleTableTimeSyncStartup	STARTUP
OsScheduleTblExpPointOffset	OFFSET
OsScheduleTblExplicitPrecision	PRECISION

XDM schema name	OIL name
OsScheduleTblSyncStrategy	SYNC_STRATEGY
OsSchedulingAlgorithm	SCHEDULING_ALGORITHM
OsSecondsPerTick	S_PER_HW_TICK
OsServiceTrace	SERVICETRACE
OsShutdownHook	SHUTDOWNHOOK
OsSourceOptimization	SOURCEOPTIMIZATION
OsStackMonitoring	STACKCHECK
OsStackOptimization	STACKOPTIMIZATION
OsStacksize	STACKSIZE
OsStartupChecks	STARTUP_CHECKS
OsStartupHook	STARTUPHOOK
OsStatus	STATUS
OsStrictServiceProtection	STRICT_SERVICE_PROTECTION
OsTaskAccessingApplication	ACCESSING_APPLICATION
OsTaskActivation	ACTIVATION
OsTaskAllInterruptLockBudget	ALLINTERRUPTLOCKTIME
OsTaskAppModeRef	APPMODE
OsTaskCallScheduler	CALLSCHEDULER
OsTaskCountLimit	COUNTLIMIT
OsTaskEventRef	EVENT
OsTaskExecutionBudget	EXECUTIONBUDGET
OsTaskInterruptLock	INTERRUPTLOCK
OsTaskInterruptLockBudget	INTERRUPTLOCKTIME
OsTaskMaxAllInterruptLockTime	MAXALLINTERRUPTLOCKTIME
OsTaskMaxOsInterruptLockTime	MAXOSINTERRUPTLOCKTIME
OsTaskOsInterruptLockBudget	OSINTERRUPTLOCKTIME
OsTaskPriority	PRIORITY
OsTaskResourceLock	RESOURCELOCK
OsTaskResourceLockBudget	RESOURCELOCKTIME
OsTaskResourceLockResourceRef	RESOURCE
OsTaskResourceRef	RESOURCE

XDM schema name	OIL name
OsTaskSchedule	SCHEDULE
OsTaskTimeFrame	TIMEFRAME
OsTaskType	TYPE
OsTimeConstant	TIMECONSTANT
OsTimeUnit	UNIT
OsTimestampTimer	TIMESTAMP_TIMER
OsTrace	TRACE
OsTracebuffer	TRACEBUFFER
OsTrappingKernel	TRAPPINGKERNEL
OsTrustedFunctionName	NAME
OsTrustedFunctionStacksize	STACKSIZE
OsUseGetServiceId	USEGETSERVICEID
OsUseLastError	USELASTERROR
OsUseParameterAccess	USEPARAMETERACCESS
OsUseResScheduler	USERESSCHEDULER
OsUserTaskReturn	USERTASKRETURN

Table 7.1. Translation table

7.2. API Reference

7.2.1. OSEK/VDX API

7.2.1.1. General Description

The OSEK API is implemented in terms of the underlying EB tresos AutoCore OS API through a personality layer that is implemented as set of macros and library functions.

In most cases, the OSEK API function `XxxYyy()` is implemented by calling the EB tresos AutoCore OS user-library function `OS_UserXxxYyy()`. Where minor differences in the API occur, these are translated either directly in the macro or indirectly using a library function called `OS_XxxYyy()`.

OSEK API data types are implemented in terms of the underlying EB tresos AutoCore OS data types using macros. In some cases the range of values returned by the underlying API is larger than the OSEK standard allows. In these cases the extended values are translated by a library function.

The interface layer therefore behaves exactly like a standard OSEK/VDX implementation. The programmer need only concern himself with the underlying EB tresos AutoCore OS API if the extended features need to be accessed, or in the unlikely event that the address of an API function needs to be taken.

The OSEK API can be obtained by including the header file `Os.h` in your programs.

7.2.1.2. Reference

The following pages describe the OSEK/VDX data types, constants and system services as implemented by EB tresos AutoCore OS.

7.2.1.2.1. OSEK Data Types

Datatype	Description

AlarmBaseType	A structure holding the characteristics of the counter associated with an alarm. The fields of the structure include the following:
	`maxallowedvalue`
	The maximum count before the counter rolls over.
	`ticksperbase`
	The number of ticks required to reach a counter-specific unit.
	`mincycle`
	The minimum number of ticks required for a cyclic alarm (extended mode only).

AlarmBaseRefType	A reference to a variable of type `AlarmBaseType`.
AlarmType	An alarm identifier.
EventMaskType	An event identifier.
EventMaskRefType	A pointer to a variable of type `EventMaskType`.
ResourceType	A resource identifier.

Datatype	Description
StatusType	The status returned by the system calls. The status can either be `E_OK` if the service was executed successfully, or one of the error codes listed below.
TaskType	A task identifier.
TaskRefType	A pointer to a variable of type `TaskType`.
TaskStateType	A task state descriptor. The possible values are listed below in the section `CONSTANTS`.
TaskStateRefType	A pointer to a variable of type `TaskStateType`.
TickType	A counter value in ticks.
TickRefType	A pointer to a variable of type `TickType`.

Table 7.2. OSEK Data Types

7.2.1.2.2. OSEK Constants

Task states (type `TaskStateType`).

Task state	Description
`RUNNING`	Task is in the running state
`WAITING`	Task is in the waiting state
`READY`	Task is in the ready state.
`SUSPENDED`	Task is in the suspended state.

Table 7.3. OSEK Constants

Alarm base values for the system counter.

Alarm value	Description
`OSMAXALLOWEDVALUE`	The maximum tick count before the counter rolls over.
`OSTICKSPERBASE`	The number of system counter ticks required to reach a specific unit
`OSMINCYCLE`	The minimum number of ticks required for a cyclic alarm.

Alarm value	Description
`OSTICKDURATION`	The duration of a system counter tick in nanoseconds.

Table 7.4. Alarm base values for the system counter

Alarm base values of other counters, where `x` is the name of the counter.

Alarm value	Description
`OSMAXALLOWEDVALUE_x`	The maximum tick count before counter x rolls over.
`OSTICKSPERBASE_x`	The number of ticks required to reach a specific unit.
`OSMINCYCLE_x`	The minimum allowed number of ticks required for a cyclic alarm of counter x.

Table 7.5. Alarm base values of other counters, where `x` is the name of the counter

Other Constants:

Constant	Description
`RES_SCHEDULER`	(ResourceType) The scheduler resource
`INVALID_TASK`	(TaskType) The ID of an invalid task
`OSDEFAULTAPPMODE`	(AppModeType) The default application mode

Table 7.6. Other Constants

`Error Codes`

When an error occurs and debugging is enabled (*extended status* mode), system services can return the following error codes:

Eror code	Value
`E_OS_ACCESS`	1
`E_OS_CALLEVEL`	2
`E_OS_ID`	3
`E_OS_LIMIT`	4
`E_OS_NOFUNC`	5
`E_OS_RESOURCE`	6
`E_OS_STATE`	7

Eror code	Value
E_OS_VALUE	8

Table 7.7. Error Codes

7.2.1.2.3. API Functions

7.2.1.2.3.1. ALARMCALLBACK_XXXXXXXXXX()

NAME	**ALARMCALLBACK_XXXXXXXXXX**
SYNOPSIS	Define an alarm Callback function
SYNTAX	`ALARMCALLBACK(alarmcallbackname) { /* place your code here */ }`
DESCRIPTION	The `ALARMCALLBACK` macro defines a function to implement the alarm callback whose OIL name is give in the *alarmcallbackname* parameter. The code you wish to execute when the alarm expires is placed in the body of the function. The alarm callback function is executed in the context of the kernel, so it may be necessary to increase the size of the kernel stack to ensure that a stack overflow does not occur. Increasing the stack size of an ISR or adding a dummy ISR will normally achieve this.
AVAILABILITY	The `ALARMCALLBACK` macro can only be used at the outer level of a C source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.2. ActivateTask()

NAME	**ActivateTask**
SYNOPSIS	Activate a task.
SYNTAX	`StatusType ActivateTask ( TaskType TaskID /* Id of the task to be activated */ )`

NAME	ActivateTask
DESCRIPTION	`ActivateTask` activates a task. If the specified task is currently in *suspended* state, its new state after activation will be *ready*. Otherwise, the activation will be recorded and performed after task termination, if the maximum number of activations has not been reached.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The task does not exist.
E_OS_LIMIT	The task has reached its activation limit.
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.3. CancelAlarm()

NAME	CancelAlarm
SYNOPSIS	Cancel an alarm that is currently running.
SYNTAX	`StatusType CancelAlarm ( AlarmType AlarmID /* Id of the alarm */ )`
DESCRIPTION	`CancelAlarm` removes the specified alarm from its counter's alarm list. The alarm must have been previously started wit `SetRelAlarm()`, `SetAbsAlarm()` or by autostart.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The alarm does not exist (*extended* mode only)
E_OS_NOFUNC	The alarm is not active
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.4. ChainTask()

NAME	**ChainTask**
SYNOPSIS	Terminate the current task and activate another.
SYNTAX	`StatusType ChainTask ( TaskType TaskID /* Id of the task to be activated */ )`
DESCRIPTION	`ChainTask` causes the termination of the calling task and activates the task specified by the *TaskID* parameter. The task to be acivated can be the same as calling task. In this case, the chaining does cannot result in the maximum number of activations being exceeded. This means that a task with only 1 activation can chain itself. The calling task must release all resources before calling `ChainTask()`.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The chained task does not exist
E_OS_RESOURCE	The calling task still occupies a resource
E_OS_LIMIT	The chained task has reached its activation limit
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.5. ClearEvent()

NAME	**ClearEvent**
SYNOPSIS	Clear one or more events from the calling task's pending events.
SYNTAX	`StatusType ClearEvent ( EventMaskType Mask /* event mask */ )`
DESCRIPTION	`ClearEvent` clears the events specified in the *Mask* parameter from the calling task's pending events. Multiple events can be combined using the bitwise-OR ('`|`') operator.

NAME	**ClearEvent**
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table". `ClearEvent` can only be called from *extended* tasks.
RETURNS:	
E_OK	Success
E_OS_ACCESS	The calling task is not an extended task
CONFORMANCE	ECC1,ECC2

7.2.1.2.3.6. DeclareAlarm()

NAME	**DeclareAlarm**
SYNOPSIS	Declare an alarm
SYNTAX	`DeclareAlarm(AlarmName)`
DESCRIPTION	`DeclareAlarm` is a macro that is used to declare the specified alarm.
AVAILABILITY	The macro can be used wherever an *extern* declaration can be used. The best place is at the external level of the source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.7. DeclareEvent()

NAME	**DeclareEvent**
SYNOPSIS	Declare an event
SYNTAX	`DeclareEvent(EventName)`
DESCRIPTION	`DeclareEvent` is a macro that is used to declare the specified event.

NAME	**DeclareEvent**
AVAILABILITY	The macro can be used wherever an *extern* declaration can be used. The best place is at the external level of the source file.
RETURNS:	-
CONFORMANCE	ECC1,ECC2

7.2.1.2.3.8. DeclareResource()

NAME	**DeclareResource**
SYNOPSIS	Declare a resource
SYNTAX	`DeclareResource(ResourceName)`
DESCRIPTION	`DeclareResource` is a macro that is used to declare the specified resource.
AVAILABILITY	The macro can be used wherever an *extern* declaration can be used. The best place is at the external level of the source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.9. DeclareTask()

NAME	**DeclareTask**
SYNOPSIS	Declare a task
SYNTAX	`DeclareTask(TaskName)`
DESCRIPTION	`DeclareTask` is a macro that is used to declare the specified task.
AVAILABILITY	The macro can be used wherever an *extern* declaration can be used. The best place is at the external level of the source file.
RETURNS:	-

NAME	DeclareTask
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.10. DisableAllInterrupts()

NAME	DisableAllInterrupts
SYNOPSIS	Disable all category 1 and 2 interrupts
SYNTAX	`void DisableAllInterrupts(void)`
DESCRIPTION	`DisableAllInterrupts` disables all category 1 and 2 interrupts. How this is achieved depends on the architecture, but it is not guaranteed that interrupts that are unknown to the kernel (not declared in the OIL file) will be disabled. `DisableAllInterrupts()` can be nested inside `SuspendOSInterrupts()/ResumeOSInterrupts()` pairs, but not inside `SuspendAllInterrupts()/ResumeAllInterrupts()` or further `DisableAllInterrupts()/EnableAllInterrupts()` pairs. Moreover, `DisableAllInterrupts()` prevents the caller from being preempted by another task. To achieve this `DisableAllInterrupts()` may defer cross-core kernel communication.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.11. EnableAllInterrupts()

NAME	EnableAllInterrupts
SYNOPSIS	Re-enable interrupts that were disabled.
SYNTAX	`void EnableAllInterrupts(void)`

NAME	EnableAllInterrupts
DESCRIPTION	EnableAllInterrupts restores the interrupt locking to the state that is was in before the most recent call to DisableAllInterrupts. DisableAllInterrupts must have been called previously in the execution thread.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.12. ErrorHook()

NAME	ErrorHook
SYNOPSIS	A hook function to obtain error information.
SYNTAX	`void ErrorHook ( StatusType Error /* the error code */ )`
DESCRIPTION	If so configured, the kernel calls the user-supplied ErrorHook function whenever an error occurs. This typically happens when a system service would return a status code other than E_OK, but system services that do not return a status code can also cause the ErrorHook to be called. In addition, the ErrorHook can be called when the kernel detects an internal error.
AVAILABILITY	The ErrorHook function is called in the context of the kernel with category 2 interrupts disabled.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.13. GetActiveApplicationMode()

NAME	GetActiveApplicationMode
SYNOPSIS	Return the application-mode with whicht he kernel was started.

NAME	GetActiveApplicationMode
SYNTAX	`AppModeType GetActiveApplicationMode(void)`
DESCRIPTION	`GetActiveApplicationMode` returns the application mode that was passed to `StartOS` when the kernel was started.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
OS_NULLAPP-MODE	The service was called with interrupts disabled or from the wrong context.
Otherwise	The current application mode.
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.14. GetAlarm()

NAME	GetAlarm
SYNOPSIS	Get the time remaining before an alarm expires.
SYNTAX	`StatusType GetAlarm ( AlarmType AlarmID, /* Id of the alarm */ TickRefType Tick /* Where to put the result */ )`
DESCRIPTION	`GetAlarm` calculates how many ticks of the counter remain before the specified alarm expires. The result is placed in the `TickType` variable referenced by the *Tick* parameter and the service returns `E_OK`. If the return value is an error code, the referenced variable is not overwritten. If `GetAlarm` is called from an ISR, it is possible that the alarm is about to expire in a lower-priority ISR. In this case `GetAlarm` places zero in the *Tick* and returns E_OK.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success

NAME	**GetAlarm**
E_OS_ID	The alarm does not exist
E_OS_NOFUNC	The alarm is not in use
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.15. GetAlarmBase()

NAME	**GetAlarmBase**
SYNOPSIS	Get information about the alarm's counter.
SYNTAX	`StatusType GetAlarmBase ( AlarmType AlarmID, /* Id of the alarm */ AlarmBaseRefType Info /* Where to put the result */ )`
DESCRIPTION	`GetAlarmBase` places the MAXALLOWEDVALUE, TICKSPERBASE and MINCYCLE attributes of the counter to which the alarm is attached into the `AlarmBaseType` structure referenced by the `Info` parameter and returns E_OK. If an error code is returned the referenced structure is not overwritten.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The alarm does not exist
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.16. GetEvent()

NAME	**GetEvent**
SYNOPSIS	Get the pending events for a task.

NAME	GetEvent
SYNTAX	`StatusType GetEvent ( TaskType TaskID, /* Id of the task */ EventMaskRefType Event /* Where to put the result */ )`
DESCRIPTION	`GetEvent` places the mask of pending events for the specified extended task into the `EventMaskType` variable referenced by the *Event* parameter and returns `E_OK`. If an error code is returned the referenced variable is not overwritten.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The task does not exist
E_OS_ACCESS	The task is not an extended task
E_OS_STATE	The task is not currently active
CONFORMANCE	ECC1,ECC2

7.2.1.2.3.17. GetResource()

NAME	GetResource
SYNOPSIS	Enter a critical code section by acquiring a resource.
SYNTAX	`StatusType GetResource ( ResourceType ResID /* Id of the resource */ )`
DESCRIPTION	`GetResource` permits the caller to enter a critical section associated wth the specifed resource. As long as the resource is occupied no other task can successfully acquire the same resource. The resource is released when the acquiring task calls `ReleaseResource`. Resources that are associated with ISRs will also cause the associated ISR to be blocked. This may result in other ISRs being blocked too. The exact behaviour is architecture-dependent.

NAME	**GetResource**
	When multiple resources are acquired they must be released in reverse order. A task that occupies a resource must not call `TerminateTask()`, `ChainTask()` or `WaitEvent()`.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The resource does not exist
E_OS_ACCESS	The resource is in use
E_OS_ACCESS	The resource's ceiling priority is lower than the task's priority
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.18. GetTaskID()

NAME	**GetTaskID**
SYNOPSIS	Get the id of the running task.
SYNTAX	`StatusType GetTaskID ( TaskRefType TaskID /* Where to put the result */ )`
DESCRIPTION	`GetTaskID` places the identifier of the current task into the `TaskType` variable referenced by the *TaskID* parameter and returns `E_OK`. If no task is currently running, `INVALID_TASK` is placed in the variable. If an error code is returned, the referenced variable is not overwritten.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success

NAME	**GetTaskID**
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.19. GetTaskState()

NAME	**GetTaskState**
SYNOPSIS	Get the current state of a task.
SYNTAX	`StatusType GetTaskState ( TaskType TaskID, /* Id of the task */ TaskStateRefType State /* Where to put the result */ )`
DESCRIPTION	`GetTaskState` places the current state of the specified task into the `TaskStateType` variable referenced by the *State* parameter and returns `E_OK`. If an error code is returned, the referenced variable is not overwritten. WARNING: the task's state is a snapshot taken during the execution of `GetTaskState`. By the time the caller evaluates the result it might no longer be correct.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The task does not exist
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.20. ISR()

NAME	**ISR**
SYNOPSIS	Define a category 2 ISR function

NAME	**ISR**
SYNTAX	`ISR(isrname) { /* place your code here */ }`
DESCRIPTION	The `ISR` macro defines a function to implement the body of the category 2 ISR whose OIL name is give in the *isrname* parameter. The code you wish to execute when the ISR runs is placed in the body of the function.
AVAILABILITY	The `ISR` macro can only be used at the outer level of a C source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.21. OSErrorGetServiceId()

NAME	**OSErrorGetServiceId**
SYNOPSIS	Get the identifier of the system service that detected an error.
SYNTAX	`OSServiceIdType OSErrorGetServiceId(void)`
DESCRIPTION	`OSErrorGetServiceId` returns the identifier of the system service that caused the `ErrorHook` function to be called. The possible return values are `OSServiceId_xx`, where `xx` is the name of a system service.
AVAILABILITY	`OSErrorGetServiceId` only returns valid information when called from an `ErrorHook` function (including the Autosar application-specific error hooks and protection hook). The return value is undefined if `OSError-GetServiceId` is called from elsewhere. If an optimized kernel is build from the source files, `OSErrorGetSer-viceId` is only available when the `USEGETSERVICEID` attribute of the `OS` object is set to `TRUE`.
RETURNS:	
ServiceId	when called from an error hook
Undefined	when called from elsewhere

NAME	**OSErrorGetServiceId**
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.22. OSError_x1_x2()

NAME	**OSError_x1_x2**
SYNOPSIS	Get the value of a parameter to a service.
SYNTAX	`OSError_x1_x2(void)`
DESCRIPTION	`OSError_x1_x2` is a collection of macros that return the parameters passed to the system service that caused the `ErrorHook` to be called. `x1` is the name of the system service and `x2` is the name of the parameter. The return type of the macro is the same as the type of the parameter.
AVAILABILITY	`OSError_x1_x2` only returns valid information when called from an `ErrorHook` function (including the Autosar application-specific error hooks and protection hook). The return value is undefined if `OSError_x1_x2` is called from elsewhere. If an optimized kernel is build from the source files, `OSError_x1_x2` is only available when the `USEPARAMETERACCESS` attribute of the `OS` object is set to `TRUE`.
RETURNS:	
Parameter value	when called from an error hook
Undefined	when called from elsewhere
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.23. PostTaskHook()

NAME	**PostTaskHook**
SYNOPSIS	A hook routine for notifying task termination.

NAME	**PostTaskHook**
SYNTAX	`void PostTaskHook(void)`
DESCRIPTION	If so configured, the kernel calls the user-supplied `PostTaskHook` function whenever a task leaves the `RUNNING` state. This happens when `TerminateTask` or `ChainTask` is called, when `WaitEvent` is called and results in a transfer to the waiting state or when a task is pre-empted by a higher-priority task. When called from the `PostTaskHook` function, `GetTaskID` returns the ID of the outgoing task and `GetTaskState` for the outgoing task returns `RUNNING`.
AVAILABILITY	The `PostTaskHook` function is called in the context of the kernel with category 2 interrupts disabled.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.24. PreTaskHook()

NAME	**PreTaskHook**
SYNOPSIS	A hook routine for notifying task start
SYNTAX	`void PreTaskHook(void)`
DESCRIPTION	If so configured, the kernel calls the user-supplied `PreTaskHook` function whenever a task enters the `RUNNING` state. This happens when the task first starts, when it returns from `WaitEvent` after having been in the `WAITING` state and when it regains the CPU after having been pre-empted. When called from the `PreTaskHook` function, `GetTaskID` returns the ID of the incoming task and `GetTaskState` for the incoming task returns `RUNNING`.
AVAILABILITY	The `PreTaskHook` function is called in the context of the kernel with category 2 interrupts disabled.
RETURNS:	-

NAME	**PreTaskHook**
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.25. ReleaseResource()

NAME	**ReleaseResource**
SYNOPSIS	Leave a critical code section by freeing a resource.
SYNTAX	`StatusType ReleaseResource ( ResourceType ResID /* Id of the resource */ )`
DESCRIPTION	`ReleaseResource` allows the calling task or ISR to leave a critical section associated with the specified resource. It is the counterpart of `GetResource()`. Each call to `GetResource()` must be matched by a correctly-nested call to `ReleaseResource()`.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The resource does not exist
E_OS_NOFUNC	The resource is not occupied by the task, or another resource must be released first
E_OS_ACCESS	This OSEK-specified return value cannot occur because E_OS_NOFUNC takes precendence.
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.26. ResumeAllInterrupts()

NAME	**ResumeAllInterrupts**
SYNOPSIS	Enable interrupts that were disabled by `SuspendAllInterrupts`

NAME	ResumeAllInterrupts
SYNTAX	`void ResumeAllInterrupts(void)`
DESCRIPTION	`ResumeAllInterrupts` restores the interrupt lock status to the state it was in before the corresponding `SuspendAllInterrupts()` service was called.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.27. ResumeOSInterrupts()

NAME	ResumeOSInterrupts
SYNOPSIS	Enable interrupts that were disabled by `SuspendOS`
SYNTAX	`void ResumeOSInterrupts(void)`
DESCRIPTION	`ResumeOSInterrupts` restores the interrupt lock status to the state it was in before the corresponding `SuspendOSInterrupts()` service was called.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.28. Schedule()

NAME	Schedule
SYNOPSIS	Voluntarily yield the CPU
SYNTAX	`StatusType Schedule(void)`

NAME	Schedule
DESCRIPTION	`Schedule` causes the calling task to yield the CPU in favour of a higher-priority task that is in the `READY` state. The service has no effect on pre-emptive tasks.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.29. SetAbsAlarm()

NAME	SetAbsAlarm
SYNOPSIS	Set an alarm to expire at an absolute counter value
SYNTAX	`StatusType SetAbsAlarm ( AlarmType AlarmID, /* Id of the alarm */ TickType start, /* Absolute counter value in ticks */ TickType cycle /* Cycle value */ )`
DESCRIPTION	`SetAbsAlarm` sets the alarm to expire when its associated counter reaches the value specified in the *start* parameter. When the counter reaches that value, the action associated with the alarm (activate a task, set an event etc) will take place. If the *cycle* parameter is non-zero, the alarm will be reset on expiry to occur again after a further *cycle* ticks of the counter have occurred. This will be repeated indefinitely unless `CancelAlarm` is called. **NOTE** The alarm could have already expired before `SetAbsAlarm` returns to the caller.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	

NAME	**SetAbsAlarm**
E_OK	Success
E_OS_ID	The alarm does not exist
E_OS_STATE	The alarm is in use
E_OS_VALUE	The *start* parameter is greater than the MAXALLOWEDVALUE of the counter
E_OS_VALUE	The *cycle* parameter is non-zero and is less than the MINCYCLE of the counter
E_OS_VALUE	The *cycle* parameter is greater than the MAXALLOWEDVALUE of the counter
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.30. SetEvent()

NAME	**SetEvent**
SYNOPSIS	Set one or more events for a task
SYNTAX	`StatusType SetEvent ( TaskType TaskID, /* The task for whom to set the event */ EventMaskType Mask /* The event or events to set */ )`
DESCRIPTION	`SetEvent` sets the events specified by *Mask* in the pending events of the task specified by *TaskID*. If the task is in the `WAITING` state and one or more of the events for which it is waiting is now pending, it enters the `READY` state. Multiple events can be combined by using the bitwise-OR ('`|`') operator.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The task does not exist
E_OS_ACCESS	The task is not an extended task
E_OS_STATE	The task is not active

NAME	SetEvent
CONFORMANCE	ECC1,ECC2

7.2.1.2.3.31. SetRelAlarm()

NAME	SetRelAlarm
SYNOPSIS	Set an alarm to expire at a relative counter value
SYNTAX	`StatusType SetRelAlarm ( AlarmType AlarmID, /* Id of the alarm */ TickType increment, /* Absolute counter value in ticks */ TickType cycle /* Cycle value */ )`
DESCRIPTION	`SetRelAlarm` sets the alarm to expire when its associated counter reaches its current value plus the value specified in the *increment* parameter. When the counter reaches that value, the action associated with the alarm (activate a task, set an event etc) will take place. If the *cycle* parameter is non-zero, the alarm will be reset on expiry to occur again after a further *cycle* ticks of the counter have occurred. This will be repeated indefinitely unless `CancelAlarm` is called. **NOTE** The alarm could have already expired before `SetRelAlarm` returns to the caller.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The alarm does not exist
E_OS_STATE	The alarm is in use
E_OS_VALUE	The *increment* parameter is greater than the MAXALLOWEDVALUE of the counter
E_OS_VALUE	The *cycle* parameter is non-zero and is less than the MINCYCLE of the counter

NAME	SetRelAlarm
E_OS_VALUE	The *cycle* parameter is greater than the MAXALLOWEDVALUE of the counter
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.32. ShutdownHook()

NAME	ShutdownHook
SYNOPSIS	A hook routine for notifying system shut-down
SYNTAX	`void ShutdownHook ( StatusType Error /* The error that caused the shutdown */ )`
DESCRIPTION	If so configured, the kernel calls the user-supplied `ShutdownHook` function when the system shuts down.
AVAILABILITY	The `ShutdownHook` function is called in the context of the kernel with all interrupts disabled.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.33. ShutdownOS()

NAME	ShutdownOS
SYNOPSIS	Shut down the operating system
SYNTAX	`void ShutdownOS ( StatusType Error /* The error that causes the shutdown */ )`
DESCRIPTION	`ShutdownOS` shuts down the operating system. All interrupts are disabled, and the `ShutdownHook` is called if configured. Finally a non-terminating loop is entered. If it is necessary to shut-down and restart the system, the user-supplied `ShutdownHook` should arrange for the CPU to be reset.

