Document Title	Specification of Operating System
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	034
Document Classification	Standard

Document Version	3.1.0
Document Status	Final
Part of Release	3.1
Revision	0002

Document Change History

Date	Version	Changed by	Change Description
15.01.2009	3.1.0	AUTOSAR Administration	• Changes in OS configuration: - removed "OsAppModeId" Parameter from OsAppModeContainer - added optional references from OsAppModeContainer to OsAlarm, OsTask and OsScheduleTable
04.08.2008	3.0.2	AUTOSAR Administration	Legal Disclaimer revised
17.04.2008	3.0.1	AUTOSAR Administration	• Added "OsScheduleTableDuration" parameter to configuration specification chapter
07.12.2007	3.0.0	AUTOSAR Administration	• Changed methods for timing protection • Moved configuration from OIL to AUTOSAR XML • Clarrified description for synchronization and schedule tables • Document meta information extended • Small layout adaptations made
31.01.2007	2.1.0	AUTOSAR Administration	• Added support for SoftwareFreeRunningTimer (SWFRT) incl. 2 new APIs • Added API to start a schedule table synchron • Misc. Corrections, Clarification and further explanations • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added

Document Change History

Date	Version	Changed by	Change Description
28.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template. • Major changes in chapter 10 • Structure of document changed partly • Other changes see chapter 14
28.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Content

1 Introduction and functional overview

This document describes the essential requirements on the AUTOSAR Operating System to satisfy the top-level requirements presented in the AUTOSAR SRS [2].

In general, operating systems can be split up in different groups according to their characteristics, e.g. statically configured vs. dynamically managed. To classify the AUTOSAR OS, here are the basic features: the OS

- is configured and scaled statically
- is amenable to reasoning of real-time performance
- provides a priority-based scheduling policy
- provides protective functions (memory, timing etc.) at run-time
- is hostable on low-end controllers and without external resources

This feature set defines the type of OS commonly used in the current generation of automotive ECUs, with the exception of Telematic/Infotainment systems. It is assumed that Telematic/Infotainment systems will continue to use proprietary Oss under the AUTOSAR framework (e.g. Windows CE, VxWorks, QNX, etc.). In the case where AUTOSAR components are needed to run on these proprietary Oss, the interfaces defined in this document should be provided as an Operating System Abstraction Layer (OSAL) according to requirement BSW00322 in [3].

This document uses the industry standard OSEK OS [12] (ISO 17356-3) as the basis for the AUTOSAR OS. The reader should be familiar with this standard before reading this document.

This document describes extensions to, and restrictions of, this OSEK OS.

2 Acronyms and abbreviations

Abbreviation	*Description*
API	Application Programming Interface
BSW	Basic Software Requirement
COM	Communications
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MCU	Microcontroller Unit
MPU	Memory Protection Unit
NM	Network Management
OIL	OSEK Implementation Language
OS	Operating System
OSEK/VDX	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
SWC	Software Component
SWFRT	Software FreeRunningTimer

2.1 Glossary of Terms

Term:	*Definition*	
Access Right	An indication that an object (e.g. Task, OsIsr, hook function) of an OS-Application has the permission of access or manipulation with respect to memory, OS services or (set of) OS objects.	
Cardinality	The number of items in a set.	
Counter	An operating system object that registers a count in ticks. There are two types of counters:	
	Hardware Counter	A counter that is advanced by hardware (e.g. timer). The count value is maintained by the peripheral “in hardware”.
	Software Counter	A counter which is incremented by making the `IncrementCounter()` API call. The count value is maintained by the operating system “in software”.
Deadline	The time at which a Task/Category 2 OsIsr must reach a certain point during its execution defined by system design relative to the stimulus that triggered activation. See Figure 2:1Figure	
Delay	The number of ticks between two adjacent expiry points on a schedule table. A pair of expiry points X and Y are said to be adjacent when: • There is no expiry point Z such that X.Offset < Z.Offset < Y.Offset. In this case the Delay = Y.Offset-X.Offset • X and Y are the Final Expiry Point and the Initial Expiry Point respectively. In this case Delay = (Duration-X.Offset)+Y.Offset When used in the text, Delay is a relative number of ticks measured from a specified expiry point. For example: X.Delay is the delay from X to the next expiry point.	
Deviation	The difference between the current position on an indirectly synchronized schedule table and the value of the synchronization count. .	
Duration	The number of ticks from a notional zero at which a schedule table wraps.	
Execution Time	Tasks: The net time a task spends in the `RUNNING` state without entering the `SUSPENDED` or `WAITING` state. An extended task executing the `WaitEvent()` API call to wait on an event which is already set notionally enters the `WAITING` state.	

	OsIsrs: The net time from the first to the last instruction of the user provided Category 2 interrupt handler excluding all preemptions due to higher priority OsIsrs executing in preference. Execution time includes the time spent in the error, pretask and posttask hooks and the time spent making OS service calls.	
Execution Budget	Maximum permitted execution time for a Task/OsIsr.	
Expiry Point	The offset on a Schedule Table, measured from zero, at which the OS activates tasks and/or sets events.	
	Initial Expiry Point	The expiry point with the smallest offset
	Final Expiry Point	The expiry point with the largest offset
Hook Function	A Hook function is implemented by the user and invoked by the operating system in the case of certain incidents. In order to react to these on system or application level, there are two kinds of hook functions	
	Application-specific	Hook functions within the scope of an individual OS-Application.
	System-specific	Hook functions within the scope of the complete system (in general provided by the integrator).
Initial Offset	The smallest expiry point offset on a schedule table. This can be zero.	
Interarrival Time	Basic Tasks The time between successively entering the `READY` state from the `SUSPENDED` state. Activation of a task always represents a new arrival. This applies in the case of multiple activations, even if an existing instance of the task is in the `RUNNING` or `READY` state. Extended Tasks: The time between successively entering the `READY` state from the `SUSPENDED` or `WAITING` states. Setting an event for a task in the `WAITING` state represents a new arrival if the task is waiting on the event. Waiting for an event in the `RUNNING` state which is already set represents a new arrival. OsIsrs: The time between successive occurrences of an interrupt.. See Figure 2:1.	
Interrupt Lock Time	The time for which a Task/OsIsr executes with Category 1 interrupts disabled/suspended and/or Category 2 interrupts disabled/suspend .	
Interrupt Source Enable	The switch which enables a specific interrupt source in the hardware.	
Interrupt Vector Table	Conceptually, the interrupt vector table contains the mapping from hardware interrupt requests to (software) interrupt service routines. The real content of the Interrupt Vector Table is very hardware specific, e.g. it can contain the start addresses of the interrupt service routines.	
Final Delay	The difference between the final expiry point offset and the duration on a schedule table in ticks. This value defines the delay from the final expiry point to the logical end of the schedule table for single-shot and “nexted” schedule tables. Final Delay = Final Expiry Point Delay if Initial Offset = 0	
Forced OS-Application Termination	The operating system frees all system objects, e.g. forcibly terminates Tasks, disables interrupts, etc., which are associated to the OS-Application. OS-Application and internal variables are potentially left in an undefined state.	
Forced Termination	The OS terminates the Task/Category 2 OsIsr and does ”unlock” its held resources. For details see OS108 and OS109.	
Linker File	File containing linking settings for the linker. The syntax of the linker file depends on the specific linker and, consequently, definitions are stored “linker-specific” in the linker file.	

Lock Budget	Maximum permitted Interrupt Lock Time or Resource Lock Time.	
Memory Protection Unit	A Memory Protection Unit (MPU) enables memory partitioning with individual protection attributes. This is distinct from a Memory Management Unit (MMU) that provides a mapping between virtual addresses and physical memory locations at runtime. Note that some devices may realise the functionality of an MPU in an MMU.	
Mode	Describes the permissions available on a processor.	
	Privileged	In general, in »privileged mode« unrestricted access is available to memory as well as the underlying hardware.
	Non-privileged	In »non-privileged mode« access is restricted.
Modulus	The number of ticks required to complete a full wrap of an OSEK counter. This is equal to `OsCounterMaxAllowedValue` +1 ticks of the counter.	
OS-Application	A collection of OS objects	
	Trusted	An OS-Application that is executed in privileged mode and has unrestricted access to the API and hardware resources. Only trusted applications can provide trusted functions.
	Non-trusted	An OS-Application that is executed in non-privileged mode has restricted access to the API and hardware resources.
OS object	Object that belongs to a single OS-Application: Task, OsIsr, Alarm, Event, Schedule Table, Resource, Trusted Function, Counter, Applicaton-specific hook.	
OS Service	OS services are the API of the operating system.	
Protection Error	Systematic error in the software of an OS-Application.	
	Memory access violation	A protection error caused by access to an address in a manner for which no access right exists.
	Timing fault	A protection error that violates the timing protection.
	Illegal service	A protection error that violates the service protection, e.g. unauthorized call to OS service.
	Hardware exception	division by zero, illegal instruction etc.
Resource Lock Time	The time an OSEK resource is held by a Task/OsIsr (excluding the preemptions of the Task/OsIsr by higher prior Tasks/OsIsrs).	
Response Time	The time between a Task/OsIsr being made ready to execute and generating a specified response. The time includes all preemptions. Figure	
Restart an OS-Application	An OS-Application can be restarted after self-termination or being forcibly terminated because of a protection error. When an OS-Application is restarted, the OS activates the configured `OsRestartTask`.	
Scalability Class	The features of the OS (e.g. Memory Protection or Timing Protection), described by this document, can be grouped together to customize the operating system to the needs of the application. There are 4 defined groups of features which are named scalability classes. For details see Chapter 7.9	
Schedule Table	Encapsulation of a statically defined set of expiry points.	
Section	Part of an object file in which instructions or data are combined to form a unit (contiguous address space in memory allocated for data or code). A section in an object file (object file format) has a name and a size. From the linker perspective, two different sides can be distinguished:	
	input section	memory section in an input object file of the linker.
	Output section	memory section in an output object file of the linker.
Set (of OS objects)	This document uses the term set, indicating a collection of the same type of OS objects, in the strict mathematical sense, i.e.: - a set contains zero or more OS objects (this means a set can be empty) - the OS objects in the set are unique (this means there cannot be duplicate OS objects in the set)	
Symbol	Address label that can be imported/used by software modules and resolved by the linker. The precise syntax of the labels is linker-specific. Here, these address labels are used to identify the start and end of memory sections.	
	Start symbol	Tags the start of a memory section
	End symbol	Tags the end of a memory section