NAME	**ShutdownOS**
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.34. StartOS()

NAME	**StartOS**
SYNOPSIS	Start the operating system in a specific mode.
SYNTAX	`void StartOS ( AppModeType Mode /* application mode */ )`
DESCRIPTION	`StartOS` starts the operating system in the specified application mode. The tasks, alarms etc. that are configured to be started automatically in that mode are started.
AVAILABILITY	`StartOS` can only be called from outside the OS, for example in the user-supplied `main()` function.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.35. StartupHook()

NAME	**StartupHook**
SYNOPSIS	A hook routine for notifying system start
SYNTAX	`void StartupHook(void)`
DESCRIPTION	If so configured, the kernel calls the user-supplied `StartupHook` function when the system starts. It is called after all internal structures etc. have been initialised, but before the scheduler starts running. The `StartupHook` function can be used to initialise hardware that cannot be initialised before calling `StartOS`.

NAME	**StartupHook**
	WARNING On some architectures it is necessary to perform some hardware initialisation before calling `StartOS`. Please refer to the Architecture Notes for your CPU.
AVAILABILITY	The `StartupHook` function is called in the context of the kernel with category 2 interrupts disabled.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.36. SuspendAllInterrupts()

NAME	**SuspendAllInterrupts**
SYNOPSIS	Disables all interrupts and saves the previous state
SYNTAX	`void SuspendAllInterrupts(void)`
DESCRIPTION	`SuspendAllInterrupts` disables all category 1 and 2 interrupts and saves the previous state. Nested calls to this system service are permitted. Moreover, `SuspendAllInterrupts()` prevents the caller from being preempted by another task. To achieve this `SuspendAllInterrupts()` may defer cross-core kernel communication.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.37. SuspendOSInterrupts()

NAME	**SuspendOSInterrupts**
SYNOPSIS	Disables category 2 interrupts and saves the previous state
SYNTAX	`void SuspendOSInterrupts(void)`

NAME	SuspendOSInterrupts
DESCRIPTION	`SuspendOSInterrupts` disables category 2 interrupts and saves the previous state. Nested calls to this system service are permitted. Moreover, `SuspendOSInterrupts()` prevents the caller from being preempted by another task. To achieve this `SuspendOSInterrupts()` may defer cross-core kernel communication.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.38. TASK()

NAME	TASK
SYNOPSIS	Define a Task function
SYNTAX	`TASK(taskname) { /* place your code here */ TerminateTask(); /* or ChainTask() */ }`
DESCRIPTION	The `TASK` macro defines a function to implement the body of the task whose OIL name is give in the *taskname* parameter. The code you wish to execute when the task runs is placed in the body of the function.
AVAILABILITY	The `TASK` macro can only be used at the outer level of a C source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.39. TerminateTask()

NAME	TerminateTask
SYNOPSIS	Terminate the calling task
SYNTAX	`StatusType TerminateTask(void)`

NAME	**TerminateTask**
DESCRIPTION	`TerminateTask` causes the termination of the calling task. The task moves from the `RUNNING` state to the `SUSPENDED` state. If successful, `TerminateTask` does not return to the caller. **NOTE** Returning from the TASK function without calling either `TerminateTask` or `ChainTask` is an error. **WARNING** All resources occupied by the task must be released before calling `TerminateTask`.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OS_RESOURCE	The calling task still occupies a resource
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.1.2.3.40. WaitEvent()

NAME	**WaitEvent**
SYNOPSIS	Wait for one or more events
SYNTAX	`StatusType WaitEvent ( EventMaskType Mask /* Events for which to wait */ )`
DESCRIPTION	`WaitEvent` causes the calling task to enter the `WAITING` state until one or more of the events specified in the *Mask* parameter becomes set. If one or more of the events was already set, `WaitEvent` returns immediately and does not enter the `WAITING` state. A task in the `WAITING` state moves to the `READY` state and becomes eligible to run when one or more of the events for which it is waiting gets set.

NAME	WaitEvent
	When the task resumes execution, it does so on the statement after the call to `WaitEvent`.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ACCESS	The task is not an extended task
E_OS_RESOURCE	The task occupies a resource
E_OS_VALUE	No events were specified (*Mask* was zero)
CONFORMANCE	ECC1,ECC2

7.2.2. AUTOSAR API

7.2.2.1. General Description

The Autosar-OS API is implemented in terms of the underlying EB tresos AutoCore OS API through a personality layer that is implemented as set of macros and library functions.

In most cases, the Autosar-OS API function `XxxYyy()` is implemented by calling the EB tresos AutoCore OS user-library function `OS_UserXxxYyy()`. Where minor differences in the API occur, these are translated either directly in the macro or indirectly using a library function called `OS_XxxYyy()`.

Autosar-OS API data types are implemented in terms of the underlying EB tresos AutoCore OS data types using macros. In some cases the range of values returned by the underlying API is larger than the Autosar specification allows. In these cases the extended values are translated by a library function.

The interface layer therefore behaves exactly like a standard Autosar-OS implementation. The programmer need only concern himself with the underlying EB tresos AutoCore OS API if the extended features need to be accessed, or in the unlikely event that the address of an API function needs to be taken.

The Autosar-OS API can be obtained by including the header file `Os.h` in your programs.

7.2.2.2. Reference

The following pages describe the Autosar-OS data types, constants and system services as implemented by EB tresos AutoCore OS.

7.2.2.2.1. Autosar Data Types

Datatype	Description
ApplicationType	An application identifier.
TrustedFunctionIndexType	A trusted function identifier.
TrustedFunctionParameterRefType	A pointer to trusted function parameters.
AccessType	Holds information about how a memory region can be accessed.
ObjectAccessType	Holds information about whether an object can be accessed.
ObjectTypeType	An object-type identifier.
MemoryStartAddressType	A pointer to any location in memory.
MemorySizeType	A scalar type that can hold the size of a memory region.
ISRType	An ISR identifer.
RestartType	Specifies whether the application should be restarted.
ScheduleTableType	A schedule table identifier.
ScheduleTableStatusType	The status of a schedule table.
ScheduleTableStatusRefType	A pointer to a ScheduleTableStatusType.
CounterType	A counter identifier.
PhysicalTimeType	A scalar type that can hold a physical time in ns, us, ms or seconds.
UnitType	Specifies in what units a physical time is measured.
GlobalTimeTickType	A value of a global time source.
ProtectionReturnType	The return value of `ProtectionHook()`.

Table 7.8. Autosar Data Types

7.2.2.2.2. Autosar Constants

Object access (type `ObjectAccessType`).

Identifier	Description
`ACCESS`	The object can be accessed.
`NO_ACCESS`	Access to an object is denied.

Table 7.9. Autosar Constants

Object types (type `ObjectType`).

Identifier	Description
`OBJECT_TASK`	The object is a task.
`OBJECT_ISR`	The object is an ISR.
`OBJECT_ALARM`	The object is an alarm.
`OBJECT_RESOURCE`	The object is a resource.
`OBJECT_COUNTER`	The object is a counter.
`OBJECT_SCHEDULETABLE`	The object is a schedule table.

Table 7.10. Object types (type `ObjectType`)

Parameter to TerminateApplication() (type `RestartType`).

Identifier	Description
`RESTART`	The application should be restarted.
`NO_RESTART`	The application must not be restarted.

Table 7.11. Parameter to TerminateApplication() (type `RestartType`)

Schedule table status (type `ScheduleTableStatusType`).

Identifier	Description
`SCHEDULETABLE_NOT_STARTED`	The schedule table is not running.
`SCHEDULETABLE_RUNNING`	The schedule table is running but is not currently synchronized with global time.
`SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS`	The schedule table is running and is synchronized with global time.
`SCHEDULETABLE_NEXT`	The schedule table is waiting for the end of a running schedule table that has attached it.
`SCHEDULETABLE_WAITING`	The schedule table is waiting for global time.

Table 7.12. Schedule table status (type `ScheduleTableStatusType`)

Unit types for physical times.

Identifier	Description
`UNIT_NS`	The time is measured in nanoseconds.
`UNIT_US`	The time is measured in microseconds.
`UNIT_MS`	The time is measured in milliseconds.
`UNIT_SEC`	The time is measured in seconds.

Table 7.13. Unit types for physical times

Return values from `ProtectionHook()` (type `ProtectionReturnType`).

Identifier	Description
`PRO_KILLTASKISR`	The offending task or ISR is to be killed.
`PRO_KILLAPPL`	The offending application is to be killed.
`PRO_KILLAPPL_RESTART`	The offending application is to be killed and then restarted.
`PRO_SHUTDOWN`	The entire system is to be shut down.

Table 7.14. Return values from `ProtectionHook()` (type `ProtectionReturnType`)

Other Constants:

Identifier	Description
`INVALID_ISR`	(ISRType) The ID of an invalid ISR
`INVALID_OSAPPLICATION`	(ApplicationType) The ID of an invalid application

Table 7.15. Other Constants

`Error Codes`

When an error occurs and debugging is enabled (*extended status* mode), system services can return the following error codes:

Identifier
`E_OS_SERVICEID`
`E_OS_RATE`
`E_OS_ILLEGAL_ADDRESS`
`E_OS_MISSINGEND`
`E_OS_DISABLEDINT`

Identifier
`E_OS_STACKFAULT`
`E_OS_PROTECTION_MEMORY`
`E_OS_PROTECTION_TIME`
`E_OS_PROTECTION_LOCKED`
`E_OS_PROTECTION_EXCEPTION`

Table 7.16. Error Codes

7.2.2.2.3. API Functions

7.2.2.2.3.1. CallTrustedFunction()

NAME	**`CallTrustedFunction`**
SYNOPSIS	Call a trusted function
SYNTAX	`StatusType CallTrustedFunction ( TrustedFunctionIndexType FunctionIndex, /* ID of trusted function */ TrustedFunctionParameterRefType FunctionParams /* Parameter for trusted function */ )`
DESCRIPTION	`CallTrustedFunction()` calls the referenced function with protection levels set as though it had been called from a trusted application. The name passed to the API macro in the first parameter is the name used to identify the function in the configuration file. The prefix "TRUSTED_-" has to be added to the first parameter to form the name of the C function implementing the trusted function. If the function identifer is held in a variable, the underlying EB tresos AutoCore OS API `OS_UserCallTrustedFunction()` should be used in place of the Autosar API.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_SERVICEID	No function defined for this index

NAME	**CallTrustedFunction**
CONFORMANCE	SC3,SC4

7.2.2.2.3.2. CheckISRMemoryAccess()

NAME	**CheckISRMemoryAccess**
SYNOPSIS	Return an ISR's memory access rights
SYNTAX	`AccessType CheckISRMemoryAccess ( ISRType isrid, MemoryStartAddressType address, MemorySizeType size )`
DESCRIPTION	`CheckISRMemoryAccess()` returns the access rights that the referenced ISR has over the specified memory region. The specified region must lie entirely within a single memory block (private data, application data, stack etc.) otherwise the function returns no access rights. If an error occurs, the function returns no access rights. The macros `OSMEMORY_IS_READABLE()`, `OSMEMORY_IS_WRITEABLE()`, `OSMEMORY_IS_EXECUTABLE()` and `OSMEMORY_IS_STACKSPACE()` can be used to examine the return value.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
AccessType	The permitted access
CONFORMANCE	SC3,SC4

7.2.2.2.3.3. CheckObjectAccess()

NAME	**CheckObjectAccess**
SYNOPSIS	Return an applications's access rights for an object
SYNTAX	`ObjectAccessType CheckObjectAccess ( ApplicationType appid, ObjectTypeType objecttype, objectid )`

NAME	**CheckObjectAccess**
DESCRIPTION	`CheckObjectAccess()` returns a value that indicates whether the referenced application has permission to access the referenced object. "Access" in this sense means using the object as a parameter to a system service. The application to which an object belongs automatically gets permission to access that object. Otherwise permission must be explicitly granted using the ACCESSING_APPLICATION attribute in the OIL file. If an error occurs, the return value is NO_ACCESS
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
ACCESS	The application is permitted access
NO_ACCESS	The application is not permitted access
CONFORMANCE	SC3,SC4

7.2.2.2.3.4. CheckObjectOwnership()

NAME	**CheckObjectOwnership**
SYNOPSIS	Returns an object's owner application
SYNTAX	`ApplicationType CheckObjectOwnership ( ObjectTypeType objecttype, objectid )`
DESCRIPTION	`CheckObjectOwnership()` returns the application to which the referenced object belongs. If the object has no owner or an error occurs, INVALID_OSAPPLICATION is returned instead.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
INVALID_OSAPPLICATION	The application could not be determined
Otherwise	The ID of the application that owns the object

NAME	CheckObjectOwnership
CONFORMANCE	SC3,SC4

7.2.2.2.3.5. CheckTaskMemoryAccess()

NAME	CheckTaskMemoryAccess
SYNOPSIS	Return a task's memory access rights
SYNTAX	`AccessType CheckTaskMemoryAccess ( TaskType taskid, MemoryStartAddressType address, MemorySizeType size )`
DESCRIPTION	`CheckTaskMemoryAccess()` returns the access rights that the referenced task has over the specified memory region. The specified region must lie entirely within a single memory block (private data, application data, stack etc.) otherwise the function returns no access rights. If an error occurs, the function returns no access rights. The macros `OSMEMORY_IS_READABLE()`, `OSMEMORY_IS_WRITEABLE()`, `OSMEMORY_IS_EXECUTABLE()` and `OSMEMORY_IS_STACKSPACE()` can be used to examine the return value.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
AccessType	The permitted access
CONFORMANCE	SC3,SC4

7.2.2.2.3.6. ErrorHook_[App]()

NAME	ErrorHook_[App]
SYNOPSIS	a application specific hook routine for error situations
SYNTAX	`void ErrorHook_[App] ( StatusType Error /* the error code */ )`

NAME	ErrorHook_[App]
DESCRIPTION	When an error occurs AND an application-specific ErrorHook is configured for the faulty OS-Application, the operating System shall call that application-specific error hook ErrorHook_[App] after the system specific ErrorHook is called (if configured).
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.2.2.3.7. GetApplicationID()

NAME	GetApplicationID
SYNOPSIS	Identify the current application
SYNTAX	`ApplicationType GetApplicationID(void)`
DESCRIPTION	`GetApplicationID` returns the identifier of the currently-running application. If the application cannot be ascertained or an error occurs, INVALID_OSAPPLICATION is returned instead.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
INVALID_OSAPPLICATION	No application is running
Otherwise	The ID of the current application
CONFORMANCE	SC3,SC4

7.2.2.2.3.8. GetCounterValue()

NAME	GetCounterValue
SYNOPSIS	Get the current value of the counter

NAME	GetCounterValue
SYNTAX	`StatusType GetCounterValue ( CounterType CounterID, TickRefType Value )`
DESCRIPTION	`GetCounterValue()` places the current value of the specified counter in the designated `Value` variable. If the counter does not exist or another error is detected, the `Value` variable remains unchanged. If this system service is called from an ISR of higher priority than the counter's own ISR, the count value might occasionally be slightly less than expected, but this will reflect the state of the alarms in the counter's queue.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OS_OK	Success
E_OS_ID	The CounterID is not valid

7.2.2.2.3.9. GetElapsedCounterValue()

NAME	GetElapsedCounterValue
SYNOPSIS	Get the number of elapsed ticks
SYNTAX	`StatusType GetElapsedCounterValue ( CounterType CounterID, TickType PreviousValue, TickRefType Value )`
DESCRIPTION	`GetElapsedCounterValue()` places the number of ticks of the specified counter that have elapsed since the counter had the value `last` in the designated `out` variable. If the counter does not exist or another error is detected, the `out` variable remains unchanged. If this system service is called from an ISR of higher priority than the counter's own ISR, there might be expired alarms still in the queue that have not been processed. CAVEAT: there is no way to calculate the number of elapsed ticks and get a new counter value simultaneously. This is as specified by Autosar.

NAME	**GetElapsedCounterValue**
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OS_OK	Success
E_OS_ID	CounterID is invalid
E_OS_VALUE	PreviousValue is larger that the MAXALLOWEDVALUE

7.2.2.2.3.10. GetISRID()

NAME	**GetISRID**
SYNOPSIS	Identify the current ISR
SYNTAX	`ISRType GetISRID(void)`
DESCRIPTION	`GetISRID()` identifies the current ISR. If no ISR is running or an error occurs, INVALID_ISR is returned instead.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
INVALID_ISR	No ISR currently running
Otherwise	Identifier of current ISR
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.11. GetScheduleTableStatus()

NAME	**GetScheduleTableStatus**
SYNOPSIS	Returns the current status of a schedule table
SYNTAX	`StatusType GetScheduleTableStatus ( ScheduleTableType st, ScheduleTableStatusRefType out )`

NAME	**GetScheduleTableStatus**
DESCRIPTION	`GetScheduleTableStatus()` places the current status table of the referenced schedule table in the ScheduleTableStatusType variable pointed to by "out" and returns E_OK. If an error occurs, an error code is returned and the "out" variable is not modified.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.12. IncrementCounter()

NAME	**IncrementCounter**
SYNOPSIS	Increment a software counter
SYNTAX	`StatusType IncrementCounter ( CounterType counter )`
DESCRIPTION	`IncrementCounter()` increments a software counter. Any alarms or schedule-table action points that become due as a result will be processed before `IncrementCounter()` returns. If the processing results in a higher priority task becoming active, the kernel will reshedule.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The counter does not exist
E_OS_ID	The counter is a hardware counter
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.13. NextScheduleTable()

NAME	**NextScheduleTable**
SYNOPSIS	Start a schedule table at the end of another
SYNTAX	`StatusType NextScheduleTable ( ScheduleTableType ScheduleTableID_From, ScheduleTableType ScheduleTableID_To )`
DESCRIPTION	`NextScheduleTable()` chains ScheduleTableID_To to ScheduleTableID_From so that when ScheduleTableID_From comes to the end of its list of actions, ScheduleTableID_To will replace it. The timing is arranged so that the first action point of ScheduleTableID_To occurs at its specified offset from the full end of ScheduleTableID_From's period. Please read the notes and caveats given for `OS_UserChainScheduleTable()` to understand the limitations of this system service.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	One or both of the schedule tables does not exist
E_OS_NOFUNC	The "current" schedule table is not running
E_OS_STATE	The "next" schedule table is already running or chained
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.14. OSMEMORY_IS_EXECUTABLE()

NAME	**OSMEMORY_IS_EXECUTABLE**
SYNOPSIS	Test access rights for execute permission.
SYNTAX	`int OSMEMORY_IS_EXECUTABLE(AccessType a)`
DESCRIPTION	`OSMEMORY_IS_EXECUTABLE()` returns a non-zero value ("true") if the parameter indicates that execute permission is granted.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.

NAME	OSMEMORY_IS_EXECUTABLE
RETURNS:	
Zero	The memory cannot be executed
Nonzero	The memory can be executed
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.15. OSMEMORY_IS_READABLE()

NAME	OSMEMORY_IS_READABLE
SYNOPSIS	Test access rights for read permission.
SYNTAX	`int OSMEMORY_IS_READABLE(AccessType a)`
DESCRIPTION	`OSMEMORY_IS_READABLE()` returns a non-zero value ("true") if the parameter indicates that read permission is granted.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
Zero	The memory cannot be read
Nonzero	The memory can be read
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.16. OSMEMORY_IS_STACKSPACE()

NAME	OSMEMORY_IS_STACKSPACE
SYNOPSIS	Test access rights for stack space indication.
SYNTAX	`int OSMEMORY_IS_STACKSPACE(AccessType a)`
DESCRIPTION	`OSMEMORY_IS_STACKSPACE()` returns a non-zero value ("true") if the parameter indicates that the memory is in the stack space.

NAME	**OSMEMORY_IS_STACKSPACE**
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
Zero	The memory is not in the stack
Nonzero	The memory is in the stack
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.17. OSMEMORY_IS_WRITEABLE()

NAME	**OSMEMORY_IS_WRITEABLE**
SYNOPSIS	Test access rights for write permission.
SYNTAX	`int OSMEMORY_IS_WRITEABLE(AccessType a)`
DESCRIPTION	`OSMEMORY_IS_WRITEABLE()` returns a non-zero value ("true") if the parameter indicates that write permission is granted.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
Zero	The memory cannot be written
Nonzero	The memory can be written
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.18. ProtectionHook()

NAME	**ProtectionHook**
SYNOPSIS	a hook routine for serious error situations
SYNTAX	`ProtectionReturnType ProtectionHook ( StatusType Fatalerror )`

NAME	**ProtectionHook**
DESCRIPTION	The protection hook is always called if a serious error occurs. E.g. exceeding the worst case execution time or violating against the memory protection. Depending on the return value the OS will either kill the task category 2 ISR which causes the problem, kill the OS-Application the task category 2 ISR belong (optional with restart) or shutdown the system.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
PRO_KILLTASKISR	Kills the task or category 2 ISR which causes the problem.
PRO_KILLAPPL	Kills the application (all application belonging objects).
PRO_KILLAPPL_RESTART	Kills the application which causes the problem and restarts it (using the restart task).
PRO_SHUTDOWN	Shutdown the OS.
CONFORMANCE	SC2,SC3,SC4

7.2.2.2.3.19. SetScheduleTableAsync()

NAME	**SetScheduleTableAsync**
SYNOPSIS	Sets a schedule table's state to "asynchronous"
SYNTAX	`StatusType SetScheduleTableAsync ( ScheduleTableType st )`
DESCRIPTION	`SetScheduleTableAsync()` sets a schedule table to the "asynchronous" state. The schedule table will remain asynchronous indefinitely and will continue to run governed only by local time. Any remaining synchronization steps from a previous invocation of `SyncScheduleTable()` will be dropped. A subsequent call to `SyncScheduleTable()` can resynchronize the schedule table. `SetScheduleTableAsync()` is intended to inform the kernel that contact with the global time provider has been lost.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.

NAME	**SetScheduleTableAsync**
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
CONFORMANCE	SC2,SC4

7.2.2.2.3.20. ShutdownHook_[App]()

NAME	**ShutdownHook_[App]**
SYNOPSIS	a application specific hook for the shutdown
SYNTAX	`void ShutdownHook_[App] ( StatusType Error /* the error that caused the shutdown */ )`
DESCRIPTION	This application-specific hook is called by the kernel with the access rights of the associated OS-Application on on shutdown of the OS and before the system-specific ShutdownHook.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.2.2.3.21. StartScheduleTableAbs()

NAME	**StartScheduleTableAbs**
SYNOPSIS	Start a schedule table with an absolute offset
SYNTAX	`StatusType StartScheduleTableAbs ( ScheduleTableType scheduletableid TickType offset )`
DESCRIPTION	`StartScheduleTableAbs()` starts the proccessing of the referenced schedule table at its first expiry point after the underlaying counter reaches the specified offset.

NAME	StartScheduleTableAbs
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
E_OS_VALUE	Offet is greater than MAXALLOWEDVALUE of the counter
E_OS_STATE	The schedule table is already running
E_OS_STATE	The counter is running another schedule table
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.22. StartScheduleTableRel()

NAME	StartScheduleTableRel
SYNOPSIS	Start a schedule table with a relativ offset
SYNTAX	`StatusType StartScheduleTableRel ( ScheduleTableType scheduletableid TickType offset )`
DESCRIPTION	`StartScheduleTableRel()` starts the proccessing of the referenced schedule table at its first expiry point after the specified offset ticks have elapsed.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
E_OS_VALUE	Offet is greater than MAXALLOWEDVALUE of the counter
E_OS_STATE	The schedule table is already running
E_OS_STATE	The counter is running another schedule table
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.23. StartScheduleTableSynchron()

NAME	**StartScheduleTableSynchron**
SYNOPSIS	Start a schedule table synchronously
SYNTAX	`StatusType StartScheduleTableSynchron ( ScheduleTableType ScheduleTableID, GlobalTimeTickType GlobalTime )`
DESCRIPTION	`StartScheduleTableSynchron()` places a schedule table into the WAITING state so that it will start synchronously when global time becomes available. The GlobalTime parameter is not used in the synchronization calculation.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	
E_OS_OK	Success
E_OS_ID	The schedule table does not exist
E_OS_ID	The schedule table cannot be synchronised
E_OS_VALUE	The GlobalTime parameter is invalid
E_OS_STATE	The schedule table has already been started
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.24. StartupHook_[App]()

NAME	**StartupHook_[App]**
SYNOPSIS	a application specific hook routine for system startup
SYNTAX	`void StartupHook_[App](void)`
DESCRIPTION	This application-specific hook is called by the kernel with the access rights of the associated OS-Application on startup of the OS but after the system-specific StartupHook.
AVAILABILITY	Refer to Figure 7.1, "Allowed calling context for OS service calls table".
RETURNS:	-

NAME	StartupHook_[App]
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.2.2.3.25. StopScheduleTable()

NAME	StopScheduleTable
SYNOPSIS	Stop a schedule table
SYNTAX	StatusType StopScheduleTable (ScheduleTableType sched-uletableid)
DESCRIPTION	StopScheduleTable() stops the schedule table immediately. If another schedule table has been chained to this one, it will not be started.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
E_OS_NOFUNC	The schedule table was not running
CONFORMANCE	SC1,SC2,SC3,SC4

7.2.2.2.3.26. SyncScheduleTable()

NAME	SyncScheduleTable
SYNOPSIS	synchronize a schedule table with global time
SYNTAX	StatusType SyncScheduleTable (ScheduleTableType st GlobalTimeTickType global)
DESCRIPTION	SyncScheduleTable() sets the synchronization mechanism of the schedule table "st" such that it will synchronize itself with global time, tak-ing account of the variation between the local and global time values given. Synchronization takes place either after every expiry point or at the end of each full processing round of the schedule table depending on the SYNC_-

NAME	**SyncScheduleTable**
	STRATEGY attribute. Synchronisation is achieved by either shortening or lengthening successive time intervals by the configured maximum increase or decrease steps. If the schedule table is synchronous (within the specificed PRECISION attribute) the maximum step sizes are determined by the MAX_INCREASE and MAX_DECREASE attributes. If the schedule table is not synchronous the the maximum step sizes are determined by the MAX_INCREASE_ASYNC and MAX_DECREASE_ASYNC attributes and the adjustment will be in whichever direction will achieve syncronisation in the least number of steps. `SyncScheduleTable()` is the sole method of determining whether a schedule table is synchronous or not. If the remaining adjustment falls below the precision, the schedule table remains in the asynchronous state until a further call to SyncScheduleTable() confirms that it is synchronous.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_ID	The schedule table does not exist
E_OS_ID	The schedule table cannot be synchronized
CONFORMANCE	SC2,SC4

7.2.2.2.3.27. TerminateApplication()

NAME	**TerminateApplication**
SYNOPSIS	Terminate an entire application
SYNTAX	`StatusType TerminateApplication ( RestartType RestartOption )`
DESCRIPTION	`TerminateApplication()` terminates the calling application. All tasks are terminated, all interrupts are disabled and pending interrupts cleared, all counters, alarms and schedule tables are stopped and all resources are freed for the assigned application object. If the `RestartOption` parameter is `RESTART`, the application is restarted by activating its restart task if it

NAME	**TerminateApplication**
	has one. If the `RestartOption` parameter is `NO_RESTART`, the application remains terminated and cannot be restarted. A successful call to `TerminateApplication()` does not return.
AVAILABILITY	Refer to Figure 7.1, “Allowed calling context for OS service calls table”.
RETURNS:	
E_OK	Success
E_OS_CALLLEVEL	Called from incorrect context
CONFORMANCE	SC3,SC4

7.2.3. EB tresos AutoCore OS API Extensions

7.2.3.1. General Description

The ProOSEK compatibilty API provides some extensions to the OSEK/VDX API. The API is compatible with the ProOSEK operating system version 4.

The ProOSEK compatibilty API is defined in terms of the underlying EB tresos AutoCore OS API using translation macros. Therefore addresses of the API cannot be taken and used in function pointer constructs. If such a construct is needed the underlying EB tresos AutoCore OS API should be used instead.

7.2.3.2. Reference

The following pages describe the data types, constants and system services offered by EB tresos AutoCore OS in addition to the standard OSEK/VDX API.

7.2.3.2.1. API Functions

7.2.3.2.1.1. AdvanceCounter()

NAME	**AdvanceCounter**
SYNOPSIS	Increment the given counter

NAME	**AdvanceCounter**
SYNTAX	`StatusType AdvanceCounter ( CounterName /* name of a counter */ )`
DESCRIPTION	`AdvanceCounter` increments the given counter by one. Any alarms that expire as a result of this will cause the appropriate alarm action to take place. If the action is an alarm callback, the callback function runs in the context of the caller of `AdvanceCounter()`.
AVAILABILITY	This service is called only from the task level and not from the interrupt level. For incrementing counters within an interrupt, see `IAdvanceCounter()`. In AUTOSAR OS, `AdvanceCounter()` and `IAdvanceCounter()` are identical, but failure to observe the above distinction may result in non-portable code.
RETURNS:	
E_OK	Success
E_OS_ID	The alarm ID is invalid
CONFORMANCE	BCC1,BCC2,ECC1,BCC2

7.2.3.2.1.2. IAdvanceCounter()

NAME	**IAdvanceCounter**
SYNOPSIS	Increment the given counter at interrupt level
SYNTAX	`StatusType IAdvanceCounter ( CounterName /* name of a counter */ )`
DESCRIPTION	`IAdvanceCounter` increments the given counter by one. Any alarms that expire as a result of this will cause the appropriate alarm action to take place. If the action is an alarm callback, the callback function runs in the context of the caller of `IAdvanceCounter()`.
AVAILABILITY	This service is called only from interrupt level and not from task level. For incrementing counters within a task, see `AdvanceCounter()`.

NAME	IAdvanceCounter
	In AUTOSAR OS, `AdvanceCounter()` and `IAdvanceCounter()` are identical, but failure to observe the above distinction may result in non-portable code.
RETURNS:	
E_OK	Success
E_OS_ID	The alarm ID is invalid
CONFORMANCE	BCC1,BCC2,ECC1,BCC2

7.2.3.2.1.3. ISR1()

NAME	ISR1
SYNOPSIS	Define a category 1 ISR function
SYNTAX	`ISR1(isrname) { /* place your code here */ }`
DESCRIPTION	The `ISR1` macro defines a function to implement the body of the category 1 ISR whose OIL name is give in the *isrname* parameter. The code you wish to execute when the ISR runs is placed in the body of the function.
AVAILABILITY	The `ISR1` macro can only be used at the outer level of a C source file.
RETURNS:	-
CONFORMANCE	BCC1,BCC2,ECC1,ECC2

7.2.3.2.1.4. getUnusedIsrStack()

NAME	getUnusedIsrStack
SYNOPSIS	Get the amount of interrupt stack that remains unused
SYNTAX	`os_size_t getUnusedIsrStack(void)`

NAME	**getUnusedIsrStack**
DESCRIPTION	`getUnusedIsrStack` returns the amount of interrupt stack that has not been overwritten. At startup, all stacks are filled with a fill pattern. The amount of interrupt stack that still contains the fill pattern in counted.
AVAILABILITY	`getUnusedIsrStack` can be used from tasks and ISRs.
RETURNS:	
s	The number of bytes of stack that remain unused.

7.2.3.2.1.5. getUnusedTaskStack()

NAME	**getUnusedTaskStack**
SYNOPSIS	Get the amount of stack the task has not used
SYNTAX	`os_size_t getUnusedTaskStack(TaskID)`
DESCRIPTION	`getUnusedTaskStack` returns the amount of stack that has not been overwritten by the given task. At startup, all stacks are filled with a fill pattern. The amount of stack that still contains the fill pattern in counted. If two or more tasks are sharing the same stack, it is not known which of the tasks has written to the stack. For this function to return 100% reliable values, the stack-sharing feature in the Generator should be turned off.
AVAILABILITY	`getUnusedTaskStack` can be used from tasks and ISRs.
RETURNS:	
s	The number of bytes of stack that remain unused.

7.2.3.2.1.6. getUsedIsrStack()

NAME	**getUsedIsrStack**
SYNOPSIS	Get the amount of interrupt stack that has been used

NAME	getUsedIsrStack
SYNTAX	`os_size_t getUsedIsrStack(void)`
DESCRIPTION	`getUsedIsrStack` returns the amount of interrupt stack that has been overwritten. At startup, all stacks are filled with a fill pattern. The amount of interrupt stack that still contains the fill pattern in counted and subtracted from the total amount.
AVAILABILITY	`getUsedIsrStack` can be used from tasks and ISRs.
RETURNS:	
s	The number of bytes of stack used.

7.2.3.2.1.7. getUsedTaskStack()

NAME	getUsedTaskStack
SYNOPSIS	Get the amount of stack the task has used
SYNTAX	`os_size_t getUsedTaskStack(TaskID)`
DESCRIPTION	`getUsedTaskStack` returns the amount of stack that has been overwritten by the given task. At startup, all stacks are filled with a fill pattern. The amount of stack that still contains the fill pattern in counted and subtracted from the total amount. If two or more tasks are sharing the same stack, it is not known which of the tasks has written to the stack. For this function to return 100% reliable values, the stack-sharing feature in the Generator should be turned off.
AVAILABILITY	`getUsedTaskStack` can be used from tasks and ISRs.
RETURNS:	
s	The number of bytes of stack used.

7.2.3.2.1.8. stackCheck()

NAME	**stackCheck**
SYNOPSIS	Check current stack use
SYNTAX	`int stackCheck(void)`
DESCRIPTION	`stackCheck` checks the stack use in the current context. If there is or has been a stack overflow, `stackCheck` returns +1. If there is a stack underflow, `stackCheck` returns -1. Otherwise `stackCheck` returns 0.
AVAILABILITY	Can be used from all tasks and ISRs.
RETURNS:	
0	OK
+1	Stack overflow
-1	Stack underflow

7.2.4. EB tresos AutoCore OS User API

7.2.4.1. General Description

The EB tresos AutoCore OS API is the underlying API of the EB tresos AutoCore OS kernel. All other APIs are written in terms of this API using, in the main, translation macros.

The functions in the API are all true functions with C calling semantics. Their addresses can be taken and used in function pointer constructs. Many of the functions --- those with names of the form `OS_UserXxx` --- are direct interfaces to the system calls in the kernel.

7.2.4.2. Reference

7.2.4.2.1. Error information structure

The error information structure is filled with detailed information about the error by the EB tresos AutoCore OS error handler. The information in the structure is valid during the error- and protection-hook functions. Outside these functions its content is not defined.

The function `OS_GetErrorInfo()` returns the error information structure for the core on which it is called.

The error information structure contains the following fields:

- `calledFrom` indicates the context in which the error occurred.

  The possible values are defined in `Os_kernel_task.h` and listed in the following table:

Value	Identifier	Description
0	`OS_INBOOT`	Error occurred while the system was starting up
1	`OS_INTASK`	Error occurred while executing a task
2	`OS_INCAT1`	Error occurred while executing a Cat-1 ISR
3	`OS_INCAT2`	Error occurred while executing a Cat-2 ISR
4	`OS_INACB`	Error occurred while executing an alarm callback
5	`OS_INSHUTDOWN`	Error occurred while the system was shutting down
6	`OS_ININTERNAL`	Error occurred while executing an internal kernel function
7	`OS_INSTARTUPHOOK`	Error occurred while executing a startup hook
8	`OS_INSHUTDOWNHOOK`	Error occurred while executing a shutdown hook
9	`OS_INERRORHOOK`	Error occurred while executing an error hook
10	`OS_INPRETASKHOOK`	Error occurred while executing a pre-task hook
11	`OS_INPOSTTASKHOOK`	Error occurred while executing a post-task hook
12	`OS_INPREISRHOOK`	Error occurred while executing a pre-ISR hook
13	`OS_INPOSTISRHOOK`	Error occurred while executing a post-ISR hook
14	`OS_INPROTECTIONHOOK`	Error occurred while executing a protection hook

Table 7.17. Possible values for `calledFrom`

- `serviceId` indicates the system service in which the error was detected. This is one of the `OS_SID_-xxx` constants defined in `OS_error.h`.
- `parameter` is an array of three parameters that contain useful information related to the point of failure, their content varies depending on the type of error and the used hardware. The following list of common errors provide parameters in an unified way, please refer to the *Architecture Notes* for more details on hardware specific error handling. If an error handling routine does not use all three parameters, the unused ones contain undefined values.
  - When the error is detected in a system service, `parameter[i]` contains the i'th parameter passed to the service, numbered 0,1,2 from left to right.
  - When the source of the error is an unconfigured interrupt being triggered, `parameter[0]` will contain the unconfigured interrupt's vector number or arbitration priority.

- errorCondition indicates the exact error condition. Its value is one of the OS_ERROR_xxx constants defined in Os_error.h.
- action indicates the action that will be taken when the hook function returns. If an error-hook function modifies the content of action, the new action will be taken instead of the default. Use this with caution! Note: modifying action in the protection hook has no effect because the return value of the protection hook determines the action.

  Out of the values defined in Os_error.h the following may be used for action:

Value	Identifier	Description
0	OS_ACTION_IGNORE	Ignore the error and return OS_E_OK
1	OS_ACTION_RETURN	Only return result to caller
2	OS_ACTION_KILL	Kill the task or ISR that caused the error
3	OS_ACTION_QUARANTINE	Quarantine the task or ISR that caused the error
4	OS_ACTION_QUARANTINEAPP	Quarantine the application that caused the error
5	OS_ACTION_RESTART	Kill and restart the application that caused the error
6	OS_ACTION_SHUTDOWN	Shut down the OS

Table 7.18. Possible values for action

- result contains the same value as the error code that was passed as a parameter to the error- or protection-hook, its value will be returned to the caller if such an action is chosen and the affected system service returns a status code. If result is modified by an error-hook function the new value will be returned instead of the default. Modifying result in the protection- hook has no effect.

7.2.4.2.2. API Functions

7.2.4.2.2.1. GetCpuLoad()

NAME	GetCpuLoad
SYNOPSIS	Return the current CPU load as an integer percentage.
SYNTAX	os_uint8_t GetCpuLoad (void)
DESCRIPTION	GetCpuLoad() returns the current CPU load. The return value is a percentage in the range 0 to 100. The measurement is taken on the core on which the function is called.

NAME	**GetCpuLoad**
	If CPU load measurement is disabled, no action takes place and the function returns 255 (0xff).
AVAILABILITY	Include the `Os_salsa.h` header file. This function must not be called from non-trusted applications; instead, `OS_UserGetCpuLoad()` can be used.
RETURNS:	
0..100	CPU load as percentage
255	Measurement is not enabled

7.2.4.2.2.2. GetCpuLoadOnCore()

NAME	**GetCpuLoadOnCore**
SYNOPSIS	Return the current CPU load as an integer percentage.
SYNTAX	`os_uint8_t GetCpuLoadOnCore ( os_coreid_t coreId /* Selects the core on which the measurement shall be taken. */ )`
DESCRIPTION	`GetCpuLoadOnCore()` returns the current CPU load. The return value is a percentage in the range 0 to 100. The measurement is taken on the core `coreId`. If CPU load measurement is disabled, no action takes place and the function returns 255 (0xff).
AVAILABILITY	Include the `Os_salsa.h` header file. This function must not be called from non-trusted applications; instead, `OS_UserGetCpuLoad()` can be used.
RETURNS:	
0..100	CPU load as percentage
255	Measurement is not enabled

7.2.4.2.2.3. GetMaxCpuLoad()

NAME	GetMaxCpuLoad
SYNOPSIS	Return the peak CPU load as an integer percentage.
SYNTAX	`os_uint8_t GetMaxCpuLoad (void)`
DESCRIPTION	`GetMaxCpuLoad()` returns the peak CPU load. The return value is a percentage in the range 0 to 100. The measurement is taken on the core on which the function is called. If CPU load measurement is disabled, no action takes place and the function returns 255 (0xff).
AVAILABILITY	Include the `Os_salsa.h` header file. This function must not be called from non-trusted applications; instead, `OS_UserGetCpuLoad()` can be used.
RETURNS:	
0..100	CPU load as percentage
255	Measurement is not enabled

7.2.4.2.2.4. GetMaxCpuLoadOnCore()

NAME	GetMaxCpuLoadOnCore
SYNOPSIS	Return the peak CPU load as an integer percentage.
SYNTAX	`os_uint8_t GetMaxCpuLoadOnCore ( os_coreid_t coreId /* Selects the core on which the measurement shall be taken. */ )`
DESCRIPTION	`GetMaxCpuLoadOnCore()` returns the peak CPU load. The return value is a percentage in the range 0 to 100. The measurement is taken on the core `coreId`. If CPU load measurement is disabled, no action takes place and the function returns 255 (0xff).

NAME	GetMaxCpuLoadOnCore
AVAILABILITY	Include the Os_salsa.h header file. This function must not be called from non-trusted applications; instead, OS_UserGetCpuLoad() can be used.
RETURNS:	
0..100	CPU load as percentage
255	Measurement is not enabled

7.2.4.2.2.5. InitCpuLoad()

NAME	InitCpuLoad
SYNOPSIS	Reset the CPU load monitor's peak load detector.
SYNTAX	`void InitCpuLoad(void)`
DESCRIPTION	InitCpuLoad() resets the peak CPU load detector of the load monitoring system. The peak load latch is set to the current load, after first ensuring that the current load is up-to-date. If CPU load measurement is disabled, no action takes place.
AVAILABILITY	Include the Os_salsa.h header file. This function must not be called from non-trusted applications; instead, OS_UserResetPeakCpuLoad() can be used.
RETURNS:	-

7.2.4.2.2.6. InitCpuLoadOnCore()

NAME	InitCpuLoadOnCore
SYNOPSIS	Reset the CPU load monitor's peak load detector.
SYNTAX	`void InitCpuLoadOncore ( os_coreid_t coreId /* Selects the core of which the peak load shall be initialized. */ )`

NAME	InitCpuLoadOnCore
DESCRIPTION	InitCpuLoad() resets the peak CPU load detector of the load monitoring system. The peak load latch is set to the current load, after first ensuring that the current load is up-to-date. This is done for the specified core. If CPU load measurement is disabled, no action takes place.
AVAILABILITY	Include the Os_salsa.h header file. This function must not be called from non-trusted applications; instead, OS_UserResetPeakCpuLoad() can be used.
RETURNS:	-

7.2.4.2.2.7. OS_DiffTime32()

NAME	OS_DiffTime32
SYNOPSIS	Calculates the 32-bit length of an interval between two times
SYNTAX	os_uint32_t OS_DiffTime32 (const os_timestamp_t *newTime, /* new timestamp value */ const os_timestamp_t *oldTime /* old timestamp value */)
DESCRIPTION	OS_DiffTime32() calculates the difference (newTime - oldTime) (i.e. the duration of the interval that starts at oldTime and ends at newTime). The result is returned as 32-bit number. If the time difference is too large to be represented in 32 bits, the function returns the maximum value that can be represented (0xffffffff).
AVAILABILITY	No restrictions.
RETURNS:	-

7.2.4.2.2.8. OS_FastResumeAllInterrupts()

NAME	OS_FastResumeAllInterrupts
SYNOPSIS	Resume interrupts to a previously-saved level

NAME	OS_FastResumeAllInterrupts
SYNTAX	void OS_FastResumeAllInterrupts(void)
DESCRIPTION	OS_FastResumeAllInterrupts() restores the interrupt level of the processor or interrupt controller to the level that it was before the corresponding call to OS_FastSuspendAllInterrupts(). It is used to implement the ResumeAllInterrupts() and DisableAllInterrupts() system services. ResumeAllInterrupts() is nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls. According to the OSEK standard, EnableAllInterrupts() is not nestable, but due to the implementation here it is freely nestable with itself and with ResumeAllInterrupts(). CAVEAT: There are no tests or checks. It is up to the user to ensure that no OS services are called while interrupts are disabled and that the nesting count is not exceeded. Furthermore there is no interrupt lock timing.
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.9. OS_FastResumeOsInterrupts()

NAME	OS_FastResumeOsInterrupts
SYNOPSIS	Resume interrupts to a previously-saved level
SYNTAX	void OS_FastResumeOsInterrupts(void)
DESCRIPTION	OS_FastResumeOsInterrupts() restores the interrupt level of the processor or interrupt controller to the level that it was before the corresponding call to OS_FastSuspendOsInterrupts(). It is used to implement the ResumeOsInterrupts() system service. ResumeOsInterrupts() is nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls.

NAME	OS_FastResumeOsInterrupts
	CAVEAT: There are no tests or checks. It is up to the user to ensure that no OS services are called while interrupts are disabled and that the nesting count is not exceeded. Furthermore there is no interrupt lock timing.
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.10. OS_FastSuspendAllInterrupts()

NAME	OS_FastSuspendAllInterrupts
SYNOPSIS	Suspend interrupts up to the "all" lock level
SYNTAX	`void OS_FastSuspendAllInterrupts(void)`
DESCRIPTION	`OS_FastSuspendAllInterrupts()` raises the interrupt level of the processor or interrupt controller to a level that locks out all Cat1 and Cat2 interrupts. It is used to implement the `SuspendAllInterrupts()` and `DisableAllInterrupts()` system services. `SuspendAllInterrupts()` and `DisableAllInterrupts()` are interchangable and mutually nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls. The equivalence of the two services is a deviation from a strict interpretation of the OSEK and Autosar standards. WARNING: no error checking is implemented, therefore incorrect nesting, or calling system services between Suspend and Resume pairs, could cause unexpected behaviour. No interrupt lock timing is implemented, and a running execution or resource lock timer might be disabled for the duration of the interrupt lock.
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.11. OS_FastSuspendOsInterrupts()

NAME	**OS_FastSuspendOsInterrupts**
SYNOPSIS	Suspend interrupts up to the "all" lock level
SYNTAX	`void OS_FastSuspendOsInterrupts(void)`
DESCRIPTION	`OS_FastSuspendOsInterrupts()` raises the interrupt level of the processor or interrupt controller to a level that locks out all Cat2 interrupts. It is used to implement the `SuspendOsInterrupts()` system service. `SuspendOsInterrupts()` is nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls. WARNING: no error checking is implemented, therefore incorrect nesting, or calling system services between Suspend and Resume pairs, could cause unexpected behaviour. No interrupt lock timing is implemented, and a running execution or resource lock timer might be disabled for the duration of the interrupt lock.
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.12. OS_GetCurrentStackArea()

NAME	**OS_GetCurrentStackArea**
SYNOPSIS	Get current stack boundaries
SYNTAX	`void OS_GetCurrentStackArea(void **begin, void **end)`
DESCRIPTION	`OS_GetCurrentStackArea()` it places the base and limit addresses of the stack of the currently-executing object into the two referenced variables. For a Task, this is simply the stack area as allocated by the OS generator. For ISRs, if the ISR has a private stack, this is returned. Otherwise the entire kernel stack area is returned. This does not imply that the whole area is accessible by the caller.

NAME	**OS_GetCurrentStackArea**
AVAILABILITY	Can be used from all tasks and Category 2 ISRs.
RETURNS:	-

7.2.4.2.2.13. OS_GetErrorInfo()

NAME	**OS_GetErrorInfo**
SYNOPSIS	Get error status information
SYNTAX	`const os_errorstatus_t * OS_GetErrorInfo()`
DESCRIPTION	`OS_GetErrorInfo()` returns a pointer to the error information status structure of the current core. The information in this structure is valid during the `ErrorHook()` and `ProtectionHook()`. It will be overwritten, once the next error occurs.
AVAILABILITY	No restrictions.
RETURNS:	
	Pointer to the error information structure.

7.2.4.2.2.14. OS_GetIsrMaxRuntime()

NAME	**OS_GetIsrMaxRuntime**
SYNOPSIS	Get longest observed runtime of an ISR
SYNTAX	`os_result_t OS_GetIsrMaxRuntime ( os_isrid_t t /* ID of ISR */ os_tick_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_GetIsrMaxRuntime()` places the longest observed execution time of the specified ISR into the variable referenced by 'out'. If the ISR ID is invalid or the ISR does not have execution-time measurement enabled (attribute

NAME	**OS_GetIsrMaxRuntime**
	MEASURE_MAX_RUNTIME), OS_GetIsrMaxRuntime() returns OS_E_ID.
AVAILABILITY	Available from all trusted tasks, ISRs and hook functions. One some architectures, it might be possible to call this function from non-trusted contexts as well.
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Invalid ISR ID specified.
OS_E_ID	ISR does not have the feature enabled.

7.2.4.2.2.15. OS_GetScheduleTableStatus()

NAME	**OS_GetScheduleTableStatus**
SYNOPSIS	Get status of a schedule table
SYNTAX	`StatusType OS_GetScheduleTableStatus ( ScheduleTableType s /* ID of schedule table */ ScheduleTableStatusRefType sr /* Where to put the answer */ )`
DESCRIPTION	OS_GetScheduleTableStatus() places the current state of the specified schedule table into the 'sr' variable. If the schedule table ID is invalid, OS_GetScheduleTableStatus() returns OS_E_ID. Otherwise, the schedule table's state is translated into one of the standard AUTOSAR states and stored into the 'sr' variable.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Invalid Task ID specified.

7.2.4.2.2.16. OS_GetTaskMaxRuntime()

NAME	**OS_GetTaskMaxRuntime**
SYNOPSIS	Get longest observed runtime of a task
SYNTAX	`os_result_t OS_GetTaskMaxRuntime ( os_taskid_t t /* ID of task */ os_tick_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_GetTaskMaxRuntime()` places the longest observed execution time of the specified task into the variable referenced by 'out'. If the task ID is invalid or the task does not have execution-time measurement enabled (attribute MEASURE_MAX_RUNTIME), `OS_GetTaskMaxRuntime()` returns `OS_E_ID`.
AVAILABILITY	Available from all trusted tasks, ISRs and hook functions. One some architectures, it might be possible to call this function from non-trusted contexts as well.
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Invalid Task ID specified.
OS_E_ID	Task does not have the feature enabled.