Synchronization of schedule tables with a synchronization counter	Synchronization with a synchronization counter is achieved, if the expiry points of the schedule table are processed within an absolute deviation from the synchronization counter that is smaller than or equal to a precision threshold.
Synchronization Counter	The “Synchronization Counter”, distinct from an OS counter object, is an external counter, external to the OS, against which expiry points of a schedule table are synchronized
Time Frame	The minmum inter-arrival time for a Task/OsIsr.
Trusted Function	A service provided by a trusted OS-Application that can be used by other OS-Applications (trusted or non-trusted).
Worst case execution time (WCET)	The longest possible execution time.
Write access	Storing a value in a register or memory location. All memory accesses that have the consequence of writing (e.g. reads that have the side effect of writing to a memory location) are treated as write accesses.

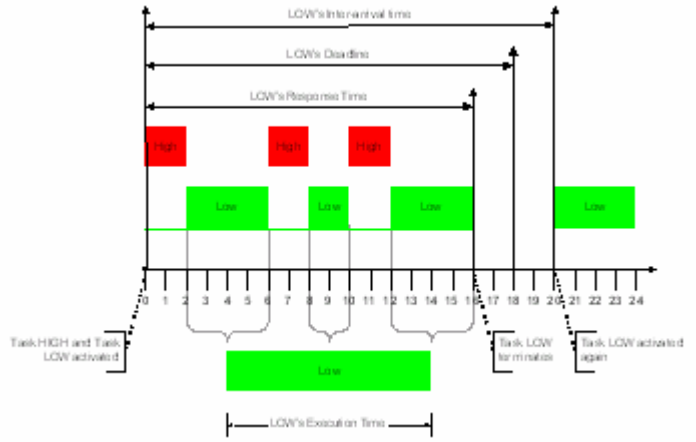


Figure 2:1: Definition of Timing Terminology

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] Requirements on Operating System
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_OS.pdf

[3] General Requirements on Basic Software Modules
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_General.pdf

[4] Specification of the Virtual Functional Bus
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_Spec_of_VFB.pdf

[5] Requirements on Software FreeRunningTimer
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_SWFreeRunnningTimer.pdf

[6] Specification of GPT Driver
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_GPT_Driver.pdf

[7] Specification of Standard Types
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_StandardTypes.pdf

[8] Specification of Memory Mapping
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_MemoryMapping.pdf

[9] Specification of RTE
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_RTE.pdf

[10] AUTOSAR ECU Configuration Specification
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_ECU_Configuration.pdf

[11] AUTOSAR Basic Software Module Description Template,
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

3.2.1 OSEK/VDX

The OSEK/VDX specifications are publically available from www.osek-vdx.org

[12] Operating System
Version 2.2.3
17th February 2005

[13] Time-Triggered Operating System
Version 1.0
24th July 2001

[14] System Generation OIL: OSEK Implementation Language
Version 2.5
1st July 2004

[15] OSEK RunTime Interface (ORTI) Part A: Language Specification
Version 2.2
14th November 2005

[16] OSEK Run Time Interface (ORTI) Part B: OSEK Objects and Attributes
Version 2.2
25th November 2005

[17] Binding Specification
Version 1.4.2
15th July 2004

3.2.2 HIS

The HIS (Hersteller Initiative Software) documents are publicly available from www.automotive-his.de

[18] Requirements for Protected Applications under OSEK
Version 1
25th September 2002.

[19] OSEK OS Extensions for Protected Applications
Version 1.0
27th July 2003

3.2.3 ISO/IEC

[20] ISO/IEC 9899:1990 Programming Language – C

(Remark: The international ISO standard ISO/IEC 9899:1990, also sometimes simply called »C90«, describes the language C. It was introduced in 1990 and replaced the ANSI C standard that was introduced only one year before, that's why it is also called »C89«. C89 differs from ISO/IEC 9899:1990 essentially only by the copyright note.)

[21] ISO/IEC 9899:1999 Programming Language – C

(Remark: A revised version of the standard was published in 1999. It is officially ISO/IEC 9899:1999, but is more often referred to as »C99«.)

3.3 Company Reports, Academic Work, etc.

[22] Extensions of OSEK OS for Protected Applications
OSEK Support Project DC058_02
DaimlerChrysler AG

4 Constraints and assumptions

4.1 Existing Standards

This document makes the following assumptions about the referenced related standards and norms:

- OSEK OS [12] provides a sufficiently flexible scheduling policy to schedule AUTOSAR systems.
- OSEK OS [12] is a mature specification and implementations are used in millions of ECUs worldwide.
- OSEK OS [12] does not provide sufficient support for isolating multi-source software components at runtime.
- OSEK OS [12] does not provide sufficient runtime support for demonstrating the absence of some classes of fault propagation in a safety-case.
- OSEKtime OS [13] and the HIS Protected OSEK [19] are immature specifications that contain concepts necessary for AUTOSAR and satisfy specific application domains. It is the purpose of this document to identify these needs and to recommend the use of parts (or all) of these specifications as appropriate.

4.2 Terminology

The specification uses the following operators when requirements specify multiple terms:

NOT : negation of a single term e.g. NOT Weekend
AND : conjunction of two terms e.g. Weekend AND Saturday
OR : disjunction of two terms e.g. Monday OR Tuesday

A requirement comprising multiple terms is evaluated left to right.

The precedence rules are:

Highest Precedence	**NOT**
Lowest Precedence	**AND OR**

The expression NOT X AND Y means (NOT X) AND (Y)

Where operators of the same precedence are used in the same sentence, commas are used to disambiguate. The expression X AND Y, OR Z means (X AND Y) OR Z.

4.3 Interaction with the RTE

The configuration of an AUTOSAR system [4] maps the »runnables« of a »software component« to (one or more) tasks that are scheduled by the operating system. All runnables in a task share the same protection boundary. In AUTOSAR, a software component must not include an interrupt handler. A software component is therefore implemented as runnables executing within the body of a task, or set of tasks, only.

Runnables get access to hardware-sourced data through the AUTOSAR RTE. The RTE provides the runtime interface between runnables and the basic software modules. The basic software modules also comprise a number of tasks and OsIsrs that are scheduled by the operating system.

It is assumed that the software component templates and the description of the basic software modules provide sufficient information about the required runtime behavior to be able to specify the attributes of tasks required to configure the OS.

4.4 Operating System Abstraction Layer (OSAL)

Systems that do not use the OS defined in AUTOSAR can provide a platform for the execution of AUTOSAR software components using an Operating System Abstraction Layer. The interface to the OSAL is exactly that defined for the AUTOSAR OS.

4.5 Limitations

4.5.1 Hardware

The core AUTOSAR operating system assumes free access to hardware resources, which are managed by the OS itself. This includes, but is not limited to, the following hardware:

- interrupt control registers
- processor status words
- stack pointer(s)

Specific (extended) features of the core operating system extend the requirements on hardware resource. The following list outlines the features that have requirements on the hardware. Systems that do not use these OS features do not have these hardware requirements.

- Memory Protection: A hardware memory protection unit is required. All memory accesses that have the consequence of writing (e.g. reads that have the side effect of writing to a memory location) shall be treated as writes.

- Time Protection: Timer Hardware for monitoring execution times and arrival rates.

- »Privileged« and »non-privileged« modes on the MCU: to protect the OS against internal corruption caused by writes to OS controlled registers. This mode must not allow OS-Applications to circumvent protection (e.g. write registers which govern memory protection, write to processor status word etc.). The privileged mode must be under full control of the protected OS which uses the mode internally and to transfer control back and forth from a non-trusted OS-Application to a trusted OS-Application. The microprocessor must support a controlled means which moves a processor into this privileged mode.

- Local/Global Time Synchronization: A global time source is needed.

In general hardware failures in the processor are not detected by the operating system. In the event of hardware failure, correct operation of the OS cannot be guaranteed.

The resources managed by a specific OS implementation have to be defined within the appropriate configuration file of the OS.

4.5.2 Programming Language

The API of the operating system is defined as C89 [20] function calls or macros. If other languages are used they must adapt to the C interface. This is because C99 [21] allows for internal dynamic memory allocation during subroutine calls. Most automotive applications are static (non-heap based).

4.5.3 Miscellaneous

The operating system does not provide services for dynamic memory management.

The operating system is only able to handle a single thread of execution at one time. It is therefore not able to manage software running on a multi-processor system. A multi-processor system must use a different OS image for each processor.