7.2.4.2.2.17. OS_GetTaskState()

NAME	**OS_GetTaskState**
SYNOPSIS	Get state of a task
SYNTAX	`os_result_t OS_GetTaskState ( os_taskid_t t /* ID of task */ os_taskstate_t *sr /* Where to put the answer */ )`
DESCRIPTION	`OS_GetTaskState()` places the current state of the specified task into the 'sr' variable. If the task ID is invalid, `OS_GetTaskState()` returns `OS_E_ID`. Otherwise, the task's state is translated into one of the standard OSEK/VDX states and stored into the 'sr' variable.

NAME	OS_GetTaskState
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Invalid Task ID specified.
OS_E_UNKNOWN	The task's state is invalid.

7.2.4.2.2.18. OS_GetTimeStamp()

NAME	OS_GetTimeStamp
SYNOPSIS	Puts a timestamp value into the indicated location
SYNTAX	`void OS_GetTimeStamp ( os_timestamp_t out /* Destination location for timestamp value */ )`
DESCRIPTION	`OS_GetTimeStamp()` stores the current timestamp value into the indicated "out" location. A timestamp is a counter that can never overflow during the expected up-time of the processor.
AVAILABILITY	No restrictions.
RETURNS:	-

7.2.4.2.2.19. OS_GetUnusedIsrStack()

NAME	OS_GetUnusedIsrStack
SYNOPSIS	Get the amount of interrupt stack that remains unused
SYNTAX	`os_size_t OS_GetUnusedIsrStack(void)`
DESCRIPTION	`OS_GetUnusedIsrStack` returns the amount of interrupt stack that has not been overwritten. At startup, all stacks are filled with a fill pattern. The amount of interrupt stack that still contains the fill pattern is counted.

NAME	OS_GetUnusedIsrStack
AVAILABILITY	`OS_GetUnusedIsrStack` can be used from Tasks and ISRs.
RETURNS:	
	The number of bytes of stack that remain unused.

7.2.4.2.2.20. OS_GetUnusedTaskStack()

NAME	OS_GetUnusedTaskStack
SYNOPSIS	Get the amount of stack the task has not used
SYNTAX	`os_size_t OS_GetUnusedTaskStack ( os_taskid_t t /* Task ID */ )`
DESCRIPTION	`OS_GetUnusedTaskStack` returns the amount of stack that has not been overwritten by the given task. At startup, all stacks are filled with a fill pattern. The amount of stack that still contains the fill pattern is counted. If two or more tasks are sharing the same stack, it is not known which of the tasks has written to the stack. For this function to return 100% reliable values, the stack-sharing feature in the Generator should be turned off.
AVAILABILITY	`OS_GetUnusedTaskStack` can be used from Tasks and ISRs.
RETURNS:	
	The number of bytes of stack that remain unused.

7.2.4.2.2.21. OS_GetUsedIsrStack()

NAME	OS_GetUsedIsrStack
SYNOPSIS	Get the amount of interrupt stack that has been used

NAME	OS_GetUsedIsrStack
SYNTAX	`os_size_t OS_GetUsedIsrStack(void)`
DESCRIPTION	`OS_GetUsedIsrStack` returns the amount of interrupt stack that has been overwritten. At startup, all stacks are filled with a fill pattern. The amount of interrupt stack that still contains the fill pattern is counted and subtracted from the total amount.
AVAILABILITY	`OS_GetUsedIsrStack` can be used from Tasks and ISRs.
RETURNS:	
	The number of bytes of interrupt stack used.

7.2.4.2.2.22. OS_GetUsedTaskStack()

NAME	OS_GetUsedTaskStack
SYNOPSIS	Get the amount of stack the task has used
SYNTAX	`os_size_t OS_GetUsedTaskStack ( os_taskid_t t /* Task ID */ )`
DESCRIPTION	`OS_GetUsedTaskStack` returns the amount of stack that has been overwritten by the given task. At startup, all stacks are filled with a fill pattern. The amount of stack that still contains the fill pattern is counted and subtracted from the total amount. If two or more tasks are sharing the same stack, it is not known which of the tasks has written to the stack. For this function to return 100% reliable values, the stack-sharing feature in the Generator should be turned off.
AVAILABILITY	`OS_GetUsedTaskStack` can be used from Tasks and ISRs.
RETURNS:	
	The number of bytes of stack used.

7.2.4.2.2.23. OS_IsScheduleNecessary()

NAME	**OS_IsScheduleNecessary**
SYNOPSIS	Determine whether a call to Schedule() is necessary
SYNTAX	`os_boolean_t OS_IsScheduleNecessary(void)`
DESCRIPTION	`OS_IsScheduleNecessary()` returns TRUE (non-zero) if there is another task in the task queue with a higher configured priority than the current task. Otherwise it returns FALSE.
AVAILABILITY	`OS_IsScheduleNecessary()` should only be called from a task. If it is called from another context and there is a current task, it will return information about that task. If there is no current task there will be a null-pointer access, which may cause a memory access exception, or may cause false information to be returned, depending on the system hardware and configuration. `OS_IsScheduleNecessary()` can onyl be called from tasks that have read access to kernel variables. On most systems this will be true, but in SC3 and SC4 memory protection might prevent access if so configured and will detect a memory protection error in the calling task.
RETURNS:	
Zero	A call to schedule is not necessary
Non-zero	A call to schedule is necessary

7.2.4.2.2.24. OS_IsScheduleWorthwhile()

NAME	**OS_IsScheduleWorthwhile**
SYNOPSIS	Determine whether a call to Schedule() is worthwhile
SYNTAX	`os_boolean_t OS_IsScheduleWorthwhile(void)`
DESCRIPTION	`OS_IsScheduleWorthwhile()` returns TRUE (non-zero) if there is another task in the task queue (other than the current task). Otherwise it returns FALSE. `OS_IsScheduleWorthwhile()` is faster than `OS_IsScheduleNecessary()`, but can return TRUE even if `Schedule()` will have no effect.

NAME	OS_IsScheduleWorthwhile
	However, it might result in a performance improvement in some circumstances, especially when called from a background task that is of the lowest priority.
AVAILABILITY	OS_IsScheduleWorthwhile() should only be called from a task. If it is called from another context and there is a current task, it will return information about that task. If there is no current task there will be a null-pointer access, which may cause a memory access exception, or may cause false information to be returned, depending on the system hardware and configuration. OS_IsScheduleWorthwhile() can onyl be called from tasks that have read access to kernel variables. On most systems this will be true, but in SC3 and SC4 memory protection might prevent access if so configured and will detect a memory protection error in the calling task.
RETURNS:	
Zero	A call to schedule is not worthwhile
Non-zero	A call to schedule is worthwhile

7.2.4.2.2.25. OS_ScheduleIfNecessary()

NAME	OS_ScheduleIfNecessary
SYNOPSIS	Call Schedule() if necessary
SYNTAX	os_result_t OS_ScheduleIfNecessary(void)
DESCRIPTION	OS_ScheduleIfNecessary() calls OS_IsScheduleNecessary() and if it returns TRUE, calls Schedule() and returns the result. Otherwise E_OS_OK is returned.
AVAILABILITY	OS_ScheduleIfNecessary() should only be called from a task. The conditions and restrictions for OS_IsScheduleNecessary() apply here as well.
RETURNS:	
E_OS_OK	Success, or Schedule() was not called.

NAME	**OS_ScheduleIfNecessary**

7.2.4.2.2.26. OS_ScheduleIfWorthwhile()

NAME	**OS_ScheduleIfWorthwhile**
SYNOPSIS	Call Schedule() if worthwhile
SYNTAX	`os_result_t OS_ScheduleIfWorthwhile(void)`
DESCRIPTION	`OS_ScheduleIfWorthwhile()` calls `OS_IsScheduleWorthwhile()` and if it returns TRUE, calls `Schedule()` and returns the result. Otherwise E_OS_OK is returned.
AVAILABILITY	`OS_ScheduleIfWorthwhile()` should only be called from a task. The conditions and restrictions for `OS_IsScheduleWorthwhile()` apply here as well.
RETURNS:	
E_OS_OK	Success, or Schedule() was not called.

7.2.4.2.2.27. OS_SimTimerAdvance()

NAME	**OS_SimTimerAdvance**
SYNOPSIS	Advances a simulated timer by a given value
SYNTAX	`os_result_t OS_SimTimerAdvance ( os_unsigned_t tmr, /* Index of the timer */ os_tick_t incr /* Value by which to increment the timer */ )`
DESCRIPTION	`OS_SimTimerAdvance()` increments a simulated timer by the given value. It checks for each channel whether the timer is pending or passed the value programmed in its compare register. If the channel is enabled, it calls the respective associated ISR, otherwise the channel is set to pending.
AVAILABILITY	No restrictions.

NAME	OS_SimTimerAdvance
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Timer index is out of range.
OS_E_VALUE	Increment value is above the mask value of the timer.

7.2.4.2.2.28. OS_SimTimerSetup()

NAME	OS_SimTimerSetup
SYNOPSIS	Set up a simulated timer channel
SYNTAX	`os_result_t OS_SimTimerSetup ( os_unsigned_t tmr, /* Index of the timer */ os_unsigned_t chan, /* Index of the compare register */ os_isrid_t isrId /* ISR-id of the associated interrupt */ )`
DESCRIPTION	`OS_SimTimerSetup()` sets up a simulated timer channel by clearing its compare and control registers and setting its interrupts ID.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success.
OS_E_ID	Timer index is out of range.
OS_E_ID	Compare register index is out of range.
OS_E_VALUE	ISR-id is invalid.

7.2.4.2.2.29. OS_StackCheck()

NAME	OS_StackCheck
SYNOPSIS	Check current stack use

NAME	OS_StackCheck
SYNTAX	`os_size_t OS_StackCheck(void)`
DESCRIPTION	`OS_StackCheck` checks the stack use in the current context. If If there is or has been a stack overflow, `OS_StackCheck` returns +1. If there is a stack underflow, `OS_StackCheck` returns -1. Otherwise `OS_StackCheck` returns 0.
AVAILABILITY	Can be used from all tasks and ISRs.
RETURNS:	
0	OK
+1	Stack overflow
-1	Stack underflow

7.2.4.2.2.30. OS_TimeGetHi()

NAME	OS_TimeGetHi
SYNOPSIS	Returns high word of a timestamp value
SYNTAX	`os_uint32_t OS_TimeGetHi ( os_timestamp_t t /* Source timestamp value */ )`
DESCRIPTION	`OS_TimeGetHi()` returns the high word of a given timestamp value.
AVAILABILITY	No restrictions.
RETURNS:	
	High word of the timestamp value.

7.2.4.2.2.31. OS_TimeGetLo()

NAME	**OS_TimeGetLo**
SYNOPSIS	Returns low word of a timestamp value
SYNTAX	`os_uint32_t OS_TimeGetLo ( os_timestamp_t t /* Source timestamp value */ )`
DESCRIPTION	`OS_TimeGetLo()` returns the low word of a given timestamp value.
AVAILABILITY	No restrictions.
RETURNS:	
	Low word of the timestamp value.

7.2.4.2.2.32. OS_TimeSub64()

NAME	**OS_TimeSub64**
SYNOPSIS	Returns high word of a timestamp value
SYNTAX	`void OS_TimeSub64 ( os_timestamp_t *diffTime /* Destination */, const os_timestamp_t *newTime /* new time value */, const os_timestamp_t *oldTime /* old time value */ )`
DESCRIPTION	OS_TimeSub64() calculates the difference (`newTime` - `oldTime`) (i.e. the duration of the interval that starts at `oldTime` and ends at `newTime`). The two input values are variables provided by the caller whose addresses are passed as parameters. The result is placed into the variable whose address is specified by the `diffTime` parameter. The caller must have permission to modify this variable.
AVAILABILITY	No restrictions.
RETURNS:	-

7.2.4.2.2.33. OS_UserActivateTask()

NAME	**OS_UserActivateTask**
SYNOPSIS	Activate a task
SYNTAX	`os_result_t OS_UserActivateTask ( os_taskid_t t /* ID of task */ )`
DESCRIPTION	`OS_UserActivateTask()` activates a task. If the specified task is currently in the *suspended* state its new state will be *ready*. If the task is already *ready* or *running* the activation will be recorded and performed after the task terminates, if permitted.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.34. OS_UserAllowAccess()

NAME	**OS_UserAllowAccess**
SYNOPSIS	Grant access to the calling application
SYNTAX	`os_result_t OS_UserAllowAccess(void)`
DESCRIPTION	`OS_UserAllowAccess()` sets the state of application of the calling task or ISR to ACCESSIBLE, provided it is in the RESTARTING state.
AVAILABILITY	`OS_UserAllowAccess()` may only be called from a task or ISR.
RETURNS:	-

7.2.4.2.2.35. OS_UserCallTrustedFunction()

NAME	**OS_UserCallTrustedFunction**
SYNOPSIS	Call a trusted function

NAME	**OS_UserCallTrustedFunction**
SYNTAX	`os_result_t OS_UserCallTrustedFunction ( os_functionid_t fid, /* Id of function */ void *parms /* Pointer parameter to pass */ )`
DESCRIPTION	`OS_UserCallTrustedFunction()` calls the referenced trusted function with the parameter supplied, provided that the caller is in a permitted context and has permission to make the call. It is recommended to make trusted functions as short as possible, doing only those jobs such as accessing peripheral devices that can only be done with full privileges. It is not recommended to call OSEK or AUTOSAR system services from a trusted function. However, if it is absolutely necessary to use system services from a trusted function, please take careful note of the following restrictions and differences in semantic behaviour: The trusted function is called in a kernel environment, which means that all system calls that it makes will return immediately to the caller; any resulting task switch will not happen until the trusted function returns, thus affecting the calling task but not the trusted function. If the trusted function has been called from an ISR (category 2) context, the system services that it can call are restricted accordingly. Calling a system service that is not permitted will result in an error code being returned to the trusted function. In normal status mode it is possible that the calling application could have been terminated.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success
OS_E_ACCESS	Trusted function's application not accessible.

7.2.4.2.2.36. OS_UserCancelAlarm()

NAME	**OS_UserCancelAlarm**
SYNOPSIS	Cancel an alarm
SYNTAX	`os_result_t OS_UserCancelAlarm ( os_alarmid_t a /* ID of the alarm */ )`
DESCRIPTION	`OS_UserCancelAlarm()` resets the expiration time of the specified alarm.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.37. OS_UserChainScheduleTable()

NAME	**OS_UserChainScheduleTable**
SYNOPSIS	Chain a schedule table
SYNTAX	`os_result_t OS_UserChainScheduleTable ( os_scheduleid_t sc /* current table */ os_scheduleid_t sn /* next table */ )`
DESCRIPTION	`OS_UserChainScheduleTable()` chains the schedule table `sn` to start after the current round of the table `sc` ends. Chaining is only permitted if the table to be chained is stopped and if the current table is running and does not already have a chained table. The timing is arranged such that the first action point of the chained table occurs at its proper offset after the end of the period of the "current" table. If the "current" table is not periodic, the first action point takes place at its offset from the last action point of the "current" table. The Autosar specification is silent on the latter case. `CAVEAT` The chaining takes place at the last action point of the "current" table. This means that if `NextScheduleTable()` (or `OS_UserChainScheduleTable()`) is called after this (for example, in the last

NAME	OS_UserChainScheduleTable
	schedule task) the running table will process one more complete round before the chaining takes place. If the "current" table is not periodic it may already have stopped and the call to NextScheduleTable() will fail with OS_E_STATE.
AVAILABILITY	OS_UserChainScheduleTable() can be called from tasks and category 2 ISRs.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.38. OS_UserChainTask()

NAME	OS_UserChainTask
SYNOPSIS	Terminate the current task and activate another
SYNTAX	os_result_t OS_UserChainTask (os_taskid_t t /* ID of the task */)
DESCRIPTION	OS_UserChainTask() terminates the current task and activates another. The activated task may be the same as the calling task, in which case the new activation is guaranteed not to exceed the maximum number of activations for the task. The function may return of there is an error.
AVAILABILITY	OS_UserChainTask() may only be called from a task.
RETURNS:	-

7.2.4.2.2.39. OS_UserCheckIsrMemoryAccess()

NAME	OS_UserCheckIsrMemoryAccess
SYNOPSIS	Returns memory access permissions for an ISR

NAME	OS_UserCheckIsrMemoryAccess
SYNTAX	os_memoryaccess_t OS_UserCheckIsrMemoryAccess (os_isrid_t i, /* ISR ID */ void *ptr, /* Address of memory */ os_size_t len /* Length of memory */)
DESCRIPTION	OS_UserCheckIsrMemoryAccess() returns information about the ISR's access rights over the specified memory region. The return value contains a bitwise OR of the return values listed below to indicate that the memory region is readable, writeable, executable and located in the stack. If the ISR is trusted, it has read, write end execute permission over the whole of memory and the stack bit indicates that the region lies entirely within the global interrupt stack. However, this does not necessarily mean that the region can be addressed in the given manner.
AVAILABILITY	No restrictions.
RETURNS:	
OS_MA_READ	The memory is readable
OS_MA_WRITE	The memory is writeable
OS_MA_EXEC	The memory is executable
OS_MA_STACK	The memory is in the stack

7.2.4.2.2.40. OS_UserCheckObjectAccess()

NAME	OS_UserCheckObjectAccess
SYNOPSIS	Indicates whether an application has access to an object
SYNTAX	os_result_t OS_UserCheckObjectAccess (os_applicationid_t a, /* Application */ os_objecttype_t typ, /* Type of object */ os_objectid_t id, /* Object to check */)
DESCRIPTION	OS_UserCheckObjectAccess() checks if the referenced application has access permission to the specified object. The applications permission mask is checked against the permission bits of the object. The function returns true (OS_TRUE) if access is granted and false (OS_FALSE) if access is denied. If either the application or the object does not exist the error han-

NAME	OS_UserCheckObjectAccess
	dler is called and the return value is false. Since ISRs do not have a permissions field (no ACCESSING_APPLICATION in the configuration, the only application that can access an ISR is the owner. This is moot because there is no Autosar API that "accesses" the ISR.
AVAILABILITY	No restrictions.
RETURNS:	
OS_TRUE	Permission is granted
OS_FALSE	Permission is not granted

7.2.4.2.2.41. OS_UserCheckObjectOwnership()

NAME	OS_UserCheckObjectOwnership
SYNOPSIS	Returns the ID of the application that owns the object
SYNTAX	`os_result_t OS_UserCheckObjectOwnership ( os_objecttype_t typ, /* Type of object to examine */ os_objectid_t id /* Id of object to examine */ )`
DESCRIPTION	`OS_UserCheckObjectOwnership()` returns the ID of the application that owns the object specified by typ and id. Permitted object types are OS_OBJ_APPLICATION, OS_OBJ_TASK, OS_OBJ_ISR, OS_OBJ_RESOURCE, OS_OBJ_COUNTER, OS_OBJ_ALARM and OS_OBJ_SCHEDULETABLE. If no owner application can be found for any reason, the return value is OS_NULLAPP. The error handler is called if the typ parameter is an unknown or unhandled object type, or if the specified object does not exist.
AVAILABILITY	No restrictions.
RETURNS:	
ApplicationId	Success
OS_NULLAPP	No application found

7.2.4.2.2.42. OS_UserCheckTaskMemoryAccess()

NAME	**OS_UserCheckTaskMemoryAccess**
SYNOPSIS	Returns memory access permissions for a task
SYNTAX	`os_memoryaccess_t OS_KernCheckTaskMemoryAccess ( os_taskid_t t, /* Task ID */ void *ptr, /* Address of memory */ os_size_t len /* Length of memory */ )`
DESCRIPTION	`OS_UserCheckTaskMemoryAccess()` returns the access permissions (read/write/execute) for the referenced task for the specified memory region. In addition, the return value indicates whether the memory is in the task's stack. The stack is only considered to be accessible when the task is active. The return value is a logical OR of the bit fields given below.
AVAILABILITY	No restrictions.
RETURNS:	
OS_MA_READ	The memory is readable
OS_MA_WRITE	The memory is writeable
OS_MA_EXEC	The memory is executable
OS_MA_STACK	The memory is in the stack

7.2.4.2.2.43. OS_UserClearEvent()

NAME	**OS_UserClearEvent**
SYNOPSIS	Clear one or more events
SYNTAX	`os_result_t OS_UserClearEvent ( os_eventmask_t e /* Events to be cleared */ )`
DESCRIPTION	`OS_UserClearEvent()` clears all the specified events from the current task's pending events.
AVAILABILITY	`OS_UserClearEvent()` may only be called from a task.

NAME	OS_UserClearEvent
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.44. OS_UserDisableInterruptSource()

NAME	OS_UserDisableInterruptSource
SYNOPSIS	Disable the Specified Interrupt Source
SYNTAX	`os_result_t OS_UserDisableInterruptSource(os_isrid_t)`
DESCRIPTION	`OS_UserDisableInterruptSource()` disables the specified interrupt source.
AVAILABILITY	No restrictions.
RETURNS:	
E_OS_OK	success
E_OS_ID	the isr id was invalid
OS_E_ACCESS	the application is not accessible
OS_E_CORE	the core has been shut down, which is responsible for the ISR

7.2.4.2.2.45. OS_UserEnableInterruptSource()

NAME	OS_UserEnableInterruptSource
SYNOPSIS	Enable the Specified Interrupt Source
SYNTAX	`os_result_t OS_UserEnableInterruptSource(os_isrid_t)`
DESCRIPTION	`OS_UserEnableInterruptSource()` enables the specified interrupt source.
AVAILABILITY	No restrictions.

NAME	OS_UserEnableInterruptSource
RETURNS:	
E_OS_OK	success
E_OS_ID	the isr id was invalid
OS_E_ACCESS	the application is not accessible
OS_E_CORE	the core has been shut down, which is responsible for the ISR

7.2.4.2.2.46. OS_UserGetActiveApplicationMode()

NAME	OS_UserGetActiveApplicationMode
SYNOPSIS	Get the current application mode
SYNTAX	`os_appmodeid_t OS_UserGetActiveApplicationMode(void)`
DESCRIPTION	`OS_UserGetActiveApplicationMode` returns the application mode that was given to `OS_UserStartOs()` when the system started.
AVAILABILITY	
RETURNS:	
mode	Current application mode

7.2.4.2.2.47. OS_UserGetAlarm()

NAME	OS_UserGetAlarm
SYNOPSIS	Get the time remaining on the alarm
SYNTAX	`os_result_t OS_UserGetAlarm ( os_alarmid_t a, /* ID of the alarm */ os_tick_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetAlarm()` calculates the time remaining before the specified alarm expires and places the result in the designated `out` variable. If the

NAME	**OS_UserGetAlarm**
	alarm is not in use or another error is detected, the `out` variable remains unchanged.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.48. OS_UserGetAlarmBase()

NAME	**OS_UserGetAlarmBase**
SYNOPSIS	Get alarm configuration
SYNTAX	`os_result_t OS_UserGetAlarmBase ( os_alarmid_t a, /* ID of the alarm */ os_alarmbase_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetAlarmBase()` places the configured parameters `maxallowedvalue`, `mincycle` and `ticksperbase` into the specified `out` variable. If an error occurs, the `out` variable remains unchanged.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success
OS_E_ACCESS	Alarm's application is not accessible.

7.2.4.2.2.49. OS_UserGetApplicationId()

NAME	**OS_UserGetApplicationId**
SYNOPSIS	Get the current application
SYNTAX	`os_applicationid_t OS_UserGetApplicationId(void)`

NAME	OS_UserGetApplicationId
DESCRIPTION	OS_UserGetApplicationId() returns the ID of the current application. If no category 2 ISR or task is running, or if the current ISR or task does not belong to an application, OS_NULLAPP is returned instead.
AVAILABILITY	No restrictions.
RETURNS:	
AppId	ID of current application
OS_NULLAPP	No application is running

7.2.4.2.2.50. OS_UserGetApplicationState()

NAME	OS_UserGetApplicationState
SYNOPSIS	Get state of an application
SYNTAX	`os_result_t OS_UserGetApplicationState ( os_applicationid_t t, /* ID of application */ os_appstate_t *out /* Where to put the answer */ )`
DESCRIPTION	OS_UserGetApplicationState() writes the current state of the specified application to the location specified in the "out" parameter.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success
OS_E_ID	Invalid Application ID specified

7.2.4.2.2.51. OS_UserGetCounterValue()

NAME	OS_UserGetCounterValue
SYNOPSIS	Get the current value of the counter

NAME	OS_UserGetCounterValue
SYNTAX	os_result_t OS_UserGetCounterValue (os_counterid_t c, /* ID of the counter */ os_tick_t *out /* Where to put the answer */)
DESCRIPTION	OS_UserGetCounterValue() places the current value of the specified counter in the designated out variable. If the counter does not exist or another error is detected, the out variable remains unchanged. If this system service is called from an ISR of higher priority than the counter's own ISR, the count value might occasionally be slightly less than expected, but this will reflect the state of the alarms in the counter's queue.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success
OS_E_ACCESS	Counter's application is not accessible.

7.2.4.2.2.52. OS_UserGetCpuLoad()

NAME	OS_UserGetCpuLoad
SYNOPSIS	Return the CPU load as an integer percentage.
SYNTAX	os_uint8_t OS_UserGetCpuLoad (os_coreid_t coreId, /* Selects the core on which the measurement shall be taken. */ os_boolean_t getPeak /* Returns peak load if true, otherwise current load. */)
DESCRIPTION	OS_UserGetCpuLoad() returns either the current or the peak CPU load, depending on the value of the getPeak parameter. The return value is a percentage in the range 0 to 100. The core on which the measurement is taken is selected by coreId. Use the special value OS_CORE_ID_THIS_CORE to select the core on which the function is called. If CPU load measurement is disabled, no action takes place and the function returns 255 (0xff).

NAME	OS_UserGetCpuLoad
AVAILABILITY	Include the `Os_salsa.h` header file. No restrictions.
RETURNS:	
0..100	CPU load as percentage
255	Measurement is not enabled

7.2.4.2.2.53. OS_UserGetElapsedCounterValue()

NAME	OS_UserGetElapsedCounterValue
SYNOPSIS	Get the number of elapsed ticks
SYNTAX	`os_result_t OS_UserGetElapsedCounterValue ( os_counterid_t c, /* ID of the counter */ os_tick_t *last, /* The previous value of the counter */ os_tick_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetElapsedCounterValue()` places the number of ticks of the specified counter that have elapsed since the counter had the value in the designated `last` variable into the designated `out` variable. The current value of the counter is placed in the designated `last` variable. If the counter does not exist or another error is detected, the `last` and `out` variables remain unchanged. If this system service is called from an ISR of higher priority than the counter's own ISR, there might be expired alarms still in the queue that have not been processed.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.54. OS_UserGetEvent()

NAME	**OS_UserGetEvent**
SYNOPSIS	Get the pending events for a task
SYNTAX	`os_result_t OS_UserGetEvent ( os_taskid_t t, /* ID of the task */ os_eventmask_t *ep /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetEvent()` places the pending events for the specified task into the `out` variable. The task must be an extended task. If an error is detected, the `out` variable remains unchanged.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.55. OS_UserGetIsrId()

NAME	**OS_UserGetIsrId**
SYNOPSIS	Return the id of the current ISR
SYNTAX	`os_isrid_t OS_UserGetIsrId(void)`
DESCRIPTION	If `OS_UserGetIsrId()` is called from an ISR of category category 2, or from an ErrorHook or ProtectionHook caused by an ISR of category 2, it returns the ID of the ISR. Otherwise it returns OS_NULLISR. If the more relaxed (but not Autosar-conformant) calling context checks are configured, the ISR ID is also returned when called from a category 1 ISR or from an alarm callback function.
AVAILABILITY	No restrictions.
RETURNS:	
IsrId	Success

NAME	**OS_UserGetIsrId**
OS_NULLISR	Not called from an ISR.

7.2.4.2.2.56. OS_UserGetResource()

NAME	**OS_UserGetResource**
SYNOPSIS	Enter a critical section
SYNTAX	`os_result_t OS_UserGetResource ( os_resourceid_t r )`
DESCRIPTION	`OS_UserGetResource()` allows the calling task to enter a critical section of code associated with the resource. Other tasks that use the same resource must wait until this task releases the resource again. A task may not call `OS_UserGetResource()` for a resource that it already holds.
AVAILABILITY	`OS_UserGetResource()` may be used in tasks. On some architectures `OS_UserGetResource()` can be called from Category 2 ISRs as well.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.57. OS_UserGetScheduleTableStatus()

NAME	**OS_UserGetScheduleTableStatus**
SYNOPSIS	Get a schedule table's status
SYNTAX	`os_result_t OS_UserGetScheduleTableStatus ( os_scheduleid_t s /* ID of schedule table */ os_uint8_t *out /* Where to put the result */ )`
DESCRIPTION	`OS_UserGetScheduleTableStatus()` writes the current status of the schedule table to the specified location.

NAME	OS_UserGetScheduleTableStatus
AVAILABILITY	No restrictions.
RETURNS:	
Status	Success
OS_E_OK	Success
OS_E_ACCESS	Schedule Table's application is not accessbile.

7.2.4.2.2.58. OS_UserGetStackInfo()

NAME	OS_UserGetStackInfo
SYNOPSIS	Get information about a stack
SYNTAX	`os_result_t OS_UserGetStackInfo ( os_taskorisr_t id, /* ID of task or ISR */ os_stackinfo_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetStackInfo()` places information about a task or ISR stack into the specified 'out' location. `OS_TaskToTOI(task_id)` should be used to specify a task id. If the task ID is OS_NULLTASK, information about the current task is returned. If there is no current task, the 'out' location is not modified and OS_E_NOFUNC is returned, but the error handler is not called. `OS_IsrToTOI(isr_id)` should be used to specify an ISR id. If the ISR_-ID is OS_NULLISR, information about the current ISR is returned. If there is no current ISR, information about the global kernel stack is returned. Depending on the architecture and on the calling mechanism of ISRs, the kernel stack may get shared for ISRs, or private ISR stacks may get used. If private ISR stacks are used - which is quite the exception - it is not advisable to estimate free ISR stack using OS_NULLISR outside of an ISR. As a special case, if the id parameter is OS_TOI_CURRENTCONTEXT, the information about the caller's context is returned. In this case the sp is always OS_NULL.

NAME	**OS_UserGetStackInfo**
	The stackPointer field of the 'out' variable is not updated if the request is for the current task. This allows the caller to place the current SP value there before calling `OS_UserGetStackInfo()` The fields isrStackBase and isrStackLen only apply to ISRs; they are set to NULL and 0 respectively, when a task queries its stack information. These fields represent the currently accessible part of the area described by the fields stackBase and stackLen. This is relevant, when memory protection is enabled.
AVAILABILITY	Can be called from Tasks and Category 2 ISRs.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.59. OS_UserGetTaskId()

NAME	**OS_UserGetTaskId**
SYNOPSIS	Get the ID of the current task
SYNTAX	`os_result_t OS_UserGetTaskId(os_taskid_t *out)`
DESCRIPTION	`OS_UserGetTaskId()` writes the ID of the current task to the user-specified location "out". If no task is currently running, `OS_NULLTASK` is written instead.
AVAILABILITY	No restrictions.
RETURNS:	
TaskId	ID of current task
OS_NULLTASK	No task is running

7.2.4.2.2.60. OS_UserGetTaskState()

NAME	**OS_UserGetTaskState**
SYNOPSIS	Get state of a task
SYNTAX	`os_result_t OS_UserGetTaskState ( os_taskid_t t, /* ID of task */ os_taskstate_t *out /* Where to put the answer */ )`
DESCRIPTION	`OS_UserGetTaskState()` writes the current state of the specified task to the location specified in the "out" parameter.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success
OS_E_ID	Invalid Task ID specified
OS_E_ACCESS	Task's application is not accessible

7.2.4.2.2.61. OS_UserIncrementCounter()

NAME	**OS_UserIncrementCounter**
SYNOPSIS	Increment a counter
SYNTAX	`os_result_t OS_UserIncrementCounter ( os_counterid_t c /* ID of the counter */ )`
DESCRIPTION	`OS_UserIncrementCounter()` increments a counter. If any alarms attached to the counter expire as a result, the configured action for that alarm is performed. The alarm action always runs in the context of the kernel.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

NAME	OS_UserIncrementCounter

7.2.4.2.2.62. OS_UserReleaseResource()

NAME	OS_UserReleaseResource
SYNOPSIS	Leave a critical section
SYNTAX	`os_result_t OS_UserReleaseResource ( os_resourceid_t r /* ID of the resource */ )`
DESCRIPTION	`OS_UserReleaseResource()` signals that the calling task has left a critical section of code associated with the resource. Other tasks that use the same resource are now permitted to run. A task must release resources in the reverse order to which they were taken.
AVAILABILITY	`OS_UserReleaseResource()` may be used in tasks. On some architectures `OS_UserReleaseResource()` can be called from Category 2 ISRs as well.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.63. OS_UserResetPeakCpuLoad()

NAME	OS_UserResetPeakCpuLoad
SYNOPSIS	Reset the CPU load monitor's peak load detector.
SYNTAX	`void OS_UserResetPeakCpuLoad ( os_coreid_t coreId /* Selects the core of which the peak load shall be reset. */ )`
DESCRIPTION	`OS_UserResetPeakCpuLoad()` resets the peak CPU load detector of the load monitoring system. The peak load latch is set to the current load, after

NAME	**OS_UserResetPeakCpuLoad**
	first ensuring that the current load is up-to-date. This is done for the specified core. If CPU load measurement is disabled, no action takes place.
AVAILABILITY	Include the `Os_salsa.h` header file. No restrictions.
RETURNS:	-

7.2.4.2.2.64. OS_UserResumeInterrupts()

NAME	**OS_UserResumeInterrupts**
SYNOPSIS	Resume interrupts up to a given level
SYNTAX	`void OS_UserResumeInterrupts(os_intlocktype_t locktype)`
DESCRIPTION	`OS_UserResumeInterrupts()` restores the interrupt level of the processor or interrupt controller to the level that it was before the corresponding call to `OS_UserSuspendInterrupts()`. It is used to implement the `ResumeOSInterrupts()`, `ResumeAllInterrupts()` and `DisableAllInterrupts()` system services by calling it with the `locktype` parameter equal to `OS_LOCKTYPE_OS`, `OS_LOCKTYPE_ALL` and `OS_LOCKTYPE_NONEST`, respectively. Both `ResumeOSInterrupts()` and `ResumeAllInterrupts()` are nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls. If `ResumeOSInterrupts()` is called from a permitted context other than a Task or Category 2 ISR it is a no-operation, or if it is called within a code section that is controlled a `ResumeAllInterrupts()` or `DisableAllInterrupts()`, it is treated as a no-operation since interrupts are already blocked at a higher level. Interrupt lock timing is implemented for Tasks and ISRs; timing state that was saved by the corresponding `OS_UserSuspendInterrupts()` is restored.
AVAILABILITY	

NAME	OS_UserResumeInterrupts
RETURNS:	-

7.2.4.2.2.65. OS_UserSchedule()

NAME	OS_UserSchedule
SYNOPSIS	Voluntarily yield the CPU
SYNTAX	`os_result_t OS_UserSchedule(void)`
DESCRIPTION	`OS_UserSchedule()` allows the calling task to yield the CPU voluntarily. Active tasks whose running priorities are lower than the running priority of the current task but higher that its base priority are allowed to run. `OS_UserSchedule()` returns when there are no more such tasks. Tasks get a higher running priority than their base priority when they are preemptive or have an internal resource allocated to them. A task that holds a standard resource is not permitted to call `OS_UserSchedule()` since this would interfere with the resource's ceiling priority.
AVAILABILITY	`OS_UserSchedule()` may only be called from a task.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.66. OS_UserSetAbsAlarm()

NAME	OS_UserSetAbsAlarm
SYNOPSIS	Set an alarm at an absolute counter value
SYNTAX	`os_result_t OS_UserSetAbsAlarm ( os_alarmid_t a, /* ID of the alarm */ os_tick_t start, /* Time of first expiry */ os_tick_t cyc /* Time of subsequent expiries */ )`

NAME	OS_UserSetAbsAlarm
DESCRIPTION	OS_UserSetAbsAlarm() sets the specified alarm to expire the next time that its counter reaches the start value and, if the cyc parameter is non zero, thereafter every cyc ticks of the counter. The values of start and cyc must lie within the permitted range configured for the counter. The specified alarm must not already be in use. If the counter is about to reach the start value, the alarm could expire before OS_UserSetAbsAlarm() returns. If the counter has already reached the specified start value, the alarm will not expire until the counter wraps around and reaches the value again. Depending on the configuration of the counter, this could be a *very* long time.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.67. OS_UserSetEvent()

NAME	OS_UserSetEvent
SYNOPSIS	Set one or more events for a task
SYNTAX	`os_result_t OS_UserSetEvent ( os_taskid_t t, /* ID of the task */ os_eventmask_t evt /* Events to set */ )`
DESCRIPTION	OS_UserSetEvent() sets the events given in evt for the specified task. If the task is waiting for one or more of the events, it will be reawakened and queued for execution. The task must be an extended task.
AVAILABILITY	
RETURNS:	

NAME	OS_UserSetEvent
OS_E_OK	Success

7.2.4.2.2.68. OS_UserSetRelAlarm()

NAME	OS_UserSetRelAlarm
SYNOPSIS	Set an alarm at a relative counter value
SYNTAX	`os_result_t OS_UserSetRelAlarm ( os_alarmid_t a, /* ID of the alarm */ os_tick_t inc, /* First expiry time */ os_tick_t cyc /* Subsequent expiry times */ )`
DESCRIPTION	`OS_UserSetRelAlarm()` sets the specified alarm to expire after `inc` ticks of its associated counter and, if the `cyc` parameter is non zero, thereafter every `cyc` ticks of the counter. The values of `start` and `cyc` must lie within the permitted range configured for the counter. The specified alarm must not already be in use. If the `inc` value is very small, the alarm could expire before `OS_UserSetRelAlarm()` returns.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.69. OS_UserSetScheduleTableAsync()

NAME	OS_UserSetScheduleTableAsync
SYNOPSIS	Synchronise a schedule table to global time

NAME	**OS_UserSetScheduleTableAsync**
SYNTAX	`os_result_t OS_UserSetScheduleTableAsync(void)`
DESCRIPTION	`OS_UserSetScheduleTableAsync()`
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.70. OS_UserShutdownOs()

NAME	**OS_UserShutdownOs**
SYNOPSIS	Shut down the OS kernel
SYNTAX	`void OS_UserShutdownOs ( os_uint32_t code /* Shutdown code */ )`
DESCRIPTION	`OS_UserShutdownOs()` shuts down the OS kernel. Interrupts are disabled, the scheduler is stopped. If the shutdown hook is configured it is called with the `code` as the parameter. If and when the shutdown hook returns, the kernel waits until the CPU is powered down or reset.
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.71. OS_UserStartOs()

NAME	**OS_UserStartOs**
SYNOPSIS	Start the OS

NAME	**OS_UserStartOs**
SYNTAX	`void OS_KernStartOs ( os_uint8_t mode /* Startup mode */ )`
DESCRIPTION	`OS_UserStartOs()` starts the OS. The `mode` parameter determines the set of tasks and alarms that should be started automatically. After the kernel data structures have been initialized, the startup hook is called, if it has been configured. Normally `OS_UserStartOs()` does not return. If the OS has already been started or the `mode`parameter is not valid the function could return, depending on how the error handler is defined to handle the error.
AVAILABILITY	`OS_UserStartOs()` can only be called once, from outside the OS. It is typically called from the system's `main()` function.
RETURNS:	-

7.2.4.2.2.72. OS_UserStartScheduleTable()

NAME	**OS_UserStartScheduleTable**
SYNOPSIS	Start a schedule table
SYNTAX	`os_result_t OS_UserStartScheduleTable ( os_scheduleid_t s /* ID of table */ os_tick_t offset, /* Time of first event */ os_boolean_t rel /* TRUE if offset is relative */ )`
DESCRIPTION	`OS_UserStartScheduleTable()` starts a schedule table such that the first expiry point occurs either `offset` ticks from now or when the underlying counter reaches the absolute `offset` value, depending on the value of `rel`.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

NAME	OS_UserStartScheduleTable

7.2.4.2.2.73. OS_UserStartScheduleTableSynchron()

NAME	OS_UserStartScheduleTableSynchron
SYNOPSIS	Start a schedule table synchronously
SYNTAX	`os_result_t OS_UserStartScheduleTableSynchron ( os_scheduleid_t s /* ID of table */ )`
DESCRIPTION	`OS_UserStartScheduleTableSynchron()` places a schedule table into the WAITING state so that it will start synchronously when global time becomes available.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.74. OS_UserStopScheduleTable()

NAME	OS_UserStopScheduleTable
SYNOPSIS	Stop a schedule table
SYNTAX	`os_result_t OS_UserStopScheduleTable ( os_scheduleid_t s /* ID of table */ )`
DESCRIPTION	`OS_UserStopScheduleTable()` stops a schedule table immediately. If another schedule table has been chained behind the specified schedule table, that chained table is also placed in the STOPPED state. If the specified schedule table is itself in the CHAINED state, the chaining link is broken.
AVAILABILITY	
RETURNS:	

NAME	OS_UserStopScheduleTable
OS_E_OK	Success

7.2.4.2.2.75. OS_UserSuspendInterrupts()

NAME	OS_UserSuspendInterrupts
SYNOPSIS	Suspend interrupts up to a given level
SYNTAX	`void OS_UserSuspendInterrupts(os_intlocktype_t locktype)`
DESCRIPTION	`OS_UserSuspendInterrupts()` raises the interrupt level of the processor or interrupt controller to a level that depends on the `locktype` parameter. It is used to implement the `SuspendOSInterrupts()`, `SuspendAllInterrupts()` and `DisableAllInterrupts()` system services by calling it with the `locktype` parameter equal to `OS_LOCKTYPE_OS`, `OS_LOCKTYPE_ALL` and `OS_LOCKTYPE_NONEST`, respectively. Both `SuspendOSInterrupts()` and `SuspendAllInterrupts()` are nestable; this is implemented by a counter. The interrupt level is only truly manipulated on the outermost of the nested calls. If `SuspendOSInterrupts()` is called from a permitted context other than a Task or Category 2 ISR it is a no-operation, or if it is called within a code section that is controlled a `SuspendAllInterrupts()` or `DisableAllInterrupts()`, it is treated as a no-operation since interrupts are already blocked at a higher level. Interrupt lock timing is implemented for Tasks and ISRs; the current context's "OS Interrupts Lock Time" is used for `SuspendOSInterrupts()` and "All Interrupts Lock Time" is used for the other two system services. If timing is already active its state is saved before activating the interrupt lock timing. WARNING: if `SuspendOSInterrupts()` is called for the first time within a code section protected by `SuspendAllInterrupts()` or `DisableAllInterrupts()`, the "OS" interrupt lock timing is not activated. The checker should always ensure that if the "OS Interrupt Lock Time" is activated for

NAME	OS_UserSuspendInterrupts
	an OS-Object, the "All interrupt lock time" is also activated and is less than or equal to the "OS interrupt lock time"
AVAILABILITY	
RETURNS:	-

7.2.4.2.2.76. OS_UserSyncScheduleTable()

NAME	OS_UserSyncScheduleTable
SYNOPSIS	Synchronise a schedule table to global time
SYNTAX	`os_result_t OS_UserSyncScheduleTable ( os_scheduleid_t s, /* Schedule table */ os_tick_t globalTime /* Current global time */ )`
DESCRIPTION	`OS_UserSyncScheduleTable()` sets up the synchronisation variables of the schedule table such that the period will be adjusted at the next and subsequent end-of-round interrupts, subject to the configured maximum increase and maximum decrease values, until the time discrepancy is zero. When performing the adjustment, the adjustment direction is chosen so as to minimise the number of rounds taken to perform the synchronisation. The local time needed for the calculations is itself caclulated from the time-to-next-interrupt and the offset of the next expiry point. This means that processing delays in the schedule table mechanisms, including this function, cannot be eliminated.
AVAILABILITY	No restrictions.
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.77. OS_UserTerminateApplication()

NAME	**OS_UserTerminateApplication**
SYNOPSIS	Terminates the current application
SYNTAX	`os_result_t OS_UserTerminateApplication(os_applicationid_t, os_restart_t)`
DESCRIPTION	`OS_UserTerminateApplication(os_applicationid_t, os_restart_t)` disables all ISRs, alarms, scheduletables and tasks of the application with the given ID. Afterwards a possibly configured restart task will be activated if the parameter of TerminateApplication is set to RESTART.
AVAILABILITY	
RETURNS:	
OS_E_OK	Success

7.2.4.2.2.78. OS_UserTerminateTask()

NAME	**OS_UserTerminateTask**
SYNOPSIS	Terminate the current task
SYNTAX	`os_result_t OS_UserTerminateTask(void)`
DESCRIPTION	`OS_UserTerminateTask()` terminates the current task. The calling task is transferred from the *running* state to the *suspended* state. The calling task must have released all resources and resumed all suspended interrupts before calling `OS_UserTerminateTask()`. The function does not normally return unless an error is detected.
AVAILABILITY	`OS_UserTerminateTask()` may only be called from a task.
RETURNS:	-

7.2.4.2.2.79. OS_UserWaitEvent()

NAME	**OS_UserWaitEvent**
SYNOPSIS	Wait for one of a set of events
SYNTAX	`os_result_t OS_UserWaitEvent ( os_eventmask_t e )`
DESCRIPTION	`OS_UserWaitEvent()` causes the calling task to wait until one or more of the events specified in the `e` parameter occurs. If an event is already pending, the function returns immediately. Otherwise, the task enters the *waiting* state until one of the events occurs. Calling `OS_UserWaitEvent()` with an empty set of events is considered to be an error and handled accordingly.
AVAILABILITY	`OS_UserWaitEvent()` may only be called from an extended task.
RETURNS:	
OS_E_OK	Success

7.2.5. Permitted calling context

Figure 7.1, “Allowed calling context for OS service calls table” shows which Os API function is callable from which context.

Service	Task	Cat1 ISR	Cat2 ISR	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
ActivateTask	✓		✓							
TerminateTask	✓		C							
ChainTask	✓		C							
Schedule	✓		C							
GetTaskID	✓		✓	✓	✓	✓				✓
GetTaskState	✓		✓	✓	✓	✓				
DisableAllInterrupts	✓	✓	✓							
EnableAllInterrupts	✓	✓	✓							
SuspendAllInterrupts	✓	✓	✓	✓	✓	✓			✓	
ResumeAllInterrupts	✓	✓	✓	✓	✓	✓			✓	
SuspendOSInterrupts	✓	✓	✓							
ResumeOSInterrupts	✓	✓	✓							
GetResource	✓		✓							
ReleaseResource	✓		✓							
SetEvent	✓		✓							
ClearEvent	✓		C							
GetEvent	✓		✓	✓	✓	✓				
WaitEvent	✓		C							
GetAlarmBase	✓		✓	✓	✓	✓				
GetAlarm	✓		✓	✓	✓	✓				
SetRelAlarm	✓		✓							
SetAbsAlarm	✓		✓							
CancelAlarm	✓		✓							
GetActiveApplicationMode	✓		✓	✓	✓	✓	✓	✓		
StartOS										
ShutdownOS	✓		✓	✓			✓			
GetApplicationID	✓		✓	✓	✓	✓	✓	✓	✓	✓
GetISRID	✓		✓	✓						✓
CallTrustedFunction	✓		✓							
CheckISRMemoryAccess	✓		✓	✓						✓
CheckTaskMemoryAccess	✓		✓	✓						✓
CheckObjectAccess	✓		✓	✓						✓
CheckObjectOwnership	✓		✓	✓						✓
StartScheduleTableRel	✓		✓							
StartScheduleTableAbs	✓		✓							
StopScheduleTable	✓		✓							
NextScheduleTable	✓		✓							
StartScheduleTableSynchron	✓		✓							
SyncScheduleTable	✓		✓							
GetScheduleTableStatus	✓		✓							
SetScheduleTableAsync	✓		✓							
IncrementCounter	✓		✓							
GetCounterValue	✓		✓							
GetElapsedCounterValue	✓		✓							
TerminateApplication	✓		✓	✓						
DisableInterruptSource	✓		✓							
EnableInterruptSource	✓		✓							

Figure 7.1. Allowed calling context for OS service calls table

NOTE

C indicates that validity is only Checked in Extended status by `E_OS_CALLEVEL`. Calling `TerminateApplication` is only allowed in application specific error hooks.

7.3. Generator Error Codes

When generating or verifying a EB tresos AutoCore OS configuration the code generator may issue the following errors. The error is printed in the Message Window of the Generator or to the standard output in command line mode with the following information:

Code
: The ErrorCode

Short Description
: A short description of the error (printed in italic in the following tables

Architecture specific codes can be found in the architecture notes for the corresponding hardware architecture.

1. Errors

Code	Description
`OS_4`	*Could not launch Os generator: {0}* The generator executable could not be started. This is an internal error, please contact the vendor.
`OS_5`	*OS-Generation failed for project {0}* The Autosar OS generator reported an error during generation.
`OS_8`	*The time limit in the release clearance has been exceeded.* The Os generator is used after the date specified in the release clearance file.
`OS_9`	*Aborting: Link phase of generator failed. Please contact vendor.* An internal error occurred during the link stage of the generation process. Please file a bug report.
`OS_10`	*Aborting: Setup phase of generator failed. Please contact vendor.* An internal error occurred during the setup stage of the generation process. Please file a bug report.