4.6 Applicability to car domains

The operating system has the same design constraints regarding size and scalability under which the OSEK OS was designed. The immediate domain of applicability is therefore currently body, chassis and power train ECUs. However, there is no reason that the OS cannot be used to implement ECUs for infotainment applications.

5 Dependencies to other modules

There are no forced dependencies on other modules, however:

- It is assumed that the operating system may use timer units (e.g. a GPT channel) to drive counters.
- If the user needs to drive scheduling directly from global time, then a global time interrupt is required.
- If the user needs to synchronize the processing of a schedule table to a global time, the operating system needs to be told the global time using the `SyncScheduleTable()` service.

5.1 File structure

5.1.1 Code file structure

The code file structure of the Operating system is not fixed, besides the requirements in the General SRS.

5.1.2 Header file structure

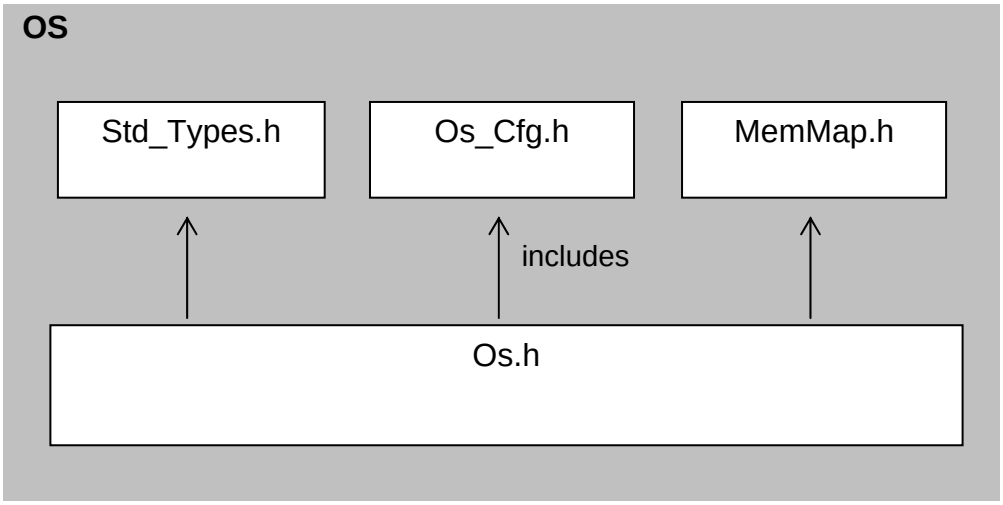


Figure 5:1: Header File Structure for the OS

The figure above contains the defined AUTOSAR header file hierarchy of the Operating System.

Users of the Operating Systems shall only include the **Os.h** file.

If an implementation of the Operating System requires additional header files, it is free to include them. The header files are self contained, that means they will include

all other header files which are required by them (e.g include files for the GPT driver if required).

6 Requirements Traceability

This chapter contains references to requirements of other AUTOSAR documents.

6.1 General Requirements on Basic Software Modules

Requirement	*Satisfied by*
Functional Requirements - Configuration	
[BSW00344] Reference to link time configuration	Not applicable (AUTOSAR OS is a statically configured Operating System.)
[BSW00404] Reference to post build time configuration	Not applicable (AUTOSAR OS does not support post build time configuration.)
[BSW00405] Reference to multiple configuration sets	Not applicable (AUTOSAR OS does not support post build time configuration.)
[BSW159] Tool-based configuration	OS uses standard XML, so various tools may be used to configure the OS
[BSW167] Static configuration checking	See Chapter 11.2
[BSW171] Configurability of optional functionality	See Chapter 10. Requirement focuses on implementation.
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable
[BSW00380] Separate C-Files for configuration parameters	Not applicable (Requirement for implementation)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Not applicable (Requirement for implementation)
[BSW00381] Separate configuration header file for pre-compile time parameters	Not applicable (Requirement for implementation)
[BSW00412] Separate H-File for configuration parameters	Not applicable (Requirement for implementation)
[BSW00383] List dependencies of configuration files	Not applicable (SWS has no dependencies for configuration files.)
[BSW00384] List dependencies to other modules	Not applicable (SWS has no dependencies to other modules.)
[BSW00387] Specify the configuration class of callback function	See Chapter 8.5 for details.
[BSW00388] Introduce containers	See Chapter 10.2.
[BSW00389] Containers shall have names	See Chapter 10.2.
[BSW00390] Parameter content shall be unique within the module	See Chapter 10.2.
[BSW00391] Parameter shall have unique names	See Chapter 10.2.
[BSW00392] Parameters shall have a type	See Chapter 10.2.
[BSW00393] Parameters shall have a range	See Chapter 10.2.

Requirement	*Satisfied by*
[BSW00394] Specify the scope of the parameters	See Chapter 10.2.
[BSW00395] List the required parameters (per parameter)	See Chapter 10.2.
[BSW00396] Configuration classes	See Chapter 10.2.
[BSW00397] Pre-compile-time parameters	See Chapter 10.2.
[BSW00398] Link-time parameters	See Chapter 10.2.
[BSW00399] Loadable Post-build time parameters	See Chapter 10.2.
[BSW00400] Selectable Post-build time parameters	See Chapter 10.2.
[BSW00438] Post-build configuration data structure	See Chapter 10.2.
[BSW00402] Published information	See Chapter 10.2.
Functional Requirements - Wake Up	
[BSW00375] Notification of wake-up reason	Not applicable
Functional Requirements - Initialization	
[BSW101] Initialization interface	`StartOS()`
[BSW00416] Sequence of Initialization	`StartOS()` is called by the user. Sequence of calls with other modules is not affected.
[BSW00406] Check module initialization	Not applicable (Requirement for implementation)
[BSW00437] NoInit—Area in RAM	N/A – The OS only covers it's own data areas.
Functional Requirements - Normal Operation	
[BSW168] Diagnostic interface of SW components	Not applicable
[BSW00407] Function to read out published parameters	Not applicable (Requirement for implementation)
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interface	Not applicable (AUTOSAR OS does not interact directly with SW-C.)
[BSW00424] BSW main processing function task allocation	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00425] Trigger conditions for schedulable objects	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00426] Exclusive areas in BSW modules	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00427] OsIsr description for BSW modules	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00428] Execution order dependencies of main processing functions	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00429] Restricted BSW OS functionality access	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00431] The BSW Scheduler module implements task bodies	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data	Requirement for users of AUTOSAR OS together with the RTE.

Requirement	*Satisfied by*
path	
[BSW00433] Calling of main processing functions	Requirement for users of AUTOSAR OS together with the RTE.
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Requirement for users of AUTOSAR OS together with the RTE.
Functional Requirements - Shutdown Operation	
[BSW00336] Shutdown Interface	`ShutdownOS()`
Functional Requirements - Fault Operation and Error Detection	
[BSW00337] Classification of errors	Not applicable (AUTOSAR OS does not distinguish between “development” and “production” errors. See Section 7.11.
[BSW00338] Detection and Reporting of development errors	Not applicable (AUTOSAR OS calls the ErrorHook (defined by the OSEK OS specification [12]) and the ProtectionHook (see Section 7.8) in case of errors. It is possible to call Debug Error Tracer from these hook routines.)
[BSW00369] Do not return development error codes via API	Not applicable (AUTOSAR OS does not distinguish between “development” and “production” errors. In accordance with OSEK OS all possible errors are reported via the `ErrorHook()` and as return values of system services.)
[BSW00339] Reporting of production relevant error status	Not applicable (AUTOSAR OS calls the `ErrorHook()` (defined by the OSEK OS specification [12]) and the `ProtectionHook()` (see Section 7.8) in case of errors. It is possible to call the function inhibition or diagnostic event manager (DEM) and handle the debouncing from these hook routines.)
[BSW00422] Debouncing of production relevant error status	Not applicable (AUTOSAR OS calls the `ErrorHook()` (defined by the OSEK OS specification [12]) and the `ProtectionHook()` (see Section 7.8) in case of errors. It is possible to call the function inhibition or diagnostic event manager (DEM) and handle the debouncing from these hook routines.)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (e.g. this module does not provide any wake-up reason)
[BSW00323] API parameter checking	See Section 7.7.3
[BSW004] Version check	Not applicable (Requirement for implementation)
[BSW00409] Header files for production code error IDs	Not applicable (AUTOSAR OS does not distinguish between “development” and “production” errors. See Section 7.11.
[BSW00385] List possible error notifications	Not applicable (AUTOSAR OS does not distinguish between “development” and “production” errors. See Section 7.11.
[BSW00386] Configuration for detecting an error	Not applicable (AUTOSAR OS calls the `ErrorHook()` (defined