Code	Description
`OS_11`	*Could not read the architecture database.* The architecture database could not be read. Verify your installation.
`OS_12`	*The feature is not supported by the license. Please obtain one of the following license features: {0}.* The license does not support the requested feature. Please contact the vendor.
`OS_13`	*The release clearance information is inconsistent.* The release clearance information is inconsistent. This hints at an installation problem of the OS plugin. Please try to re-install it.
`OS_20`	*An Os element of type {0} was configured without name.* All Os components must have a name. An Os object without name was passed to the generator.
`OS_21`	*The name of {0} {1} is not a valid C identifier.* The names of all configured Os objects must be valid C identifiers.
`OS_22`	*The name of {0} {1} uses a reserved Autosar OS prefix.* The names of all configured Os objects must not start with the prefix OS_ (case insensitive).
`OS_23`	*Parameter {0} has the invalid value {1}.* The value of the specified parameter is invalid and not recognized by the AutosarOS generator.
`OS_24`	*{0} {1}: The reference to {2} {3} is invalid.* The target object of the specified reference does not exist. Verify that the correct object was selected.
`OS_25`	*More than {0} {1} elements have been defined.* The maximum number of OS objects of the given type was exceeded. Reduce the number of objects of this type.
`OS_26`	*{0} {1} does not have the permission to access {2} {3}.* Access to the object was denied. Ensure that both objects are in the same application or grant the right via the accessing applications list.
`OS_32`	*{0} {1} is only available for the microkernel.*

Code	Description
	Please choose a different value.
OS_33	*Stack slot {0} has a calculated size of {1} which is larger than the allowed maximum {2}.* Check configured stack sizes for tasks, ISRs and the operating system.
OS_34	*{0} {1} is only available for EB tresos AutoCore OS but not for EB tresos Safety OS.* Please choose a different value.
OS_100	*Parameter {0}: Wrong conformance class {1}. Conformance class should be {2}.* Features were selected that are not supported by the configured conformance class. Adjust the conformance class or disable the conformance class parameter for auto-calculation.
OS_107	*Parameter {0}: Wrong OS schedule type {1}. Schedule type should be {2}.* Task properties were selected that are not supported by the configured OS schedule type. Adjust the schedule type or disable the schedule type parameter for auto-calculation.
OS_113	*Your configured execution timer {0} belongs to counter {1}.* The configured execution timer is already used by a hardware counter. Select another timer.
OS_115	*CPU load measurement is enabled, but a timestamp timer has not been configured.* For CPU load measurement to work, a timer has to be selected as time stamp timer.
OS_117	*Timer {0}, which is selected as timestamp timer, was not found.* A timer has been selected which should also be used for a hardware counter, but the timer could not be found in the system. Check the timestamp timer and counter configuration.
OS_118	*Execution time protection is enabled, but the highest interrupt priority is used by another interrupt.* The interrupt priority of the execution timer must be higher than that of all other interrupts, therefore it uses the highest available priori-

Code	Description
	ty. This value must not be assigned to another interrupt or hardware counter.
`OS_119`	*The interrupt priority of the timestamp timer cannot get assigned.* The interrupt priority of the timestamp timer needs to be higher than that of all category 2 interrupts. A spare priority could not be assigned. To resolve this problem, rearrange the interrupt priorities.
`OS_121`	*OsInitCoreId ({0}) must lie between 0 and OsMaxNumberOfCores ({1}) or be -1 (which means a value is chosen automatically).* OsInitCoreId must lie between 0 and OsMaxNumberOfCores or be -1 (which means a value is chosen automatically).
`OS_122`	*Requested less cores than used: OsNumberOfCores is {0} while {1} cores are used.* OsNumberOfCores must be the number of cores which are under control of the OS.
`OS_200`	*Priority of element {0} is out of range. Minimum: {1}, Maximum: {2}* The given priority is out of range. The priorities must be within the displayed boundaries. Please note that some architectures may have a reverse priority scheme for interrupts, i.e. lower values mean higher priorities.
`OS_201`	*{0} defines an interrupt or resource lock time which exceeds its execution budget.* The maximum lock time for interrupts or resources cannot be longer than the maximum allowed run time of the task or interrupt.
`OS_202`	*Stack of element {0} is out of range. Minimum: {1}, Maximum {2}* The specified stack size is out of range. The stack size must be within the displayed boundaries.
`OS_204`	*{0} specifies locking time for unreferenced resource {1}. Add resource {1} to the resource list.* A locking budget can only be specified for resources that are actually used by the task or interrupt.
`OS_206`	*Timing protection is not allowed when category 1 interrupts are used.* Timing protection and category 1 interrupts must not be used together.

Code	Description
OS_207	*Timing protection is not supported in scalability class {0}.* Timing protection is only supported in scalability classes 2 and above.
OS_208	*{0} {1}: Resource {2} is referenced multiple times.* The task or interrupt references a resource multiple times. Delete the additional references.
OS_209	*{0} {1}: Timing protection and/or runtime measurement is enabled, but no execution timer is selected in OsCoreConfig.* An execution timer must be selected if timing protection or runtime measurement is to be performed.
OS_210	*{0} {1}: Rate monitoring is enabled, but no timestamp timer is selected in OsOS.* A timestamp timer must be selected if arrival rate monitoring (i.e. a time frame and a count limit are set for a Task or Isr) is enabled.
OS_211	*{0} {1}: OS interrupt lock budget of {2} exceeds all interrupt lock budget of {3}.* If interrupt lock timing protection is configured, the interrupt lock budget for all interrupts must be larger than or equal to the lock budget of OS interrupts.
OS_212	*Couldn't allocate cross core level. Please leave a free interrupt level below Cat-1 and above Cat-2 ISRs.* The generator failed to allocate the cross core level between Cat-1 and Cat-2 ISRs.
OS_231	*Task {0} is a basic task, but specifies events.* Basic tasks may not use events. Either remove the events or change the task type to an extended task.
OS_232	*Task {0} references {1} internal resources. Only one is allowed.* Tasks may occupy only one internal resource. Remove the other internal resources or change their resource type.
OS_233	*Extended task {0} specifies more than one activation.* Extended tasks allow only one activation. Reduce the number of activations or make it a basic task.

Code	Description
OS_234	*Number of activations of task {0} is out of range. Minimum: {1}, Maximum: {2}* The given number of activations is out of range. The number of activations must be within the displayed boundaries.
OS_260	*Category 1 interrupt {0} uses resources. Only category 2 interrupts may use resources.* Category 1 interrupts must not use resources. Change the interrupt type to category 2 or delete the references to resources.
OS_261	*Interrupt {0} references internal resources, which is not allowed.* Internal resources may only be used by tasks. Delete all references to internal resources.
OS_262	*Interrupt {0}: Invalid category {1}. Only category 1 and 2 interrupts are allowed.* The configured category is not supported by Autosar OS.
OS_263	*Interrupt {0}: Configured vector {1} does not exist on {2}* The configured interrupt vector does not exist on this MCU.
OS_264	*Interrupt {0}: vector {1} is already in use.* The configured interrupt vector is already in use.
OS_265	*Interrupt {0}: the priority of the category 1 interrupt is lower than or equal to that of a category 2 interrupt.* The priority of category 1 interrupts must be higher than that of any category 2 interrupt.
OS_266	*The lowest priority or all priorities in a configuration with only category 1 interrupts are used.* In a configuration with only category 1 interrupts, the lowest priority must not be used, it is reserved for internal use.
OS_267	*Interrupt {0}: Priority {1} of the category 1 interrupt is too high for the current configuration.* Depending on execution budget monitoring and multicore settings, priorities at or above {2} are reserved for internal use.
OS_268	*Interrupt {0}: Priority {1} of the category 2 interrupt is too high for the current configuration.*

Code	Description
	Depending on execution budget monitoring and multicore settings, priorities at or above {2} are reserved for internal use.
`OS_301`	*Resource RES_SCHEDULER is configured, but RES_SCHEDULER is disabled via OsOS/UseResScheduler.* The usage of RES_SCHEDULER has to be enabled via OsOS/UseResScheduler, otherwise access to this resource is not possible.
`OS_302`	*Linked resource {0} links to RES_SCHEDULER, but RES_SCHEDULER is disabled via OsOS/UseResScheduler.* The usage of RES_SCHEDULER has to be enabled via OsOS/UseResScheduler, otherwise access to this resource is not possible.
`OS_303`	*The resource RES_SCHEDULER must be of type STANDARD.* The only allowed type of the resource RES_SCHEDULER is STANDARD.
`OS_305`	*Linked resource {0} links to itself.* Linked resources may not link to themselves. Check that no circular reference was created.
`OS_306`	*Resource {0} links to internal resource {1}. Links to internal resources are not allowed.* Linked references to internal resources are not allowed.
`OS_309`	*User defined RES_SCHEDULER is not allowed in multi-core configurations.* You may only override RES_SCHEDULER if only one core is used. In multi-core configurations RES_SCHEDULER exists once per used core.
`OS_310`	*Resource {0} is used by {1} {2} and {4} {5} that are on different cores, {3} and {6} respectively.* Resource is used by any TASKs or ISRs assigned to different cores.
`OS_401`	*Parameter {0} of Alarm {1} exceeds MaxAllowedValue of counter {2}.* The alarm time value (first alarm or periodic) exceeds the maximum allowed counter value of the selected counter.

Code	Description
`OS_402`	*Parameter {0} of Alarm {1} is below the MinCycle value of counter {2}.* The cycle time of the alarm must be larger than the MinCycle value of the associated counter.
`OS_403`	*Alarm {0} references event {1} which is not used by task {2}* The event to be set by the alarm must also be referenced by the associated task. Add the event to be set to the event list of the task.
`OS_404`	*Alarm {0} increments the counter {1}, which is used to trigger the alarm.* An alarm must not increment the counter that is used to trigger the alarm.
`OS_405`	*Parameter {0} of Alarm {1} specifies invalid callback function name {2}.* The callback function name must be a valid C identifier.
`OS_408`	*Alarm {0} increments Counter {1}, which is on a different core. This is not supported by the Safety OS.* Alarms must be on the same core as the counter they shall increment.
`OS_409`	*Alarm {0} uses Counter {1}, which is on a different core. This is not supported by the EB tresos AutoCore OS.* Alarms must be on the same core as the counter.
`OS_410`	*Alarm {0} increments its own counter {1} indirectly through an alarm chain.* An alarm must not increment its own counter indirectly through an alarm chain.
`OS_499`	*Counter {1} drives Alarm {0} which is on a different core.* Alarms must be on the same core as the corresponding counter.
`OS_500`	*Event {0} and event {1} use an overlapping event mask.* All events used by a group of tasks must have a unique bit mask, i.e. an event mask must not exist twice in the group of all tasks using these events.
`OS_501`	*Event {0} specifies a bit mask without any bit set.*

Code	Description
	At least one bit must be set in event mask. Disable the parameter to use auto-calculation of event masks.
`OS_502`	*Event {0} uses multiple bits in its mask.* Every event may only use a single bit for its event mask.
`OS_503`	*A mask could not be assigned to event {0}. The maximum of {1} events was exceeded.* A maximum number of events can be configured per task group. This maximum was exceeded, so that no mask could be calculated.
`OS_600`	*Application {0} is empty.* An application must contain Os objects and may not be empty.
`OS_601`	*Number of OS applications exceeded. Maximum is {0}.* Only the displayed number of OS applications is allowed by the system.
`OS_602`	*Application {0} claims the resource RES_SCHEDULER which may not be owned by any application.* The special resource RES_SCHEDULER must not belong to any application.
`OS_603`	*Application {0} claims {1} which is already owned by another application.* Os objects must not belong to more than one application. Configure the accessing applications to grant permission.
`OS_604`	*Non-trusted Application {0} claims interrupt {1} which is a category 1 interrupt.* Category 1 interrupts can only belong to trusted applications.
`OS_605`	*Application {0} claims task {1} as restart task, but does not own it.* The restart task of an application must belong to the application that references it.
`OS_606`	*The following elements do not belong to an application: {0}* In strict Autosar, all Os objects must belong to an application, if applications are used. Enable OsOS/OsAutosarCustomization/OsPermitSystemObjects to relax this constraint.

Code	Description
`OS_607`	*Parameter {0} of application {1} specifies an invalid stack size for the hook. Minimum: {2}, Maximum {3}* The specified stack size for the application hook is out of range. The stack size must be within the displayed boundaries.
`OS_608`	*Trusted function {0} of application {1} specifies an invalid stack size. Minimum: {2}, Maximum {3}* The specified stack size for the trusted function is out of range. The stack size must be within the displayed boundaries.
`OS_610`	*Scalability class {0} does not support applications.* The configured scalability class of the system does not support applications. Increase the scalability class to allow applications.
`OS_613`	*Application {0} is assigned to core {1}, but there is/are only {2} core(s).* Applications can't be assigned to inexistent cores.
`OS_614`	*Missing OsApplicationCoreAssignment at application {0}. Either none or all applications must have a core assignment.* Single-core applications must not be mixed with multi-core applications.
`OS_701`	*Schedule table {0}: no expiry points defined.* A schedule table must define at least one expiry point.
`OS_703`	*Schedule table {0} references event {1} which is not used by task {2}.* The event to be set by the schedule table must also be referenced by the associated task. Add the event to be set to the event list of the task.
`OS_704`	*Schedule table {0}: expiry point {1} at offset {2} exceeds the schedule table duration.* The offset of an expiry point must not exceed the duration of the schedule table.
`OS_705`	*ScheduleTable {0}: expiry point {1} uses a MaxAdvance value which exceeds the schedule table's duration.*

Code	Description
	For synchronizable schedule tables, the the OsScheduleTableMax-Advance value must not be larger than the duration of the schedule table.
`OS_706`	*Schedule table {0}: expiry point {1} at offset {2} exceeds the counter's maximum allowed value.* The offsets of the schedule table must not exceed the maximum allowed value of the attached counter.
`OS_707`	*ScheduleTable {0}: time delta between expiry point {1} and expiry point {2} (plus maxAdvance value {3}) exceeds the counter's maximal allowed value.* The time difference between two expiry points, also considering the synchronization, must not exceed the maximum allowed value of the attached counter.
`OS_708`	*Schedule table {0}: time delta between expiry point {1} (plus maxAdvance value {2}) and expiry point {3} (plus maxAdvance value {4}) in next round exceeds the counter's maximal allowed value.* For repeating schedule tables, the time delta between the first offset of the next round and the last offset of the previous round, also considering the synchronization, must not exceed the maximum allowed value of the attached counter.
`OS_709`	*ScheduleTable {0}: time delta between expiry point {1} (plus max-Advance value {2}) and the end of the schedule table exceeds the counter's maximal allowed value.* The time difference between the end of the schedule table and the last expiry point, also considering the synchronization, must not exceed the maximum allowed value of the attached counter.
`OS_710`	*Schedule table {0} is synchronizable but is attached to software counter {1}.* Synchronizable schedule tables must be attached to a hardware counter.
`OS_711`	*Schedule table {0} uses implicit synchronization, but the duration does not equal the counters maximum allowed value + 1.* Schedule tables using implicit synchronization must have a duration of the counter's maximum allowed value + 1.

Code	Description
`OS_712`	*Schedule table {0}: time delta between expiry point {1} and expiry point {2} (minus maxRetard value {3}) is below the counter's minimum cycle value.* The time difference between two expiry points, also considering the synchronization, must not be lower than the minimum cycle value of the attached counter.
`OS_713`	*ScheduleTable {0} uses explicit synchronization, but the precision is greater than half of the duration.* The precision of a schedule table using explicit synchronization must not be larger than half of its duration.
`OS_714`	*Schedule table {0}: the selected scalability class {1} does not support synchronization.* In the selected scalability class, synchronization is not available.
`OS_715`	*ScheduleTable {0}: expiry point {1} specifies no action.* An expiry point must not be empty. Delete the expiry point if it is not needed.
`OS_716`	*Schedule table {0}, expiration point {1}: Task and event lists do not match.* To every event a corresponding task has to be given. If this error occurs, the task and event lists in the specified expiration point of the schedule table do not match.
`OS_717`	*Schedule table {0}: duration exceeds the drive counter.* A schedule table that is explicitly synchronized shall have a duration not greater than modulus of the drive counter.
`OS_718`	*Schedule table {0}, expiration point {1}: Initial Offset is not 0 or in the range OsCounterMinCycle .. OsCounterMaxAllowedValue.* The Initial Offset shall be zero OR in the range OsCounterMinCycle .. OsCounterMaxAllowedValue of the underlying counter.
`OS_719`	*Schedule table {0}: The final Delay between Expiry Point {1} and the end of the Schedule Table is out of range.* The value of Final Delay of a periodic Schedule Table shall be in the range OsCounterMinCycle .. OsCounterMaxAllowedValue of the underlying counter.

Code	Description
`OS_721`	*Schedule table {0}: autostart value of {1} exceeds the counter's maximum allowed value.* The start value for starting the schedule table automatically must not exceed the maximum allowed value of the attached counter.
`OS_722`	*Schedule table {0}: counter {1} should be on the same core* Schedule table and counter on different cores
`OS_800`	*Counter {0}: Configured hardware timer {1} does not exist on {2}.* The selected hardware timer does not exist on the configured MCU. Choose another hardware timer.
`OS_803`	*Counter {0}: value of parameter {1} is lower than the wrap value {2} of hardware timer {3}.* The maximum allowed value of a hardware counter must be at least equal to the wrap value of the attached hardware timer.
`OS_805`	*Counter {0} tries to use hardware timer {1} which is already in use.* The configured hardware timer is already in use. Select another hardware timer for the counter.
`OS_806`	*Counter {0} of type software is incremented by multiple timer drivers.* A software timer can only be automatically incremented by a single driver, e.g. either GPT driver or a hardware module.
`OS_807`	*Counter {0}: Configured counter incrementer module {1} does not exist on {2}.* The selected hardware module for incrementing the software counter does not exist on the configured MCU. Choose another module.
`OS_808`	*Counter {0}: no timer period specified for counter incrementer module {1}. Please configure OsSecondsPerTick.* If a counter of type HARDWARE is used, an interrupt level for the timer has to be configured.
`OS_809`	*Counter {0}: vector {1} of counter incrementer module {2} is already in use.* The interrupt vector for the incrementer module is already in use. Select another module/channel or verify your interrupt configuration.

Code	Description
`OS_810`	*Counter {0}: the incrementer module is already in use by hardware counter {1}.* The hardware incrementer module must not be used as a hardware counter.
`OS_811`	*Counter {0}: the incrementer module is already in use by software counter {1}.* A hardware incrementer module can only drive a single software counter.
`OS_812`	*Counter {0}: the OS does not support GPT-driven hardware counters.* The OS does not support GPT-driven hardware counters, only GPT driven software counters.
`OS_814`	*Counter {0}: An interrupt level has to be configured for a counter of type HARDWARE.* The maximum resolution the OS supports for counter values is 1ns per tick. Values below 1ns will be truncated, which may lead to inaccuracies in the conversion to counter ticks (and vice versa).
`OS_900`	*Timer {0}: vector {1} is already in use.* The interrupt vector for the hardware timer is already in use. Select another hardware timer or verify your interrupt configuration.
`OS_1101`	*IOC configuration may not contain callbacks on architectures with memory protection* The IOC is configured to use notification callbacks. Callbacks are, however, not supported on architectures with memory protection.
`OS_1102`	*IocCommunication {0}: If callbacks are used, only one receiver object may exist.* If callback notification shall be used, only 1 receiver object must exist.
`OS_1103`	*IocCommunication {0}, datatype container {1}: datatype name \"{2}\" is invalid.* The DataTypeName parameter in the IocDataType container must be a valid C data type.
`OS_1105`	*The parameter OsIocIntraCoreLockType of IocCommunication {0}, is set to {1}, which is an invalid value.*

Code	Description
	The OsIocIntraCoreLockType parameter shall have a valid value.
`OS_1107`	*At IOC channel {0}: Primitive data types, like {1}, can not have variable length.* Primitive data type can not have variable length.
`OS_1108`	*IOC channel {0} has no data elements.* Every channel must at least have one data element.
`OS_1109`	*IOC channel {0} must not have an init-symbol (OsIocInitValueSymbol). Reason: {1}.* Only fixed-length non-group last-is-best channels may have an init-symbol.
`OS_1200`	*Parameter {0} specifies an invalid stack size. Minimum: {1}, Maximum: {2}* The specified stack size is out of range. The stack size must be within the displayed boundaries.
`OS_1201`	*Parameter {0} specifies an invalid function name: \"{1}\".* The specified function name is invalid or empty. It must be a vaild C identifier.
`OS_1202`	*{0} {1} is not available for the microkernel.* Please choose a different value.
`OS_1203`	*The executable region \"{0}\" is part of one or more dynamic partitions.* The microkernel does not support dynamic partitions which contain executable regions on this derivative.
`OS_1205`	*At {0}: The number of {3} regions required by your configuration ({1}) exceeds the number of memory regions available on this derivative ({2}).* Maximum number of memory regions exceeded.
`OS_1206`	*Memory region {0} has access permission EXECUTE, which is not supported or reasonable for this target. Use READ_EXECUTE instead.* Access permission EXECUTE is not supported for this target. READ_EXECUTE should be used instead.

Code	Description
`OS_1207`	*Memory region {0} has MkMemoryRegionInitializePerCore but not MkMemoryRegionInitialize. Please set the latter or unset the first one.* If a region has MkMemoryRegionInitializePerCore it must have MkMemoryRegionInitialize.
`OS_1300`	*Counter {0} is configured as microkernel ticker, but OsCounterMaxAllowedValue ({1}) is larger than 2**30.* The specified OsCounterMaxAllowedValue is too large. It must be at most 2**30.
`OS_1301`	*Counter {0} is configured as microkernel ticker, but is used by more than one element.* References from more than one ScheduleTable or Alarm are present. Only one ScheduleTable is allowed.
`OS_1303`	*Counter {0} is configured as microkernel ticker, but alarm {1} is attached to it.* Only a ScheduleTable may get attached to a microkernel ticker.
`OS_1304`	*ScheduleTable {0} is attached to a microkernel ticker, but does not use TICKS as time unit.* Simple ScheduleTables (implemented in the microkernel) only support TICKS as time unit.
`OS_1305`	*The simple schedule table {0} is attached to a microkernel counter but uses synchronization (i.e., OsScheduleTblSyncStrategy != NONE).* Simple schedule tables (part of microkernel) must have OsScheduleTblSyncStrategy set to NONE.
`OS_1306`	*ScheduleTable {0} is attached to the microkernel counter {1}. The duration {2} of {0}, though, differs from {1}'s modulus, which is {3} + 1 (i.e., OsCounterMaxAllowedValue + 1).* The duration of simple schedule tables must be equal to the duration of the attached microkernel counter.
`OS_1307`	*The number of configured fast partitions ({0}) exceeds the number of fast partitions supported for this derivative ({1}).* Maximum number of fast partitions exceeded.

Code	Description
`OS_1400`	*Invalid spinlock self-reference. Spinlock {0} is successor of Spinlock {1}, which itself is successor of Spinlock {0}.* Spinlock successor chains must not form a loop.
`OS_1401`	*Spinlock {0} has an unknown lock method ({1}).* Spinlocks must have a valid lock method. The default method is LOCK_NOTHING.

1. Warnings

Code	Description
`OS_6`	*This is an untested version, do not use for production code!* The Os generator is an untested version which has not been cleared for production use.
`OS_7`	*This is a time-restricted version. Days left: {0}* The Os generator has an expiration date specified in the release clearance file. The remaining days are displayed.
`OS_31`	*{0} {1} contains a duplicate reference to the {2} {3}.* The duplicate reference has been removed internally.
`OS_101`	*Parameter {0}: Unsuitable conformance class {1}. Conformance class could be {2} (Optimization).* The selected conformance class provides more features than actually used. The kernel could be optimized by using a lower conformance class. Disable the conformance class parameter for auto-calculation.
`OS_102`	*Parameter {0}: Wrong scalability class {1}. Scalability class should be {2}.* Features were selected that are not supported by the configured scalability class. Adjust the scalability class or disable the scalability class parameter for auto-calculation.
`OS_105`	*Parameter {0}: Unsuitable scalability class {1}. Scalability class could be {2} (Optimization).* The selected scalability class provides more features than actually used. The kernel could be optimized by using a lower scalability class. Disable the scalability class parameter for auto-calculation.

Code	Description
OS_108	*Parameter {0}: Unsuitable OS schedule type {1}. OS schedule type could be {2} (Optimization).* The selected OS schedule type provides more features than actually used. The kernel could be optimized by using another schedule type. Disable the schedule type parameter for auto-calculation.
OS_109	*{0} is {1}, but should be EXTENDED for scalability class {2}.* Autosar requires the OS status type EXTENDED for scalability classes 3 and above.
OS_110	*{0} is {1}, but EXTENDED is recommended for scalability class {2}.* Autosar recommends the OS status type EXTENDED for scalability classes 1 and 2.
OS_112	*Non-trusted applications and category 1 interrupts found. Memory protection is recommended.* Autosar recommends memory protection when non-trusted applications and category 1 interrupts are used together.
OS_114	*Memory protection is disabled via OsProtection. Do not use in production environment!* A system providing memory protection is configured ((i.e. Trapping is allowed), but the memory protection (OsProtection) has been turned off. This is only a debugging help, turn on memory protection for production use.
OS_120	*The configured initialization core (OsInitCoreId={0}) has no applications.* The configured initialization core (OsInitCoreId) has no applications. This means it will be controlled by the OS, but will do nothing after start-up.
OS_123	*Requested more cores than used: OsNumberOfCores is {0} while {1} cores are used.* Requested more cores than used.
OS_124	*Ignoring {0} {1}, which was configured for unused core {2}* Ignored a core configuration item, which was configured for an unused core.

Code	Description
OS_203	*Element {0} has no stack. Ensure that the element does not require any stack (this includes local variables or function calls).* No stack was given for the task or interrupt. Ensure that the object really needs no stack.
OS_205	*{0} {1} belongs to an untrusted application and has unlimited execution budget.* The task or interrupt belongs to an untrusted application and uses timing protection, but an execution budget was not specified.
OS_230	*Task {0} is set to autostart, but does not specify an application mode. Using OSDEFAULTAPPMODE.* An application mode should be defined if a task is configured to start automatically. If none is given, the standard OSDEFAULTAPPMODE will be used.
OS_304	*Accessing applications will be ignored for RES_SCHEDULER.* The special resource RES_SCHEDULER is available to all tasks in the system. Application permissions will therefore be ignored.
OS_307	*Resource {0} is configured, but not used.* Warning that no task or interrupt uses the resource.
OS_308	*Resource {0} used only once. It has therefore no effect on priority ceiling calculations.* Warning that only one task or interrupt uses the resource. This resource thus has no influence on task or interrupt priorities.
OS_400	*Alarm {0} is set to autostart, but does not specify an application mode. Using OSDEFAULTAPPMODE.* An application mode should be defined if an alarm is configured to start automatically. If none is given, the standard OSDEFAULTAPPMODE will be used.
OS_406	*Scalability class {0} does not support alarm callbacks.* The configured scalability class of the system does not allow alarm callbacks. Lower the scalability class to allow alarm callbacks.
OS_407	*Alarm {0} increments Counter {1}, which is on a different core. This is not supported by AUTOSAR.*

Code	Description
	Alarms should be on the same core as the counter they shall increment.
OS_504	*Event {0} is configured, but not used.* Warning that no task uses the configured event.
OS_609	*Trusted function {0} in application {1}: The default stack size {2} will be used.* A stack size was not specified for the trusted function, this the default size will be used.
OS_611	*Application {0} is non-trusted, but trapping is disabled.* If non-trusted applications are used on a system with memory protection, trapping should be enabled in the OsOS configuration container.
OS_612	*Scalability class {0} is not intended to support applications.* The configured scalability class of the system is not intended to support applications. Autosar requires scalability class 3 or higher for applications.
OS_700	*Schedule table {0} is set to autostart, but does not specify an application mode. Using OSDEFAULTAPPMODE.* An application mode should be defined if a schedule table is configured to start automatically. If none is given, the standard OSDEFAULTAPPMODE will be used.
OS_720	*Schedule table {0}: Worst case times can be violated by the Offset of Expiry Point {1}.* Worst case times can be violated if ((offset-maxRetard) - (offsetPrev+maxAdvance)) < counterMinValue or (offset+maxAdvance) > duration or (offset-maxRetard) < 0.
OS_804	*No hardware counter available that could be used as system counter. Corresponding macros are not available.* A hardware counter which could act as a system counter could not be found. The macros OSMAXALLOWEDVALUE, OSTICKSPERBASE, OSMINCYCLE and OSTICKDURATION are not available. If you have a counter defined, use the counter-specific macros of type <macroname>_<counter>.
OS_813	*Counter {0}: OsSecondsPerTick has a resolution below 1ns. Values smaller than 1ns will be truncated.*

Code	Description
	The maximum resolution the OS supports for counter values is 1ns per tick. Values below 1ns will be truncated, which may lead to inaccuracies in the conversion to counter ticks (and vice versa).
`OS_1104`	*The parameter OsIocIntraCoreLockType of IocCommunication {0}, is set to {1}, which is not permitted for trapping channels if the microkernel is used.* The fallback value NO_LOCK was automatically chosen for code generation. Please set OsIocIntraCoreLockType to NO_LOCK.
`OS_1106`	*Ignored OsIocUseInterCoreLock of IocCommunication {0}, because this is a single core configuration.* Inter core locks are never used in single core configurations.
`OS_1208`	*Currently StartupHook support of Safety OS is limited. Be aware that the StartupHook will run in QM-OS context.* Currently dedicated StartupHook threads are not supported.
`OS_1302`	*Counter {0} is configured as microkernel ticker, but is not used by any element.* No reference from a ScheduleTable is present. This means that the counter is not used at all.