Requirement	*Satisfied by*
	by the OSEK OS specification [12]) and the `ProtectionHook()` (see Section 7.8) in case of errors. It is possible to call the function inhibition or diagnostic event manager (DEM) and handle the debouncing from these hook routines.)
Non-functional Requirements - Software Architecture Requirements	
[BSW161] Microcontroller Abstraction	N/A
[BSW162] ECU Layout Abstraction	N/A
[BSW005] No hard coded horizontal interfaces within MCAL	OS does not belong to MCAL
[BSW00415] User dependant include files	N/A – OS is used by several BSWs and SWCs
Non-functional Requirements - Software Integration Requirements	
[BSW164] Implementation of interrupt service routines	OK – OS is allowed to implement interrupt service routines
[BSW00325] Runtime of interrupt service routines	N/A
[BSW00326] Transition of OsIsrs to OS tasks	N/A
[BSW00342] Usage of source code and object code	N/A
[BSW00343] Specification and configuration of time	OS supports ticks for OSEK compatibility and physical times (with convert macros)
[BSW160] Human-readable configuration data	OS is using XML
Non-functional Requirements - Software Documentation Requirements	
[BSW009] User Module Documentation	N/A
[BSW00401] Documentation of multiple instance of configuration parameters	N/A
[BSW172] Compatibility and documentation of scheduling strategy	N/A
[BSW010] Memory resource documentation	N/A
[BSW00333] Documentation of callback function context	N/A
[BSW00374] Module vendor identification	N/A
[BSW00379] Module identification	N/A
[BSW003] Version identification	N/A
[BSW00318] Format of module version numbers	N/A
[BSW00321] Enumeration of module version numbers	N/A
[BSW00341] Microcontroller compatibility documentation	N/A
[BSW00334] Provision of XML file	N/A

6.2 Requirements on Software Free-Running Timer

Requirement	*Satisfied by*
[SWFRT00019] Configure HW Timer Type	OS390
[SWFRT00020] Configuration/initialization of HW Timer	OS371, OS374
[SWFRT00021] Import Used HW Timer's Configuration	GPT is referenced by OS configuration (via XML) See chapter 10.2 for details.
[SWFRT00022] State which HW Timer is used	OS370
[SWFRT00023] Set up Duration of one Tick	OS385
[SWFRT00024] Support different Ranges/Resolutions	OS375
[SWFRT00025] Set up Access Methods	OS383, OS392
[SWFRT00026] Set up Target Count Values	OS386
[SWFRT00028] Ensure Continuous Running Mode	OS441
[SWFRT00029] Init Function	The init function of the OS is the `StartOS()` service. If the GPT driver is used by the OS, the init function of the GPT driver has to be called before starting the OS via `StartOS()`.
[SWFRT00030] Start with Zero	OS384
[SWFRT00031] Increment Counter	OS384
[SWFRT00032] Wrap Around	N/A. This requirement targets specific timer features. The OS should minimize the software access to timers (e.g. by using automatic reload features of the hardware).
[SWFRT00033] Read out Ticks	OS377
[SWFRT00034] Calculate Ticks Elapsed since given value	OS382
[SWFRT00041] Shutdown Function	There is no function to shutdown the timers which drive counters
[SWFRT00047] Convert Ticks to Time	OS393
[SWFRT00048] EcuM Modes	The OS is not aware of any EcuM modes and switching modes which influences timers (e.g. stop them or slow down counting) is neither recognized nor handled by the OS. It is the responsibility of the user to take care of this affects.

6.3 AUTOSAR SRS OS Requirements

Requirement	*Satisfied by*
[BSW097] Existing OSEK OS	OS001
[BSW11001] Object Grouping	OS114, OS056

Requirement	*Satisfied by*
[BSW098] Table based schedules	OS002, OS007
[BSW099] Switchable schedules	OS191
[BSW11002] Synchronization with global time	OS206, OS200, OS201, OS013, OS199, OS260, OS227
[BSW11003] Stack Monitoring	OS067, OS068
[BSW11005] Memory Write Access	OS207, OS208, OS195
[BSW11006] Data exchange	OS086, OS196, OS087
[BSW11007] Code Sharing	OS081
[BSW11000] Memory read access	OS026
[BSW11008] Timing Protection	OS028, OS089, OS033, OS037, OS048, OS064, OS465, OS469, OS470, OS471, OS472, OS473, OS474
[BSW11009] Protection of the OS	OS051, OS088, OS052, OS069, OS070, OS092, OS093
[BSW11010] Protection of OS-Applications	OS056
[BSW11011] Protecting the OS managed hardware	OS096, OS245
[BSW11012] Scalable Protection	OS241, OS240
[BSW11016] Scalability of the OS	OS241, OS240
[BSW11013] Error Notification	OS068, OS044, OS210, OS033, OS037, OS064, OS051, OS088, OS070, OS093, OS056, OS246
[BSW11014] Protection Error Handling	OS033, OS037, OS106, OS107, OS108, OS109, OS110, OS243, OS244,
[BSW11018] Interrupt services	OS299
[BSW11020] Interface for ticking counters	OS286
[BSW11021] Cascading counters	OS301
[BSW11019] Creation of Interrupt Vector Table	OS336

6.4 AUTOSAR SWS Service Requirements to API

Requirement	*Associated API*
OS016	8.4.1
OS097	8.4.3
OS358	8.4.9
OS347	8.4.8
OS006	8.4.10
OS191	8.4.11
OS012	8.4.16
OS199	8.4.13, 8.4.14
OS227, OS359	8.4.15

Requirement	*Associated API*
OS099	8.4.4,8.4.5,8.4.2
OS256	8.4.6
OS017	8.4.7
OS258	8.4.19
OS383, OS392	8.4.17, 8.4.18
OS201	8.4.12

7 Functional specification

7.1 Core OS

7.1.1 Background & Rationale

The OSEK/VDX Operating System [12] is widely used in the automotive industry and has been proven in use in all classes of ECUs found in modern vehicles. The concepts that OSEK OS has introduced are widely understood and the automotive industry has many years of collective experience in engineering OSEK OS based systems.

OSEK OS is an event-triggered operating system. This provides high flexibility in the design and maintenance of AUTOSAR based systems. Event triggering gives freedom for the selection of the events to drive scheduling at runtime, for example angular rotation, local time source, global time source, error occurrence etc.

For these reasons the core functionality of the AUTOSAR OS shall be based upon the OSEK OS. In particular OSEK OS provides the following features to support concepts in AUTOSAR:

- fixed priority-based scheduling
- facilities for handling interrupts
- only interrupts with higher priority than tasks
- some protection against incorrect use of OS services
- a startup interface through `StartOS()` and the `StartupHook()`
- a shutdown interface through `ShutdownOS()` and the `ShutdownHook()`

OSEK OS provides many features in addition to these. Readers should consult the OSEK specification [12] for details.

Basing AUTOSAR OS on OSEK OS means that legacy applications will be backward compatible – i.e. applications written for OSEK OS will run on AUTOSAR OS. However, some of the features introduced by AUTOSAR OS require restrictions on the use of existing OSEK OS features or extend existing OSEK OS features.

7.1.2 Requirements

OS001: The Operating System shall provide an API that is backward compatible with the OSEK OS API [12].

7.1.2.1 Restrictions on OSEK OS

It is too inefficient to achieve timing and memory protection for alarm callbacks. They are therefore not allowed in specific scalability classes (OS242)

OS242: The Operating System shall only allow Alarm Callbacks in Scalability Class 1.

OSEK OS is required to provide functionality to handle inter-task (internal) communication according to the OSEK COM specification when internal communication only is required in the system. In AUTOSAR, internal communication is provided by the AUTOSAR RTE or by AUTOSAR COM at least one of which will be present for all AUTOSAR ECUs.

AUTOSAR OS, when used in an AUTOSAR system, therefore does not need to support internal communication.

An OSEK OS must implement internal communication if the symbol `LOCALMESSAGESONLY` is defined. AUTOSAR OS can deprecate the need to implement OSEK COM functionality and maintain compatibility with OSEK suite of specifications by ensuring that AUTOSAR OS always exists in an environment where `LOCALMESSAGESONLY` is undefined. This leads to the following requirement:

OS398: The OS shall not define the symbol `LOCALMESSAGESONLY`

7.1.2.2 Undefined Behaviour in OSEK OS

There are a number of cases where the behaviour of OSEK OS is undefined. These cases represent a barrier to portability. AUTOSAR OS tightens the OSEK OS specification by defining the required behaviour.

OS304: If in a call to `SetRelAlarm()` the parameter “increment” is set to zero, the service shall return `E_OS_VALUE` in standard and extended status .

OS424: The first call to `StartOS()` (for starting the Operating System) shall not return.

OS425: If `ShutdownOS()` is called and `ShutdownHook()` returns then the operating system shall disable all interrupts and enter an endless loop.

7.1.2.3 Extensions to OSEK OS

OS299:The Operating System shall provide the services `DisableAllInterrupts()`, `EnableAllInterrupts()`, `SuspendAllInterrupts()`, `ResumeAllInterrupts()` prior to calling `StartOS()` and after calling `ShutdownOS()`. (It is assumed that the static variables of these functions are initialized).

OS301: The Operating System shall provide the ability to increment a software counter as an alternative action on alarm expiry.

OS399: The OS shall provide an API call to increment a software counter.

7.2 Software Free Running Timer

Due to the fact that the number of timers is often very limited, some functionality and configuration is added to extend the reuse of timers. E.g. this allows timer measurments and the integration of GPT drivers. For more details see also [5] (SWFRT).

OS374: The Operating System shall handle all the initialization and configuration of timers used directly by the OS and not handled by the GPT driver.

OS383: The Operating System shall provide a service to read the current count value of a counter (returning either the hardware timer ticks if counter is driven by hardware or the software ticks when user drives counter).

OS392: The Operating System shall provide a service to get the number of ticks between the current tick value and a previously read tick value.

OS384: The Operating System shall adjust the read out values of hardware timers (which drive counters) in such that the lowest value is zero and consecutive reads return an increasing count value until the timer wraps at its modulus.