1. Information

Code	Description
`OS_1`	**** AutosarOS {0}.{1}.{2} Build {3} ({4}/{5}) **** The version, build, target architecture and derivate of the Autosar OS generator.
`OS_3`	*OS-Generation succeeded for project {0}* The Autosar OS generator finished successfully.

7.4. Kernel Error Codes

Using this appendix you can quickly find the exact cause of any error that causes an `ErrorHook`, `ProtectionHook` or `ShutdownHook` function to be called. These hook functions are called with the standard OSEK/VDX or Autosar error codes as parameter. The standard OSEK/VDX or Autosar error codes are not very informative, so EB tresos AutoCore OS provides extended error information which can be found in the error information structure. The function `OS_GetErrorInfo()` returns this structure.

In the error information structure there are three fields of interest, `result`, `serviceId` and `errorCondition`. The `result` field contains the OSEK/VDX or Autosar error code. The `serviceId` field identifies which kernel function reported the error. The `errorCondition` field identifies exactly what the error is. The meaning of the field `result` can be translated using Section 7.4.1, “List of OSEK/VDX and Autosar Error Codes”. The meaning of the field `serviceId` can be translated using Section 7.4.2, “List of Service Identifiers”. The meaning of the field `errorCondition` can be translated using Section 7.4.3, “List of Error Identifiers”.

NOTE

Most numerical values in the following tables are implementation defined and may change. The numerical values are listed to help with debuggers that don't show the symbolic names. In application code only the symbolic names should be used.

Further information can be obtained by looking at the detailed error description for the service and error code (see Section 7.4.4, “Detailed Error Descriptions”). This provides a full description of the error, along with the action that will be taken and the OSEK/VDX error code that is associated with the error.

7.4.1. List of OSEK/VDX and Autosar Error Codes

The OSEK/VDX and Autosar error codes are returned to the caller by various OS services and are passed as parameter to the `ErrorHook`, `ProtectionHook` and `ShutdownHook` functions. The OSEK/VDX and Autosar error code is also stored in the structure returned by `OS_GetErrorInfo()` in the field `result`.

Value	Identifier
0	E_OK
1	E_OS_ACCESS
2	E_OS_CALLEVEL
3	E_OS_ID
4	E_OS_LIMIT
5	E_OS_NOFUNC
6	E_OS_RESOURCE
7	E_OS_STATE
8	E_OS_VALUE
9	E_OS_STACKFAULT
10	E_OS_PROTECTION_MEMORY
11	E_OS_PROTECTION_TIME

Value	Identifier
12	E_OS_PROTECTION_LOCKED
13	E_OS_PROTECTION_ARRIVAL
14	E_OS_PROTECTION_EXCEPTION
15	E_OS_ILLEGAL_ADDRESS
16	E_OS_DISABLEDINT
17	E_OS_MISSINGEND
18	E_OS_SERVICEID

7.4.2. List of Service Identifiers

The service identifier specifies which kernel function reported the error. It is stored in the structure returned by `OS_GetErrorInfo()` in the field `serviceId`.

Value	Identifier
0	OS_SID_GetApplicationId
1	OS_SID_GetIsrId
2	OS_SID_CallTrustedFunction
3	OS_SID_CheckIsrMemoryAccess
4	OS_SID_CheckTaskMemoryAccess
5	OS_SID_CheckObjectAccess
6	OS_SID_CheckObjectOwnership
7	OS_SID_StartScheduleTableRel
8	OS_SID_StartScheduleTableAbs
9	OS_SID_StopScheduleTable
10	OS_SID_ChainScheduleTable
11	OS_SID_StartScheduleTableSynchron
12	OS_SID_SyncScheduleTable
13	OS_SID_SetScheduleTableAsync
14	OS_SID_GetScheduleTableStatus
15	OS_SID_IncrementCounter
16	OS_SID_GetCounterValue
17	OS_SID_GetElapsedCounterValue

Value	Identifier
18	OS_SID_TerminateApplication
19	OS_SID_AllowAccess
20	OS_SID_GetApplicationState
21	OS_SID_UnknownSyscall
22	OS_SID_ActivateTask
23	OS_SID_TerminateTask
24	OS_SID_ChainTask
25	OS_SID_Schedule
26	OS_SID_GetTaskId
27	OS_SID_GetTaskState
28	OS_SID_SuspendInterrupts
29	OS_SID_ResumeInterrupts
30	OS_SID_GetResource
31	OS_SID_ReleaseResource
32	OS_SID_SetEvent
33	OS_SID_ClearEvent
34	OS_SID_GetEvent
35	OS_SID_WaitEvent
36	OS_SID_GetAlarmBase
37	OS_SID_GetAlarm
38	OS_SID_SetRelAlarm
39	OS_SID_SetAbsAlarm
40	OS_SID_CancelAlarm
41	OS_SID_GetActiveApplicationMode
42	OS_SID_StartOs
43	OS_SID_ShutdownOs
44	OS_SID_GetStackInfo
45	OS_SID_DisableInterruptSource
46	OS_SID_EnableInterruptSource
47	OS_SID_TryToGetSpinlock
48	OS_SID_ReleaseSpinlock

Value	Identifier
49	OS_SID_ShutdownAllCores
50	OS_SID_GetCpuLoad
51	OS_SID_ResetPeakCpuLoad
52	OS_SID_Dispatch
53	OS_SID_TrapHandler
54	OS_SID_IsrHandler
55	OS_SID_RunSchedule
56	OS_SID_KillAlarm
57	OS_SID_TaskReturn
58	OS_SID_HookHandler
59	OS_SID_ArchTrapHandler
60	OS_SID_MemoryManagement

7.4.3. List of Error Identifiers

The error identifier specifies exactly what the error is. It is stored in the structure returned by `OS_GetErrorInfo()` in the field `errorCondition`.

Value	Identifier
0	OS_ERROR_NoError
0	OS_ERROR_UnknownError
1	OS_ERROR_UnknownSystemCall
2	OS_ERROR_InvalidTaskId
3	OS_ERROR_InvalidTaskState
4	OS_ERROR_Quarantined
5	OS_ERROR_MaxActivations
6	OS_ERROR_WriteProtect
7	OS_ERROR_ReadProtect
8	OS_ERROR_ExecuteProtect
9	OS_ERROR_InvalidAlarmId
10	OS_ERROR_InvalidAlarmState
11	OS_ERROR_AlarmNotInUse

Value	Identifier
12	OS_ERROR_WrongContext
13	OS_ERROR_HoldsResource
14	OS_ERROR_NoEvents
15	OS_ERROR_TaskNotExtended
16	OS_ERROR_TaskNotInQueue
17	OS_ERROR_InvalidCounterId
18	OS_ERROR_CorruptAlarmList
19	OS_ERROR_ParameterOutOfRange
20	OS_ERROR_AlarmInUse
21	OS_ERROR_AlreadyStarted
22	OS_ERROR_InvalidStartMode
23	OS_ERROR_AlarmNotInQueue
24	OS_ERROR_InvalidResourceId
25	OS_ERROR_ResourceInUse
26	OS_ERROR_ResourcePriorityError
27	OS_ERROR_ResourceNestingError
28	OS_ERROR_TaskSuspended
29	OS_ERROR_NestingUnderflow
30	OS_ERROR_NestingOverflow
31	OS_ERROR_NonfatalException
32	OS_ERROR_FatalException
33	OS_ERROR_UnhandledNmi
34	OS_ERROR_UnknownInterrupt
35	OS_ERROR_TaskTimeBudgetExceeded
36	OS_ERROR_IsrTimeBudgetExceeded
37	OS_ERROR_UnknownTimeBudgetExceeded
38	OS_ERROR_Permission
39	OS_ERROR_ImplicitSyncStartRel
40	OS_ERROR_CounterIsHw
41	OS_ERROR_InvalidScheduleId
42	OS_ERROR_NotRunning

Value	Identifier
43	OS_ERROR_NotStopped
44	OS_ERROR_AlreadyChained
45	OS_ERROR_InvalidObjectType
46	OS_ERROR_InvalidObjectId
47	OS_ERROR_InvalidApplicationId
48	OS_ERROR_InvalidIsrId
49	OS_ERROR_InvalidMemoryRegion
50	OS_ERROR_NotChained
51	OS_ERROR_InvalidFunctionId
52	OS_ERROR_NotSyncable
53	OS_ERROR_NotImplemented
54	OS_ERROR_StackError
55	OS_ERROR_RateLimitExceeded
56	OS_ERROR_InterruptDisabled
57	OS_ERROR_ReturnFromTask
58	OS_ERROR_InsufficientStack
59	OS_ERROR_WatchdogTimeout
60	OS_ERROR_PllLockLost
61	OS_ERROR_ArithmeticTrap
62	OS_ERROR_MemoryProtection
63	OS_ERROR_NotTrusted
64	OS_ERROR_TaskResLockTimeExceeded
65	OS_ERROR_IsrResLockTimeExceeded
66	OS_ERROR_TaskIntLockTimeExceeded
67	OS_ERROR_IsrIntLockTimeExceeded
68	OS_ERROR_IncrementZero
69	OS_ERROR_DifferentCounters
70	OS_ERROR_ScheduleTableNotIdle
71	OS_ERROR_InvalidRestartOption
72	OS_ERROR_TaskAggregateTimeExceeded
73	OS_ERROR_IncorrectKernelNesting

Value	Identifier
74	OS_ERROR_KernelStackOverflow
75	OS_ERROR_TaskStackOverflow
76	OS_ERROR_IntEException
77	OS_ERROR_ExceptionInKernel
78	OS_ERROR_SysReq
79	OS_ERROR_StackOverflow
80	OS_ERROR_StackUnderflow
81	OS_ERROR_SoftBreak
82	OS_ERROR_UndefinedOpcode
83	OS_ERROR_AccessError
84	OS_ERROR_ProtectionFault
85	OS_ERROR_IllegalOperandAccess
86	OS_ERROR_UnknownException
87	OS_ERROR_UndefinedInstruction
88	OS_ERROR_Overflow
89	OS_ERROR_BrkInstruction
90	OS_ERROR_WdgTimer
91	OS_ERROR_NMI
92	OS_ERROR_RegisterBank
93	OS_ERROR_DebugInterface
94	OS_ERROR_InsufficientPageMaps
95	OS_ERROR_InsufficientHeap
96	OS_ERROR_TLB_multiple_hit
97	OS_ERROR_Userbreak
98	OS_ERROR_InstructionAddressError
99	OS_ERROR_InstructionTlbMiss
100	OS_ERROR_TlbProtectionViolation
101	OS_ERROR_GeneralIllegalInstruction
102	OS_ERROR_SlotIllegalInstruction
103	OS_ERROR_GeneralFPUDisable
104	OS_ERROR_SlotFPUDisable

Value	Identifier
105	OS_ERROR_DataAddressErrorRead
106	OS_ERROR_DataAddressErrorWrite
107	OS_ERROR_DataTlbMissRead
108	OS_ERROR_DataTlbMissWrite
109	OS_ERROR_DataTlbReadProtViolation
110	OS_ERROR_DataTlbWriteProtViolation
111	OS_ERROR_FpuException
112	OS_ERROR_InitialPageWrite
113	OS_ERROR_UnconditionalTrap
114	OS_ERROR_PrefetchAbort
115	OS_ERROR_DataAbort
116	OS_ERROR_IllegalSupervisorCall
117	OS_ERROR_IllegalInterrupt
118	OS_ERROR_NonMaskableInterrupt
119	OS_ERROR_HardFault
120	OS_ERROR_MemoryManagement
121	OS_ERROR_BusFault
122	OS_ERROR_UsageFault
127	OS_ERROR_SupervisorCall
128	OS_ERROR_DebugMonitor
130	OS_ERROR_PendingSupervisorCall
131	OS_ERROR_SystemTick
132	OS_ERROR_OscillatorFailureTrap
133	OS_ERROR_StackErrorTrap
134	OS_ERROR_AddressErrorTrap
135	OS_ERROR_MathErrorTrap
136	OS_ERROR_DMACErrorTrap
137	OS_ERROR_GenericHardTrap
138	OS_ERROR_GenericSoftTrap
139	OS_ERROR_UnknownTrap
140	OS_ERROR_SysErr

Value	Identifier
141	OS_ERROR_HVTrap
142	OS_ERROR_FETrap
143	OS_ERROR_Trap
144	OS_ERROR_ReservedInstruction
145	OS_ERROR_CoprocessorUnusable
146	OS_ERROR_PrivilegedInstruction
147	OS_ERROR_MisalignedAccess
148	OS_ERROR_FEINT
149	OS_ERROR_InvalidSpinlockId
150	OS_ERROR_InvalidSpinlockNesting
151	OS_ERROR_SpinlockAlreadyHeld
152	OS_ERROR_SpinlockInterferenceDeadlock
153	OS_ERROR_CoreIsDown
154	OS_ERROR_InvalidCoreId
155	OS_ERROR_ApplicationNotAccessible
156	OS_ERROR_ApplicationNotRestarting
157	OS_ERROR_HoldsLock
158	OS_ERROR_SpinlockNotOccupied
159	OS_ERROR_CallTrustedFunctionCrosscore

7.4.4. Detailed Error Descriptions

The following tables show the detailed error description for the service and error codes of each OS service. This provides a full description of the error, along with the action that will be taken and the OSEK/VDX error code that is associated with the error. The OSEK/VDX code is the code that is passed to the `ErrorHook`, `ProtectionHook` and `ShutdownHook` functions and returned to the caller if applicable.

NOTE The standard mode action and code only apply if the condition is actually checked. If you are using a precompiled library this is always true, but if you are using an optimized kernel many error conditions are not tested for.

7.4.4.1. UnknownSyscall

ServiceID:	UnknownSyscall (21)
ErrorID:	UnknownSystemCall (1)
Description:	A system call has been made with an invalid or unconfigured system-call index. This could be caused by calling a system service for features that are not configured, or by executing the SYSCALL instruction with an out-of-range operand.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.2. ActivateTask

ServiceID:	ActivateTask (22)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the task belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	ActivateTask (22)
ErrorID:	CoreIsDown (153)
Description:	The core on which the alarm task has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	ActivateTask (22)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.

Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ActivateTask (22)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ActivateTask (22)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ActivateTask (22)
ErrorID:	Quarantined (4)
Description:	The specified task has been quarantined and will not be activated.
Standard action/code:	RETURN / OS_E_DENIED
Extended action/code:	RETURN / OS_E_DENIED

ServiceID:	ActivateTask (22)
ErrorID:	MaxActivations (5)
Description:	The specified task has exceeded its activation limit.
Standard action/code:	RETURN / OS_E_LIMIT
Extended action/code:	RETURN / OS_E_LIMIT

ServiceID:	ActivateTask (22)
ErrorID:	RateLimitExceeded (55)
Description:	The specified task has exceeded its activation rate limit.
Standard action/code:	RETURN / OS_E_RATEPROT
Extended action/code:	RETURN / OS_E_RATEPROT

ServiceID:	ActivateTask (22)
ErrorID:	Permission (38)

Description:	Permission has not been granted for the caller to access the referenced task.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.3. TerminateTask

ServiceID:	TerminateTask (23)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	TerminateTask (23)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	TerminateTask (23)
ErrorID:	HoldsLock (157)
Description:	The terminating task still occupies one or more spinlocks.
Standard action/code:	QUARANTINE / OS_E_SPINLOCK
Extended action/code:	RETURN / OS_E_SPINLOCK

ServiceID:	TerminateTask (23)
ErrorID:	HoldsResource (13)
Description:	The terminating task still occupies one or more resources.
Standard action/code:	QUARANTINE / OS_E_RESOURCE
Extended action/code:	RETURN / OS_E_RESOURCE

7.4.4.4. ChainTask

ServiceID:	ChainTask (24)

ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	ChainTask (24)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ChainTask (24)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ChainTask (24)
ErrorID:	HoldsLock (157)
Description:	The terminating task still occupies one or more spinlocks.
Standard action/code:	QUARANTINE / OS_E_SPINLOCK
Extended action/code:	RETURN / OS_E_SPINLOCK

ServiceID:	ChainTask (24)
ErrorID:	HoldsResource (13)
Description:	The terminating task still occupies one or more resources.
Standard action/code:	QUARANTINE / OS_E_RESOURCE
Extended action/code:	RETURN / OS_E_RESOURCE

ServiceID:	ChainTask (24)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ChainTask (24)
ErrorID:	Quarantined (4)
Description:	The specified task has been quarantined and cannot be activated.
Standard action/code:	RETURN / OS_E_DENIED
Extended action/code:	RETURN / OS_E_DENIED

ServiceID:	ChainTask (24)
ErrorID:	MaxActivations (5)
Description:	The specified task has exceeded its activation limit.
Standard action/code:	RETURN / OS_E_LIMIT
Extended action/code:	RETURN / OS_E_LIMIT

ServiceID:	ChainTask (24)
ErrorID:	RateLimitExceeded (55)
Description:	The specified task has exceeded its activation rate limit.
Standard action/code:	RETURN / OS_E_RATEPROT
Extended action/code:	RETURN / OS_E_RATEPROT

ServiceID:	ChainTask (24)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced task.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	ChainTask (24)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the task belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.5. GetTaskState

ServiceID:	GetTaskState (27)

ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the task belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetTaskState (27)
ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetTaskState (27)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetTaskState (27)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetTaskState (27)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetTaskState (27)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID

Extended action/code:	RETURN / OS_E_ID

7.4.4.6. Schedule

ServiceID:	Schedule (25)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	Schedule (25)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	Schedule (25)
ErrorID:	HoldsLock (157)
Description:	The calling task still occupies one or more spinlocks.
Standard action/code:	QUARANTINE / OS_E_SPINLOCK
Extended action/code:	RETURN / OS_E_SPINLOCK

ServiceID:	Schedule (25)
ErrorID:	HoldsResource (13)
Description:	The calling task still occupies one or more resources.
Standard action/code:	QUARANTINE / OS_E_RESOURCE
Extended action/code:	RETURN / OS_E_RESOURCE

7.4.4.7. GetAlarm

ServiceID:	GetAlarm (37)
ErrorID:	CoreIsDown (153)

Description:	The core on which the alarm resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetAlarm (37)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetAlarm (37)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetAlarm (37)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetAlarm (37)
ErrorID:	InvalidAlarmId (9)
Description:	The specified alarm ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetAlarm (37)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced alarm.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetAlarm (37)
ErrorID:	InvalidAlarmState (10)
Description:	The specified alarm is in an invalid state. This is an internal kernel error. Please notify your vendor.
Standard action/code:	SHUTDOWN / OS_E_PANIC
Extended action/code:	SHUTDOWN / OS_E_PANIC

ServiceID:	GetAlarm (37)
ErrorID:	AlarmNotInUse (11)
Description:	The specified alarm is not currently in use.
Standard action/code:	RETURN / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.8. GetAlarmBase

ServiceID:	GetAlarmBase (36)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the alarm belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetAlarmBase (36)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetAlarmBase (36)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetAlarmBase (36)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetAlarmBase (36)
ErrorID:	InvalidAlarmId (9)
Description:	The specified alarm ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.9. CancelAlarm

ServiceID:	CancelAlarm (40)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the alarm belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	CancelAlarm (40)
ErrorID:	CoreIsDown (153)
Description:	The core on which the alarm resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	CancelAlarm (40)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CancelAlarm (40)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CancelAlarm (40)
ErrorID:	InvalidAlarmId (9)
Description:	The specified alarm ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	CancelAlarm (40)
ErrorID:	AlarmNotInUse (11)
Description:	The specified alarm is not currently in use.
Standard action/code:	RETURN / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

ServiceID:	CancelAlarm (40)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced alarm.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.10. SetRelAlarm

ServiceID:	SetRelAlarm (38)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the alarm belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetRelAlarm (38)

ErrorID:	CoreIsDown (153)
Description:	The core on which the alarm resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	SetRelAlarm (38)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SetRelAlarm (38)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	SetRelAlarm (38)
ErrorID:	InvalidAlarmId (9)
Description:	The specified alarm ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SetRelAlarm (38)
ErrorID:	IncrementZero (68)
Description:	The value of the increment parameter is zero. This is not permitted by AUTOSAR.
Standard action/code:	RETURN / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	SetRelAlarm (38)
ErrorID:	ParameterOutOfRange (19)
Description:	One or both of the specified increment and cycle parameters is out of range.
Standard action/code:	QUARANTINE / OS_E_VALUE

Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	SetRelAlarm (38)
ErrorID:	Quarantined (4)
Description:	The specified alarm has been quarantined and will not be activated.
Standard action/code:	RETURN / OS_E_DENIED
Extended action/code:	RETURN / OS_E_DENIED

ServiceID:	SetRelAlarm (38)
ErrorID:	AlarmInUse (20)
Description:	The specified alarm is already in use.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	SetRelAlarm (38)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced alarm.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.11. SetAbsAlarm

ServiceID:	SetAbsAlarm (39)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the alarm belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetAbsAlarm (39)
ErrorID:	CoreIsDown (153)
Description:	The core on which the alarm resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE

Extended action/code:	RETURN / OS_E_CORE

ServiceID:	SetAbsAlarm (39)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SetAbsAlarm (39)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	SetAbsAlarm (39)
ErrorID:	InvalidAlarmId (9)
Description:	The specified alarm ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SetAbsAlarm (39)
ErrorID:	ParameterOutOfRange (19)
Description:	One or both of the specified increment and cycle parameters is out of range.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	SetAbsAlarm (39)
ErrorID:	Quarantined (4)
Description:	The specified alarm has been quarantined and will not be activated.
Standard action/code:	RETURN / OS_E_DENIED
Extended action/code:	RETURN / OS_E_DENIED

ServiceID:	SetAbsAlarm (39)

ErrorID:	AlarmInUse (20)
Description:	The specified alarm is already in use.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	SetAbsAlarm (39)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced alarm.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.12. GetResource

ServiceID:	GetResource (30)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetResource (30)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetResource (30)
ErrorID:	InvalidResourceId (24)
Description:	The specified resource ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetResource (30)

ErrorID:	ResourceInUse (25)
Description:	The specified resource is in use.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetResource (30)
ErrorID:	ResourcePriorityError (26)
Description:	The specified resource has a lower ceiling priority than the base priority of the calling task. The probable cause is that the task does not declare the resource.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetResource (30)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced resource.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.13. ReleaseResource

ServiceID:	ReleaseResource (31)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ReleaseResource (31)
ErrorID:	InvalidResourceId (24)
Description:	The specified resource ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ReleaseResource (31)

ErrorID:	ResourceNestingError (27)
Description:	The specified resource has not been taken by the task, or another resource needs to be released first. Resources must be released in the reverse order to which they were taken.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

ServiceID:	ReleaseResource (31)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

7.4.4.14. WaitEvent

ServiceID:	WaitEvent (35)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	WaitEvent (35)
ErrorID:	NoEvents (14)
Description:	The task has called WaitEvent but has specified no events to wait for.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	WaitEvent (35)
ErrorID:	TaskNotExtended (15)
Description:	The calling task is not an extended task. Only extended tasks are permitted to wait for events.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	WaitEvent (35)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	WaitEvent (35)
ErrorID:	HoldsLock (157)
Description:	The terminating task still occupies one or more spinlocks.
Standard action/code:	QUARANTINE / OS_E_SPINLOCK
Extended action/code:	RETURN / OS_E_SPINLOCK

ServiceID:	WaitEvent (35)
ErrorID:	HoldsResource (13)
Description:	The terminating task still occupies one or more resources.
Standard action/code:	QUARANTINE / OS_E_RESOURCE
Extended action/code:	RETURN / OS_E_RESOURCE

ServiceID:	WaitEvent (35)
ErrorID:	RateLimitExceeded (55)
Description:	The calling task has exceeded its configured rate limit when waiting for an event that was already pending.
Standard action/code:	RETURN / OS_E_RATEPROT
Extended action/code:	RETURN / OS_E_RATEPROT

7.4.4.15. SetEvent

ServiceID:	SetEvent (32)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the event belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetEvent (32)
ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	SetEvent (32)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SetEvent (32)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	SetEvent (32)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SetEvent (32)
ErrorID:	TaskSuspended (28)
Description:	The specified task is currently suspended or quarantined.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	SetEvent (32)
ErrorID:	TaskNotExtended (15)
Description:	The specified task is not an extended task. Only extended tasks are permitted to wait for events.

Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetEvent (32)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced task.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetEvent (32)
ErrorID:	RateLimitExceeded (55)
Description:	The specified task has exceeded its activation rate limit.
Standard action/code:	RETURN / OS_E_RATEPROT
Extended action/code:	RETURN / OS_E_RATEPROT

7.4.4.16. GetEvent

ServiceID:	GetEvent (34)
ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetEvent (34)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetEvent (34)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.

Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetEvent (34)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetEvent (34)
ErrorID:	TaskNotExtended (15)
Description:	The specified task is not an extended task. Only extended tasks are permitted to wait for events.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetEvent (34)
ErrorID:	TaskSuspended (28)
Description:	The specified task is currently suspended or quarantined.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	GetEvent (34)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

7.4.4.17. ClearEvent

ServiceID:	ClearEvent (33)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.

Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ClearEvent (33)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ClearEvent (33)
ErrorID:	TaskNotExtended (15)
Description:	The specified task is not an extended task. Only extended tasks are permitted to wait for events.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.18. StartOs

ServiceID:	StartOs (42)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted. This probably means that the OS has already been started.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	StartOs (42)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	StartOs (42)
ErrorID:	InvalidStartMode (22)
Description:	The specified startup (application) mode is invalid.

Standard action/code:	SHUTDOWN / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.19. ShutdownOs

ServiceID:	ShutdownOs (43)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ShutdownOs (43)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ShutdownOs (43)
ErrorID:	NotTrusted (63)
Description:	ShutdownOS is not permitted from a non-trusted application.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.20. SuspendInterrupts

ServiceID:	SuspendInterrupts (28)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SuspendInterrupts (28)

ErrorID:	NestingOverflow (30)
Description:	Too many nested calls to SuspendOSInterrupts. A possible cause is that the calls to SuspendOSInterrupts/ResumeOSInterrupts are not correctly nested.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.21. ResumeInterrupts

ServiceID:	ResumeInterrupts (29)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ResumeInterrupts (29)
ErrorID:	NestingUnderflow (29)
Description:	The calls to SuspendOSInterrupts/ResumeOSInterrupts are not correctly nested.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.22. IncrementCounter

ServiceID:	IncrementCounter (15)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the counter belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	IncrementCounter (15)
ErrorID:	CoreIsDown (153)

Description:	The core on which the counter resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	IncrementCounter (15)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	IncrementCounter (15)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	IncrementCounter (15)
ErrorID:	InvalidCounterId (17)
Description:	The specified counter ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	IncrementCounter (15)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced counter.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	IncrementCounter (15)
ErrorID:	CounterIsHw (40)
Description:	The referenced counter is a hardware counter and cannot be advanced by software.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.23. GetStackInfo

ServiceID:	GetStackInfo (44)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetStackInfo (44)
ErrorID:	InvalidTaskId (2)
Description:	The specified task ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetStackInfo (44)
ErrorID:	InvalidIsrId (48)
Description:	The specified ISR ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetStackInfo (44)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

7.4.4.24. IsrHandler

ServiceID:	IsrHandler (54)
ErrorID:	InsufficientStack (58)
Description:	There isn't enough kernel stack left to run the ISR.
Standard action/code:	SHUTDOWN / OS_E_STACKPROT
Extended action/code:	SHUTDOWN / OS_E_STACKPROT

ServiceID:	IsrHandler (54)
ErrorID:	RateLimitExceeded (55)
Description:	The ISR has exceeded its trigger rate limit.
Standard action/code:	RETURN / OS_E_RATEPROT
Extended action/code:	RETURN / OS_E_RATEPROT

ServiceID:	IsrHandler (54)
ErrorID:	HoldsLock (157)
Description:	The ISR terminated without freeing all spinlocks that were taken.
Standard action/code:	QUARANTINE / OS_E_ISRRETURNSPINLOCKED
Extended action/code:	RETURN / OS_E_ISRRETURNSPINLOCKED

ServiceID:	IsrHandler (54)
ErrorID:	HoldsResource (13)
Description:	The ISR terminated without freeing all resources that were taken.
Standard action/code:	QUARANTINE / OS_E_ISRRETURNRESLOCKED
Extended action/code:	RETURN / OS_E_ISRRETURNRESLOCKED

ServiceID:	IsrHandler (54)
ErrorID:	InterruptDisabled (56)
Description:	The ISR terminated with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_ISRRETURNINTLOCKED
Extended action/code:	RETURN / OS_E_ISRRETURNINTLOCKED

ServiceID:	IsrHandler (54)
ErrorID:	KernelStackOverflow (74)
Description:	The ISR (probably a trusted ISR) overflowed the kernel stack.
Standard action/code:	SHUTDOWN / OS_E_STACKPROT
Extended action/code:	SHUTDOWN / OS_E_STACKPROT

7.4.4.25. HookHandler

ServiceID:	HookHandler (58)

ErrorID:	InsufficientStack (58)
Description:	There isn't enough kernel stack left to run the hook function.
Standard action/code:	RETURN / OS_E_STACKPROT
Extended action/code:	RETURN / OS_E_STACKPROT

ServiceID:	HookHandler (58)
ErrorID:	InterruptDisabled (56)
Description:	The hook function terminated with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

7.4.4.26. Dispatch

ServiceID:	Dispatch (52)
ErrorID:	KernelStackOverflow (74)
Description:	The kernel stack has overflowed. This could result in other data being overwritten, with undefined results.
Standard action/code:	SHUTDOWN / OS_E_STACKPROT
Extended action/code:	SHUTDOWN / OS_E_STACKPROT

ServiceID:	Dispatch (52)
ErrorID:	TaskStackOverflow (75)
Description:	The task stack has overflowed. This could result in other data being overwritten, with undefined results.
Standard action/code:	SHUTDOWN / OS_E_STACKPROT
Extended action/code:	SHUTDOWN / OS_E_STACKPROT

7.4.4.27. TrapHandler

ServiceID:	TrapHandler (53)
ErrorID:	UnknownInterrupt (34)
Description:	An unknown or unconfigured interrupt has occurred.
Standard action/code:	RETURN / OS_E_INTERNAL
Extended action/code:	RETURN / OS_E_INTERNAL

ServiceID:	TrapHandler (53)
ErrorID:	TaskTimeBudgetExceeded (35)
Description:	A task has exceeded its execution-time budget.
Standard action/code:	QUARANTINE / OS_E_TIMEPROT
Extended action/code:	QUARANTINE / OS_E_TIMEPROT

ServiceID:	TrapHandler (53)
ErrorID:	TaskResLockTimeExceeded (64)
Description:	A task has exceeded its resource-lock time.
Standard action/code:	QUARANTINE / OS_E_LOCKPROT
Extended action/code:	QUARANTINE / OS_E_LOCKPROT

ServiceID:	TrapHandler (53)
ErrorID:	TaskIntLockTimeExceeded (66)
Description:	A task has exceeded its interrupt-lock time.
Standard action/code:	QUARANTINE / OS_E_LOCKPROT
Extended action/code:	QUARANTINE / OS_E_LOCKPROT

ServiceID:	TrapHandler (53)
ErrorID:	IsrTimeBudgetExceeded (36)
Description:	An ISR has exceeded its execution-time budget.
Standard action/code:	QUARANTINE / OS_E_TIMEPROT
Extended action/code:	QUARANTINE / OS_E_TIMEPROT

ServiceID:	TrapHandler (53)
ErrorID:	IsrResLockTimeExceeded (65)
Description:	An ISR has exceeded its resource-lock time.
Standard action/code:	QUARANTINE / OS_E_LOCKPROT
Extended action/code:	QUARANTINE / OS_E_LOCKPROT

ServiceID:	TrapHandler (53)
ErrorID:	IsrIntLockTimeExceeded (67)
Description:	An ISR has exceeded its interrupt-lock time.
Standard action/code:	QUARANTINE / OS_E_LOCKPROT
Extended action/code:	QUARANTINE / OS_E_LOCKPROT

7.4.4.28. ChainScheduleTable

ServiceID:	ChainScheduleTable (10)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	ChainScheduleTable (10)
ErrorID:	CoreIsDown (153)
Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	ChainScheduleTable (10)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ChainScheduleTable (10)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ChainScheduleTable (10)
ErrorID:	InvalidScheduleId (41)
Description:	One or both of the referenced schedule tables does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ChainScheduleTable (10)
ErrorID:	Permission (38)

Description:	Permission has not been granted for the caller to access one or both of the referenced schedule tables.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	ChainScheduleTable (10)
ErrorID:	DifferentCounters (69)
Description:	The referenced "current" and "next" schedule tables are driven by different counters.
Standard action/code:	RETURN / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ChainScheduleTable (10)
ErrorID:	NotRunning (42)
Description:	The referenced "current" schedule table is not running.
Standard action/code:	RETURN / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

ServiceID:	ChainScheduleTable (10)
ErrorID:	NotStopped (43)
Description:	The referenced "next" schedule table is not in the STOPPED state.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

7.4.4.29. StartScheduleTableRel

ServiceID:	StartScheduleTableRel (7)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableRel (7)

ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	StartScheduleTableRel (7)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	StartScheduleTableRel (7)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	StartScheduleTableRel (7)
ErrorID:	ScheduleTableNotIdle (70)
Description:	The schedule table is already started.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableRel (7)
ErrorID:	AlarmInUse (20)
Description:	The schedule table's alarm is already in use. This indicates an internal error. Please notify your vendor.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableRel (7)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule tables does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	StartScheduleTableRel (7)
ErrorID:	ParameterOutOfRange (19)
Description:	The specified offset parameter is out of range. Either it is more than the MAXALLOWEDVALUE of the underlying counter, or it is zero.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	StartScheduleTableRel (7)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced schedule tables.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableRel (7)
ErrorID:	ImplicitSyncStartRel (39)
Description:	A schedule table configured with IMPLICIT synchronisation strategy cannot be started at a relative counter value. StartScheduleTableAbs() must be used!
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.30. StartScheduleTableAbs

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	CoreIsDown (153)
Description:	The core on which the task resides has been shut down.

Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	ScheduleTableNotIdle (70)
Description:	The schedule table is already started.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	AlarmInUse (20)
Description:	The schedule table's alarm is already in use. This indicates an internal error. Please notify your vendor.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule tables does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	StartScheduleTableAbs (8)

ErrorID:	ParameterOutOfRange (19)
Description:	The specified offset parameter is out of range. It is more than the MAXALLOWEDVALUE of the underlying counter.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced schedule table.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableAbs (8)
ErrorID:	ImplicitSyncStartRel (39)
Description:	A schedule table configured with IMPLICIT synchronisation strategy cannot be started at a relative counter value. StartScheduleTableAbs() must be used!
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.31. StartScheduleTableSynchron

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	CoreIsDown (153)
Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE

Extended action/code:	RETURN / OS_E_CORE

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	ScheduleTableNotIdle (70)
Description:	The schedule table is already started.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	AlarmInUse (20)
Description:	The schedule table's alarm is already in use. This indicates an internal error. Please notify your vendor.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule tables does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	Permission (38)

Description:	Permission has not been granted for the caller to access the referenced schedule table.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StartScheduleTableSynchron (11)
ErrorID:	NotSyncable (52)
Description:	The schedule table is not synchronisable. This is because its synchronisation parameters have not been configured. Perhaps the schedule table is attached to a software counter.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.32. RunSchedule

ServiceID:	RunSchedule (55)
ErrorID:	NotChained (50)
Description:	The chained schedule table's state was not OSEKMP_ST_-CHAINED. Perhaps the chained table is part of an application that was terminated.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

7.4.4.33. StopScheduleTable

ServiceID:	StopScheduleTable (9)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	StopScheduleTable (9)
ErrorID:	CoreIsDown (153)

Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	StopScheduleTable (9)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	StopScheduleTable (9)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	StopScheduleTable (9)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule table does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	StopScheduleTable (9)
ErrorID:	NotRunning (42)
Description:	The referenced schedule table is not running.
Standard action/code:	RETURN / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

ServiceID:	StopScheduleTable (9)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced schedule table.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.34. CheckObjectOwnership

ServiceID:	CheckObjectOwnership (6)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CheckObjectOwnership (6)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CheckObjectOwnership (6)
ErrorID:	InvalidObjectType (45)
Description:	The specified object type is unknown.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	CheckObjectOwnership (6)
ErrorID:	InvalidObjectId (46)
Description:	The referenced object does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.35. CheckObjectAccess

ServiceID:	CheckObjectAccess (5)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CheckObjectAccess (5)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CheckObjectAccess (5)
ErrorID:	InvalidObjectId (46)
Description:	The referenced object does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	CheckObjectAccess (5)
ErrorID:	InvalidObjectType (45)
Description:	The object type is invalid.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

ServiceID:	CheckObjectAccess (5)
ErrorID:	InvalidApplicationId (47)
Description:	The referenced application does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.36. CheckTaskMemoryAccess

ServiceID:	CheckTaskMemoryAccess (4)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CheckTaskMemoryAccess (4)
ErrorID:	InterruptDisabled (56)

Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CheckTaskMemoryAccess (4)
ErrorID:	InvalidTaskId (2)
Description:	The referenced task does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	CheckTaskMemoryAccess (4)
ErrorID:	InvalidMemoryRegion (49)
Description:	The specified memory region is invalid. It is either of zero length or it extends beyond the processor's addressing limits.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

7.4.4.37. CheckIsrMemoryAccess

ServiceID:	CheckIsrMemoryAccess (3)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CheckIsrMemoryAccess (3)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CheckIsrMemoryAccess (3)
ErrorID:	InvalidIsrId (48)
Description:	The referenced isr does not exist.

Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	CheckIsrMemoryAccess (3)
ErrorID:	InvalidMemoryRegion (49)
Description:	The specified memory region is invalid. It is either of zero length or it extends beyond the processor's addressing limits.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

7.4.4.38. TerminateApplication

ServiceID:	TerminateApplication (18)
ErrorID:	ApplicationNotAccessible (155)
Description:	The specified application has been terminated without restart.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	TerminateApplication (18)
ErrorID:	CoreIsDown (153)
Description:	The core on which the application resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	TerminateApplication (18)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced application.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	TerminateApplication (18)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.

Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	TerminateApplication (18)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	TerminateApplication (18)
ErrorID:	InvalidApplicationId (47)
Description:	The application could not be determined.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	TerminateApplication (18)
ErrorID:	InvalidRestartOption (71)
Description:	The restart option is neither RESTART nor NO_RESTART.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

7.4.4.39. KillAlarm

ServiceID:	KillAlarm (56)
ErrorID:	AlarmNotInQueue (23)
Description:	The specified alarm was not in its counter's alarm queue. This is an internal kernel error. Please notify your vendor.
Standard action/code:	SHUTDOWN / OS_E_INTERNAL
Extended action/code:	SHUTDOWN / OS_E_INTERNAL

7.4.4.40. CallTrustedFunction

ServiceID:	CallTrustedFunction (2)

ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the trusted function belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	CallTrustedFunction (2)
ErrorID:	InvalidFunctionId (51)
Description:	The specified trusted function does not exist.
Standard action/code:	QUARANTINE / OS_E_TFID
Extended action/code:	RETURN / OS_E_TFID

ServiceID:	CallTrustedFunction (2)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	CallTrustedFunction (2)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	CallTrustedFunction (2)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to call the referenced trusted function.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	CallTrustedFunction (2)
ErrorID:	StackError (54)
Description:	The call could result in the trusted function using stack outside the caller's stack boundary.

Standard action/code:	QUARANTINE / OS_E_STACKPROT
Extended action/code:	RETURN / OS_E_STACKPROT

ServiceID:	CallTrustedFunction (2)
ErrorID:	CallTrustedFunctionCrosscore (159)
Description:	If the target trusted function is part of an OS-Application on another core
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

7.4.4.41. GetScheduleTableStatus

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	CoreIsDown (153)
Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	InterruptDisabled (56)

Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule table does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetScheduleTableStatus (14)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

7.4.4.42. SetScheduleTableAsync

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	CoreIsDown (153)
Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	WrongContext (12)

Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule table does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	NotRunning (42)
Description:	The referenced "current" schedule table is not running.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced schedule table.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SetScheduleTableAsync (13)
ErrorID:	NotSyncable (52)
Description:	The schedule table cannot be explicitly synchronised.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.43. SyncScheduleTable

ServiceID:	SyncScheduleTable (12)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the schedule table belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SyncScheduleTable (12)
ErrorID:	CoreIsDown (153)
Description:	The core on which the schedule table resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	SyncScheduleTable (12)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	SyncScheduleTable (12)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	SyncScheduleTable (12)
ErrorID:	InvalidScheduleId (41)
Description:	The referenced schedule table does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SyncScheduleTable (12)

ErrorID:	NotRunning (42)
Description:	The referenced "current" schedule table is not running or waiting for global time.
Standard action/code:	RETURN / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	SyncScheduleTable (12)
ErrorID:	NotSyncable (52)
Description:	The schedule table is not synchronisable. This is because its synchronisation parameters have not been configured. Perhaps the schedule table is attached to a software counter.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	SyncScheduleTable (12)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced schedule table.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	SyncScheduleTable (12)
ErrorID:	ParameterOutOfRange (19)
Description:	The the specified global time is not within the period of the schedule table.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

7.4.4.44. GetTaskId

ServiceID:	GetTaskId (26)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL

Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetTaskId (26)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetTaskId (26)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

7.4.4.45. GetActiveApplicationMode

ServiceID:	GetActiveApplicationMode (41)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetActiveApplicationMode (41)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

7.4.4.46. GetIsrId

ServiceID:	GetIsrId (1)
ErrorID:	WrongContext (12)

Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetIsrId (1)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

7.4.4.47. GetApplicationId

ServiceID:	GetApplicationId (0)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetApplicationId (0)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

7.4.4.48. TaskReturn

ServiceID:	TaskReturn (57)
ErrorID:	ReturnFromTask (57)
Description:	A task returned from its main function without successfully calling TerminateTask() or ChainTask().
Standard action/code:	QUARANTINE / OS_E_TASKRETURN
Extended action/code:	KILL / OS_E_TASKRETURN

7.4.4.49. DisableInterruptSource

ServiceID:	DisableInterruptSource (45)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the ISR belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	DisableInterruptSource (45)
ErrorID:	CoreIsDown (153)
Description:	The core on which the ISR resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	DisableInterruptSource (45)
ErrorID:	InvalidIsrId (48)
Description:	The referenced isr does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.50. EnableInterruptSource

ServiceID:	EnableInterruptSource (46)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the ISR belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	EnableInterruptSource (46)
ErrorID:	CoreIsDown (153)
Description:	The core on which the ISR resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	EnableInterruptSource (46)
ErrorID:	InvalidIsrId (48)
Description:	The referenced isr does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.51. GetCounterValue

ServiceID:	GetCounterValue (16)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the counter belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetCounterValue (16)
ErrorID:	CoreIsDown (153)
Description:	The core on which the counter resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetCounterValue (16)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetCounterValue (16)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetCounterValue (16)

ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetCounterValue (16)
ErrorID:	InvalidCounterId (17)
Description:	The specified counter ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.52. GetElapsedCounterValue

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	ApplicationNotAccessible (155)
Description:	The application to which the counter belongs was terminated and has not yet restarted.
Standard action/code:	RETURN / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	CoreIsDown (153)
Description:	The core on which the counter resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetElapsedCounterValue (17)

ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	InvalidCounterId (17)
Description:	The specified counter ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetElapsedCounterValue (17)
ErrorID:	ParameterOutOfRange (19)
Description:	The PreviousValue parameter is out of range. It must not be greater than the MAXALLOWEDVALUE of the counter.
Standard action/code:	QUARANTINE / OS_E_VALUE
Extended action/code:	RETURN / OS_E_VALUE

7.4.4.53. TryToGetSpinlock

ServiceID:	TryToGetSpinlock (47)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	TryToGetSpinlock (47)

ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

ServiceID:	TryToGetSpinlock (47)
ErrorID:	InvalidSpinlockId (149)
Description:	The specified spinlock ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	TryToGetSpinlock (47)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced spinlock.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	TryToGetSpinlock (47)
ErrorID:	InvalidSpinlockNesting (150)
Description:	An attempt has been made to acquire a spinlock while still holding another spinlock or, if spinlock nesting is enabled, to acquire a spinlock that is not a successor to the spinlock that is already held.
Standard action/code:	QUARANTINE / OS_E_NESTING_DEADLOCK
Extended action/code:	RETURN / OS_E_NESTING_DEADLOCK

ServiceID:	TryToGetSpinlock (47)
ErrorID:	SpinlockAlreadyHeld (151)
Description:	An attempt has been made to acquire a spinlock that is already held by the caller.
Standard action/code:	QUARANTINE / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	TryToGetSpinlock (47)
ErrorID:	SpinlockInterferenceDeadlock (152)

Description:	An attempt has been made to acquire a spinlock that is already held by another task or ISR on the same core.
Standard action/code:	QUARANTINE / OS_E_INTERFERENCE_DEADLOCK
Extended action/code:	RETURN / OS_E_INTERFERENCE_DEADLOCK

7.4.4.54. ReleaseSpinlock

ServiceID:	ReleaseSpinlock (48)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ReleaseSpinlock (48)
ErrorID:	InvalidSpinlockId (149)
Description:	The specified spinlock ID is invalid.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	ReleaseSpinlock (48)
ErrorID:	Permission (38)
Description:	Permission has not been granted for the caller to access the referenced spinlock.
Standard action/code:	QUARANTINE / OS_E_ACCESS
Extended action/code:	RETURN / OS_E_ACCESS

ServiceID:	ReleaseSpinlock (48)
ErrorID:	SpinlockNotOccupied (158)
Description:	An attempt has been made to release a spinlock that is not held by the caller.
Standard action/code:	QUARANTINE / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	ReleaseSpinlock (48)

ErrorID:	InvalidSpinlockNesting (150)
Description:	An attempt has been made to release a spinlock that is not the most recent spinlock acquired by the caller.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

ServiceID:	ReleaseSpinlock (48)
ErrorID:	HoldsResource (13)
Description:	An attempt has been made to release a spinlock while a resource is held by the caller that has been acquired after the spinlock. Spinlocks and resources can only be acquired and released in strict LIFO order.
Standard action/code:	QUARANTINE / OS_E_RESOURCE
Extended action/code:	RETURN / OS_E_RESOURCE

7.4.4.55. AllowAccess

ServiceID:	AllowAccess (19)
ErrorID:	ApplicationNotRestarting (156)
Description:	AllowAccess was called from an application that was not restarting.
Standard action/code:	QUARANTINE / OS_E_STATE
Extended action/code:	RETURN / OS_E_STATE

ServiceID:	AllowAccess (19)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	AllowAccess (19)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL

Extended action/code:	RETURN / OS_E_CALLLEVEL

7.4.4.56. GetApplicationState

ServiceID:	GetApplicationState (20)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	GetApplicationState (20)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	GetApplicationState (20)
ErrorID:	CoreIsDown (153)
Description:	The core on which the alarm task resides has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetApplicationState (20)
ErrorID:	InvalidApplicationId (47)
Description:	The referenced application does not exist.
Standard action/code:	QUARANTINE / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

ServiceID:	GetApplicationState (20)
ErrorID:	WriteProtect (6)
Description:	The application has attempted to write to a memory area where writing is not permitted.
Standard action/code:	QUARANTINE / OS_E_ADDRESS
Extended action/code:	RETURN / OS_E_ADDRESS

7.4.4.57. ShutdownAllCores

ServiceID:	ShutdownAllCores (49)
ErrorID:	WrongContext (12)
Description:	The system service was called from a context that is not permitted.
Standard action/code:	QUARANTINE / OS_E_CALLLEVEL
Extended action/code:	RETURN / OS_E_CALLLEVEL

ServiceID:	ShutdownAllCores (49)
ErrorID:	InterruptDisabled (56)
Description:	The system service was called with interrupts disabled.
Standard action/code:	QUARANTINE / OS_E_INTDISABLE
Extended action/code:	RETURN / OS_E_INTDISABLE

ServiceID:	ShutdownAllCores (49)
ErrorID:	NotTrusted (63)
Description:	ShutdownOS is not permitted from a non-trusted application.
Standard action/code:	QUARANTINE / OS_E_NOFUNC
Extended action/code:	RETURN / OS_E_NOFUNC

7.4.4.58. GetCpuLoad

ServiceID:	GetCpuLoad (50)
ErrorID:	CoreIsDown (153)
Description:	The core for which the measurement shall be taken has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	GetCpuLoad (50)
ErrorID:	InvalidCoreId (154)
Description:	The core ID is invalid.
Standard action/code:	RETURN / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

7.4.4.59. ResetPeakCpuLoad

ServiceID:	ResetPeakCpuLoad (51)
ErrorID:	CoreIsDown (153)
Description:	The core of which the measured peak CPU load shall be reset has been shut down.
Standard action/code:	RETURN / OS_E_CORE
Extended action/code:	RETURN / OS_E_CORE

ServiceID:	ResetPeakCpuLoad (51)
ErrorID:	InvalidCoreId (154)
Description:	The core ID is invalid.
Standard action/code:	RETURN / OS_E_ID
Extended action/code:	RETURN / OS_E_ID

Bibliography

[1] *AUTOSAR Specification of Operating System*, Issue Version 3.1.0, Revision 0004, Publisher: AUTOSAR

[2] *OSEK/VDX/VDX Operating System Specification*, Issue Version 2.2.3, Publisher: OSEK/VDX

[3] *OSEK/VDX/VDX OSEK/VDX Run Time Interface (ORTI)*, Part A: Language Specification, Issue Version 2.2, Publisher: OSEK/VDX

[4] *OSEK/VDX/VDX OSEK/VDX Run Time Interface (ORTI)*, Part B: OSEK/VDX Objects and Attributes, Issue Version 2.2, Publisher: OSEK/VDX

Index

Symbols

A

B

C

D

E

G

H

I

V

W

X