7.3 Schedule Tables

7.3.1 Background & Rationale

It is possible to implement a statically defined task activation mechanism using an OSEK counter and a series of auto started alarms. In the simple case, this can be achieved by specifying that the alarms are not modified once started. Run-time modifications can only be made if relative synchronization between alarms can be guaranteed. This typically means modifying the alarms while associated counter tick interrupts are disabled.

Schedule Tables address the synchronization issue by providing an encapsulation of a statically defined set of expiry points. Each expiry point defines:

- one or more actions that must occur when it is processed where an action is the activation of a task or the setting of an event.
- An offset in ticks from the start of the schedule table

Each schedule table has a duration in ticks. The duration is measured from zero and defines the modulus of the schedule table.

At runtime, the OS will iterate over the schedule table, processing each expiry point in turn. The iteration is driven by an OSEK counter. It therefore follows that the properties of the counter have an impact on what is possible to configure on the schedule table.

7.3.2 Requirements

7.3.2.1 Structure of a Schedule Table

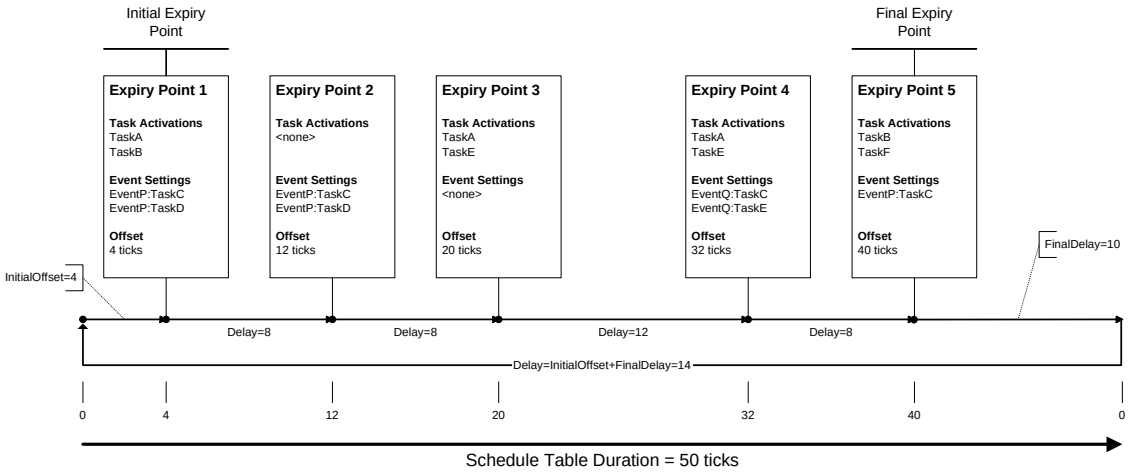


Figure 7.1: Anatomy of a Schedule Table

OS401: A schedule table shall have at least one expiry point.

OS402: An expiry point shall contain a (possibly empty) set of tasks to activate.

OS403: An expiry point shall contain a (possibly empty) set of events to set.

OS404: An expiry point shall contain an offset in ticks from the start of the schedule table.

7.3.2.2 Constraints on Expiry Points

There is no use case for an empty expiry point, so each one must define at least one action.

OS407: An expiry point shall activate at least one task OR set at least one event

The OS needs to know the order in which expiry points are processed. It is therefore necessary to ensure that the expiry points on a schedule table can be totally ordered. This is guaranteed by forcing each expiry point on a schedule table to have a unique offset.

OS442: : Each expiry point on a given schedule table shall have a unique offset.

Iteration over expiry points on a schedule table is driven by an OSEK counter. The characteristics of the counter - `OsCounterMinCycle` and `OsCounterMaxAllowedValue` - place constraints on expiry point offsets.

OS443: The Initial Offset shall be zero OR in the range `OsCounterMinCycle` .. `OsCounterMaxAllowedValue` of the underlying counter.

Simlarly, constraints apply to the delays between of adjacent expiry points and the delay to the logical end of the schedule table.

OS408: The delay between adjacent expiry points shall be in the range `OsCounterMinCycle` .. `OsCounterMaxAllowedValue` of the underlying counter.

OS444: The value of Final Delay shall be in the range `OsCounterMinCycle` .. `OsCounterMaxAllowedValue` of the underlying counter.

7.3.2.3 Processing Schedule Tables

OS002: The OS shall drive an iterator over schedule table expiry points, processing each expiry point from the InitialExpiryPoint to the FinalExpiryPoint in order of increasing offset.

OS007: The Operating System shall permit multiple schedule tables to be processed concurrently.

OS409: A schedule table shall be driven by exactly one counter.

OS410: The Operating System shall be able to process at least one schedule table per counter at any given time.

OS411: One tick on the counter shall correspond to one tick on the schedule table.

It is possible to activate a task and set (one or more unique) events for the same task at the same expiry point. The ordering of task activations and event settings performed from the expiry point could lead to different implementations exhibiting different behaviour (for example, activating a suspended task and then setting and event on the task would succeed but if the ordering was reversed then the event setting would fail). To prevent such non-determinism, it is necessary to enforce a strict ordering of actions on the expiry point.

OS412: The OS shall process all task activations on an expiry point first and then set events.

A schedule table always has a defined state and the following figure illustrates the different states (for a non-synchronized schedule table) and the transitions between them.

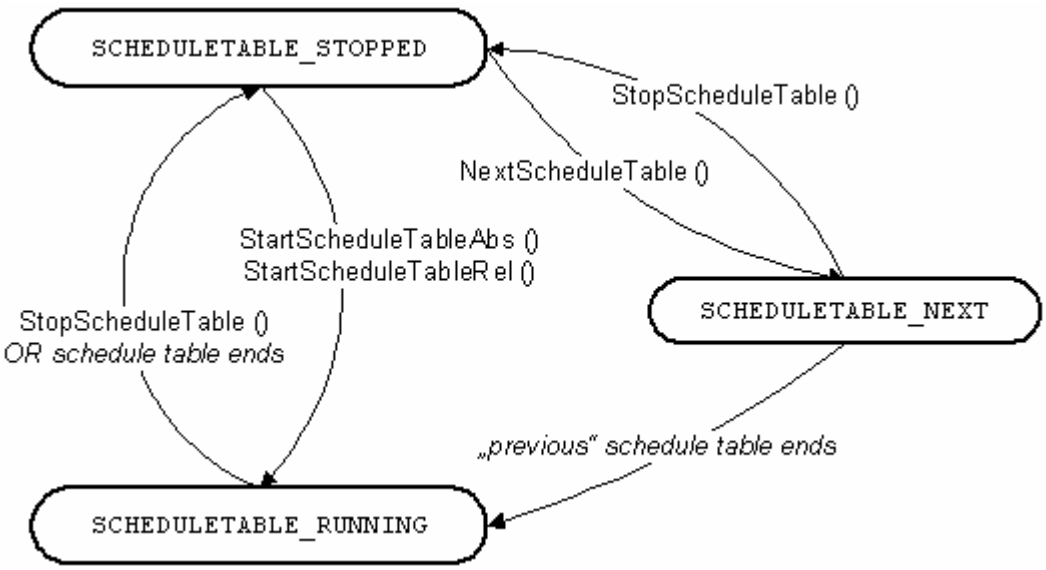


Figure 7.2: States of a schedule table

If a schedule table is not active – this means that is not processed by the Operating System – the state is `SCHEDULETABLE_STOPPED`. After starting a schedule tables enters the `SCHEDULETABLE_RUNNING` state where the OS processes the expiry points. If the service to switch a schedule table is called a schedule table enters the the `SCHEDULETABLE_NEXT` state and waits until the “current” schedule table ends.

7.3.2.4 Repeated Schedule Table Processing

A schedule table may or may not repeat after the final expiry point is processed. This allows two types of behaviour:

1. single-shot – the schedule table processes each expiry point in sequence and then stops at the end . This is useful for triggering a phased sequence of actions in response to some trigger

2. repeating – the schedule table processes each expity point in turn, After processing the final expiry point, it loops back to the initial expirt point. This is useful for building applications that perform repeated processing or system which need to synchronise processing to a driver source.

A repeating schedule table means that each expiry point is repeated at a period equal to the schedule table duration.

OS413: The schedule table shall be configurable as either single-shot or repeating.

OS009: If the schedule table is single-shot, the Operating System shall stop the processing of the schedule table Final Delay ticks after the Final Expiry Point is processed.

OS427: The Operating System shall allow the Final Delay for a single-shot schedule table to be zero.

OS194: If the schedule table is repeating, the Operating System shall process the Initial Expiry Point Final Delay plus Initial Offset ticks have elapsed after processing the Final Expiry Point.
.

7.3.2.5 Controlling Schedule Table Processing

The application is responsible for starting and stopping the processing of a schedule table. There is no implied relationship between the duration of the schedule table and the range of the counter which drives it.

OS358: The Operating System shall provide a service to start the processing of a schedule table at an absolute value “Start” on the underlying counter. (The Initial Expiry Point shall be processed when the value of the underlying counter equals Start + InitialOffset).

OS347: The Operating System shall provide a service to start the processing of a schedule table at “Offset” relative to the “Now” value on the underlying counter (The Initial Expiry Point shall be processed when the value of the underlying counter equals Now + Offset + InitialOffset).

The figure below illustrates the two different methods for a schedule table driven by a counter with a modulus of 65536 (i.e. an `OsCounterMaxAllowedValue = 65535`).

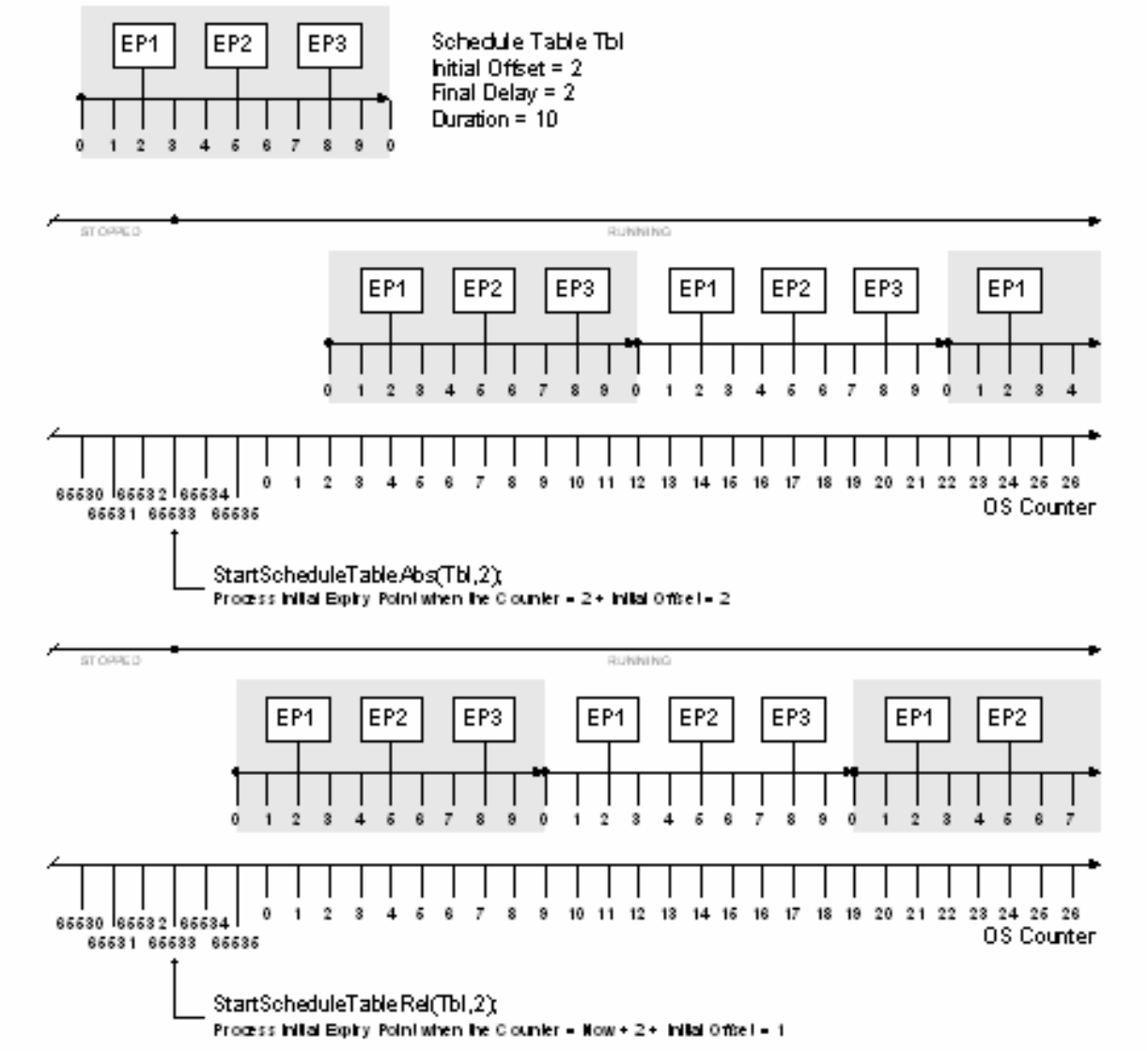


Figure 7.3: Starting a Schedule Table at an Absolute and a Relative Count

OS006: The Operating System shall provide a service to cancel the processing of a schedule table immediately at any point while the schedule table is running.

OS428: If schedule table processing has been cancelled before reaching the Final Expiry Point and is subsequently restarted then OS358/OS347 means that the re-start occurs from the start of the schedule table.

OS191: The Operating System shall provide a service to switch the processing from one schedule table to another schedule table.

OS414: When a schedule table switch is requested, the OS shall continue to process expiry points on the current schedule table up and including the Final Expiry Point

then delay for Final Delay ticks before processing the Initial Expiry Point on the switched-to schedule table (after the initial offset).

OS359: The Operating System shall provide a service to query the state of a schedule table.

7.4 Schedule Table Synchronization

7.4.1 Background & Rationale

The absolute time at which the Initial Expiry Point on a schedule table is processed is under user control. However, if the schedule table repeats then it is not guaranteed that the absolute count value at which the initial expiry point was first processed is the same count value at which it is subsequently processed. This is because the duration of the schedule table need not be equal to the counter modulus.

In many cases it may be important that schedule table expiry points are processed at specific absolute values of the underlying counter. This is called **synchronization**. Typical use-cases include:

- Synchronization of expiry points to degrees of angular rotation for motor management

- Synchronizing the computation to a global (network) time base. Note that in AUTOSAR, the Operating System does not provide a global (network) time source because
  1. a global time may not be needed in many cases
  2. other AUTOSAR modules, most notably FlexRay, provide this independently to the OS
  3. if the OS is required to synchronize to multiple global (network) time sources (for example when building a gateway between two time-triggered networks) the OS cannot be the source of a unique global time.

AUTOSAR OS provides support for synchronization in two ways:

1. implicit synchronization – the counter driving the schedule table is the counter with which synchronization is required. This is typically how synchronization with time-triggered networking technologies (e.g. FlexRay, TTP) is achieved – the underlying hardware manages network time synchronization and simply presents time as an output/compare timer interface to the OS. The following figure shows the possible states for schedule tables with implicit synchronization.

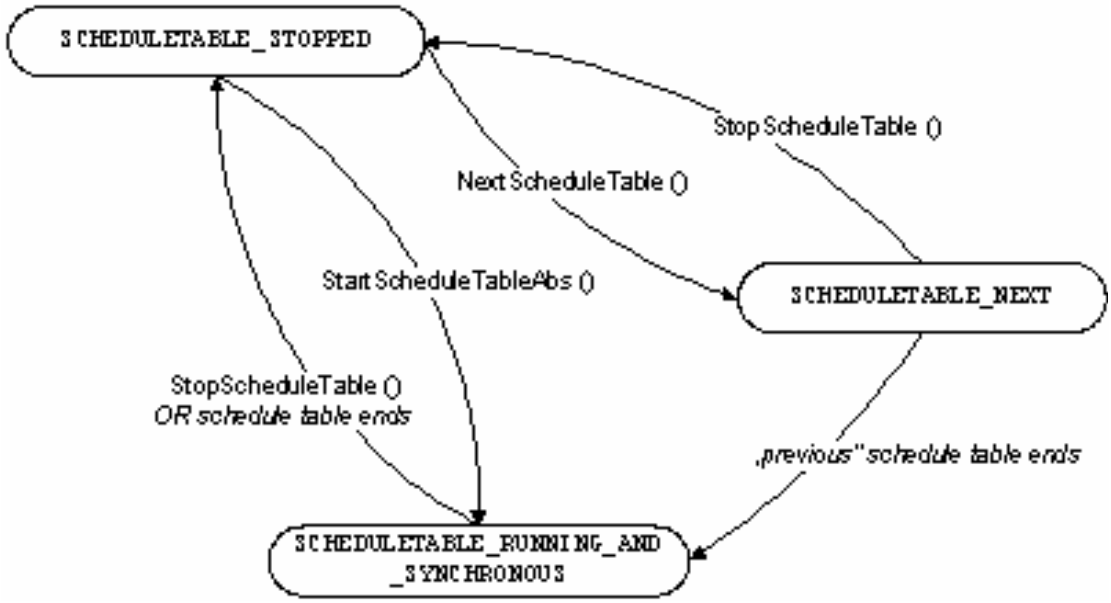


Figure 7.4: States of an implicit synchronized schedule table

2. explicit synchronization – the schedule table is driven by an OS counter which is **not** the counter with which synchronization is required. The OS provides additional functionality to keep schedule table processing driven by the OS counter synchronized with the synchronization counter. This is typically how synchronization with broadcast periodically global times works. The next figure shows the states of such schedule tables.

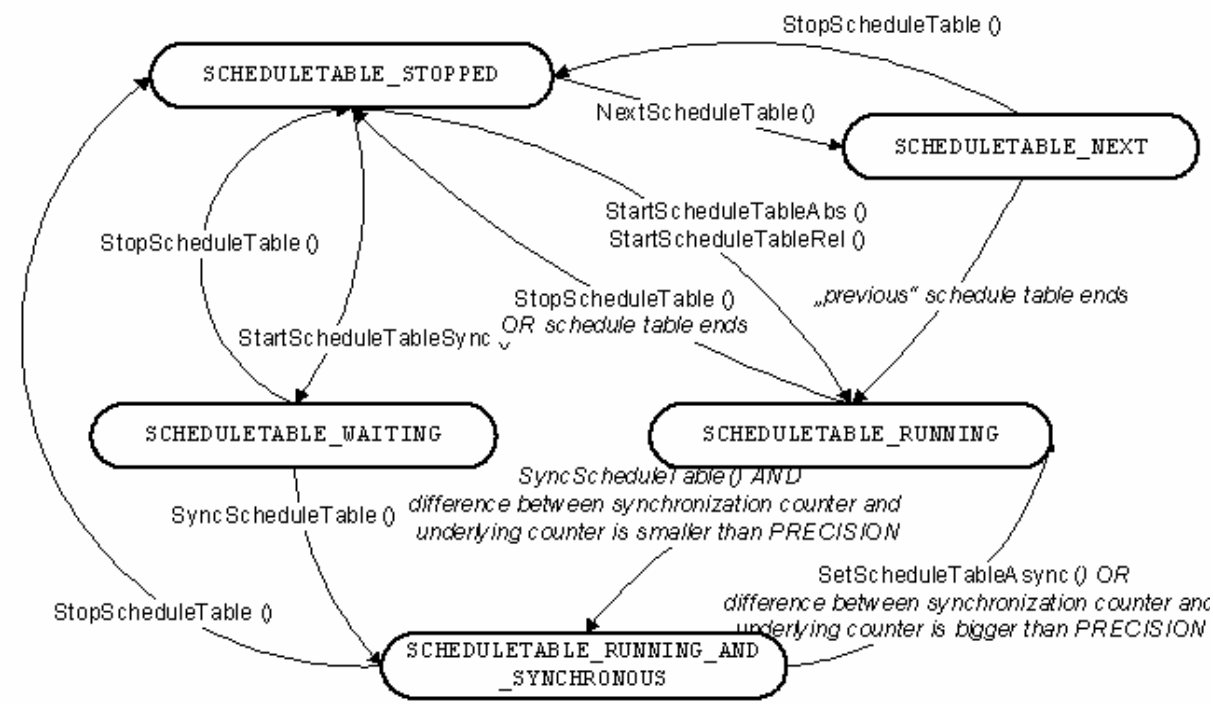


Figure 7.5: States of an explicit synchronized schedule table

7.4.2 Requirements

OS013: The Operating System shall provide the ability to synchronize the processing of schedule table to known counter values.

7.4.2.1 Implicit Synchronization

The OS does not need to provide any additional support for implicit synchronization of schedule tables. However, it is necessary to constrain configuration and runtime control of the schedule table so that ticks on the configured schedule table can be aligned with ticks on the counter. This requires the range of the schedule table to be identical to the range of the counter (the equality of tick resolution of each is guaranteed by the requirements on the schedule table / counter interaction):

OS429: A schedule table that is implicitly synchronized shall have a Duration equal to `OsCounterMaxAllowedValue + 1` of its associated OSEK OS counter.

To synchronize the processing of the schedule table it must be started at a known counter value. The implication of this is that a schedule table requiring implicit synchronization must only be started at an absolute counter value and cannot be started at a relative count value.

OS430: The OS shall prevent a schedule table that is implicitly synchronized from being started at a relative count value.

When the schedule table is started at an absolute counter value each expiry point will be processed when the counter equals the value specified in the service call plus expiry point's offset. The common use-case is to ensure that the offsets specified in the schedule table configuration correspond to absolute values of the underlying counter. This is achieved trivially using `StartScheduleTable(Tbl,0)` as shown below.

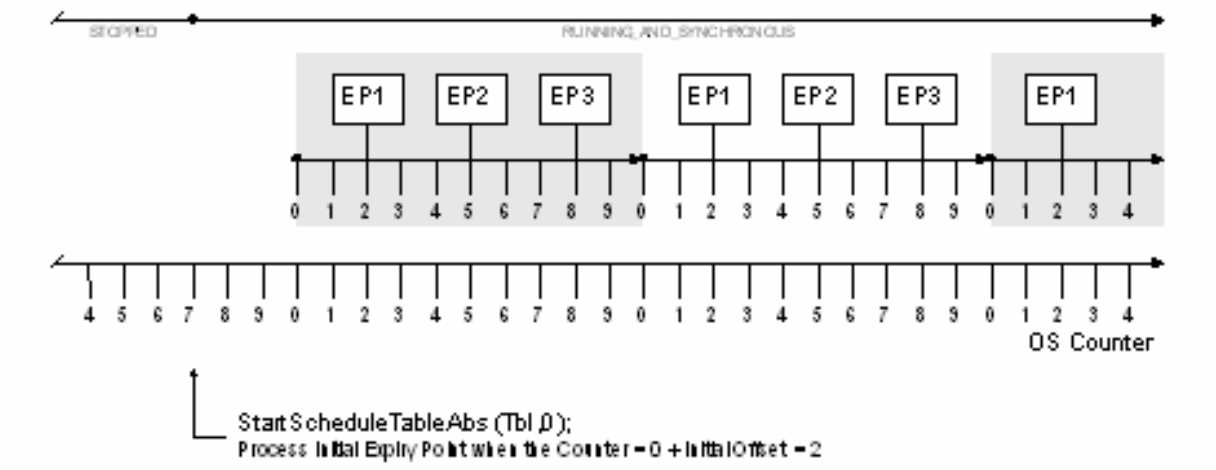


Figure 7.6: Example for implicit synchronized schedule table

7.4.2.2 Explicit Synchonization

An explicitly synchronized schedule table requires additional support from the OS. The schedule table is driven by an OS counter as normal (termed the “drive counter”) but processing needs to be synchronized with a different counter (termed the “synchronization counter”) which is not an OS counter object.

The following constraints must be enforced between the schedule table, the OS counter and the synchronization counter:

Constraint1:

OS431: A schedule table that is explicitly synchronized shall have a duration no greater than modulus of the drive counter.

Constraint2:

OS462: A schedule table that is explicitly synchronized shall have a duration equal to the modulus of the synchronization counter.

Constraint3:

OS463: The synchronization counter shall have the same resolution as the drive counter associated with the schedule table. This means that a tick on the schedule table has the same duration as a tick on the synchronization counter.

Note that it is in the responsibility of the OS user to verify that Constraints 2 and 3 are satisfied by their system.

The function of explicit synchronization is for the OS to keep processing each expiry point at absolute value of the synchronization counter equal to the expiry point’s offset. This means that explicit synchronization always assumes that the notional zero of the schedule table has to be synchronized with absolute value zero on the synchronization counter.

To achieve this, the OS must be told the value of the synchronization counter by the user. As the modulus of the synchronization counter and the schedule table are identical, the OS can use this information to calculate drift. The OS then automatically adjusts the delay between specially configured expiry points, retarding them or advancing them as appropriate, to ensure that synchronization is maintained.

7.4.2.2.1 Startup

There are two options for starting an explicitly synchronized schedule table:

1. Asynchronous start: Start the schedule table at an arbitrary value of the synchronization counter.

2. Synchronous start: Start the schedule table at absolute value zero of the synchronization counter only after a synchronization count has been provided. This may mean waiting for first synchronization indefinitely.

Asynchronous start is provided by the existing absolute and relative schedule table start services. Both of these services set the point at which the initial expiry point is processed with respect to the driver counter not the synchronization counter. This allows the schedule table to start running before the value of the synchronization counter is known.

OS434: An explicitly synchronized schedule table started at an absolute or relative counter value shall have state “running” when the service call returns.

Synchronous start requires an additional service that starts the schedule table only after the OS is told the value of the synchronization counter.

OS201:The Operating system shall provide a service to start an explicitly synchronized schedule table. The Initial Expiry Point will be processed after (Duration – Value) + Initial Offset ticks of the driver counter have elapsed where Value is the absolute value of the synchronization count provided to the schedule table.

OS435: An explicitly synchronized schedule table started synchronously shall have state “waiting” when the service call returns.

7.4.2.2.2 Providing a Synchronization Count

The OS must be told the value of the synchronization counter. Since the schedule table duration is equal to the modulus of the synchronization counter, the OS can use this to determine the drift between the current count value on the schedule table time and the synchronization count and decide whether (or not) any action to achieve synchronization is required.

OS199: The Operating System shall provide a service to provide the schedule table with a synchronization count and start synchronization.

7.4.2.2.3 Specifying Synchronization Bounds

A schedule table defaults to denying adjustment at all expiry points. Adjustment is allowed only when explicitly configured. The range of adjustment that the OS can make at an adjustable expiry point is controlled by specifying:

- `OsScheduleTableMaxRetard` : the maximum value that can be subtracted from the expiry offset
- `OsScheduleTableMaxAdvance`: the maximum value that can be added to the expiry point offset

The following figure illustrates the behaviour of `OsScheduleTableMaxRetard` and `OsScheduleTableMaxAdvance`:

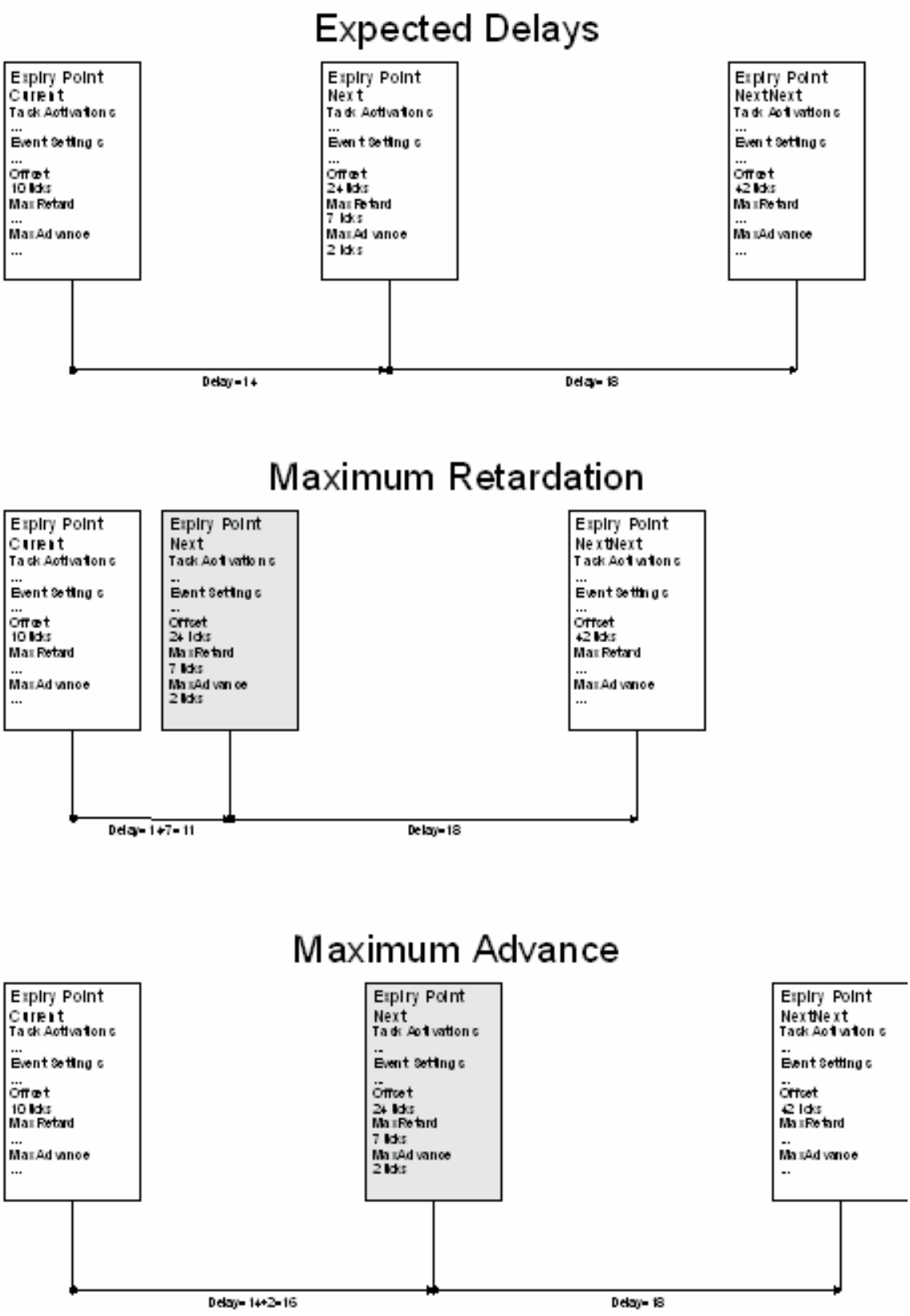


Figure 7.7: Adjustment of Exipry Points

So called “hard” and “smooth” synchronization from OSEKtime [13] are supported by this single unified concept in AUTOSAR OS. “Smooth” synchronization may be emulated by setting the small adjustment values on the final expiry point. “Hard” synchronization may be emulated by setting large adjustment values on the final expiry point.

OS415: An expiry point shall permit the configuration of a OsScheduleTableMaxRetard that defines the maximum number of ticks that can be subtracted from expiry point offset.

OS416: An expiry point shall permit the configuration of a `OsScheduleTableMaxAdvance` that defines the maximum number of ticks that can be added to expiry point offset.

When performing synchrioniszation it is important that the expiry points on the schedule table are processed according to the total ordering defined by their offsets. This means that the range of permitted values for `OsScheduleTableMaxRetard` and `OsScheduleTableMaxAdvance` must ensure that the next expiry point is not retarded into the past or advanced beyond more than one iteration of the schedule table.

OS436: The value of (Offset - `OsScheduleTableMaxRetard`) of an expiry point shall be greater than (Offset + `OsCounterMinCycle`) of the pervious expiry point.

OS437: The value of (Offset+`OsScheduleTableMaxAdvance`) of an expiry point shall be less than the duration of the schedule table.

Explicitly synchronized schedule tables allow the tolerance of some drift between the schedule table value and the synchronization counter value. This tolerance can be zero, indicating that the schedule table is not considered synchronized unless the values are indentical..

OS438: A schedule table shall define a precision bound with a value in the range 0 to duration.

7.4.2.3 Performing Synchronization

The OS uses the synchronization count to support (re-)synchronization of a schedule table at each expiry point by calculating an adjustment to the delay to the next expiry point. This provides faster re-synchronization of the schedule table than doing the action on the final expiry point.

OS206: When a new synchronization count is provided, the Operating System shall calculate the current deviation between the explicitly synchronized scheduled table and the synchronization count.

It is meaningless to try and synchronise an explicitly synchronized schedule table before a synchronization count is provided.

OS417: The OS shall start to synchronise an explicitly synchronized schedule table after a synchronization count is provided.

OS418: The OS shall set the state of an explicitly synchronized schedule table to "running and synchronous" if the absolute value of the deviation between schedule table value the synchronization count is less than the configured `OsScheduleTblExplicitPrecision` threshold.

OS419: The OS shall set the state of an explicitly synchronized schedule table to "running" if the absolute value of the deviation between schedule table value the synchronization count is greater than or equal to the configured `OsScheduleTblExplicitPrecision` threshold.

OS420: If the deviation is negative and the next expiry point is adjustable then the OS shall set the delay to the next expiry point to Delay+min(`OsScheduleTableMaxAdvance`,Deviation)

OS421: If the deviation is positive and the next expiry point is adjustable then the OS shall set the delay to the next expiry point to Delay-min(`OsScheduleTableMaxRetard`, Deviation)

Figure 7.8: shows explicit synchronization of a schedule table. It assumes the following:

- EP1-3 have `OsScheduleTableMaxAdvance`=2
- EP1-3 have `OsScheduleTableMaxRetard` =1

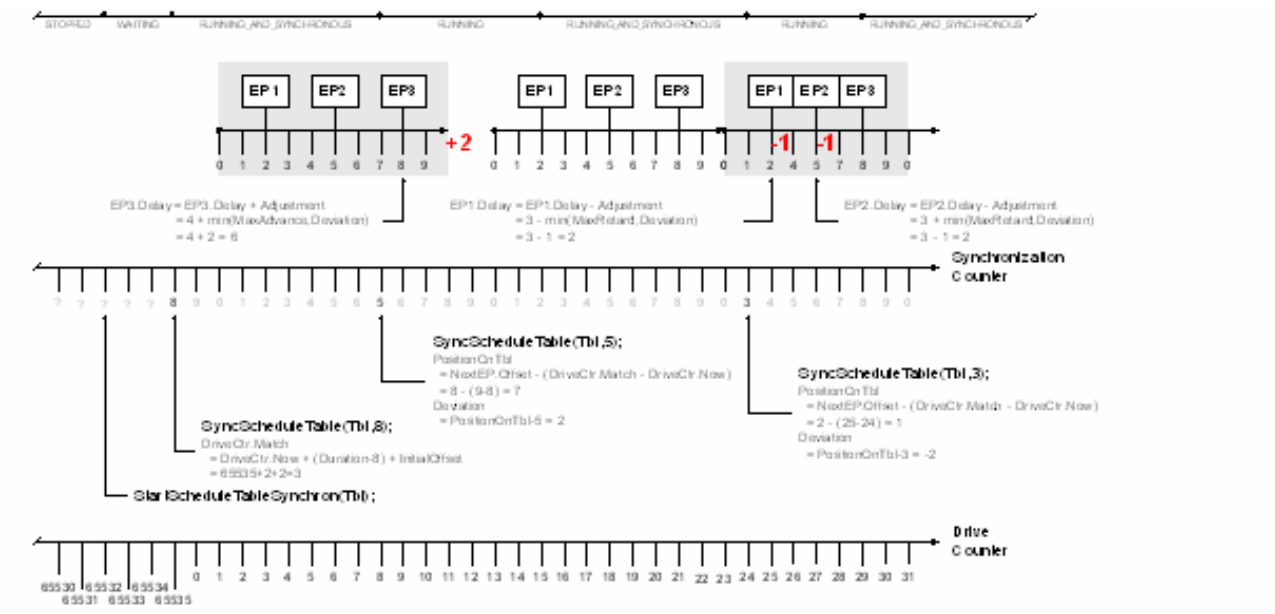


Figure 7.8: Explict Schedule Table Synchronization

OS422: The OS shall provide a service to cancel synchronization being performed at adjustable expiry points on a schedule table.
OS227: The Operating System shall extend the service from OS359 to query the state of a schedule table with respect to synchronization.

7.5 Stack Monitoring Facilities

7.5.1 Background & Rationale

On processors that do not provide any memory protection hardware it may still be necessary to provide a "best effort with available resources" scheme for detectable classes of memory faults. Stack monitoring will identify where a task or OsIsr has exceeded a specified stack usage at context switch time. This may mean that there is considerable time between the system being in error and that fault being detected. Similarly, the error may have been cleared at the point the fault is notified (the stack may be less than the specified size when the context switch occurs).

It is not usually sufficient to simply monitor the entire stack space for the system because it is not necessarily the Task/OsIsr that was executing that used more than stack space than required – it could be a lower priority object that was pre-empted.

Significant debugging time can be saved by letting the OS correctly identify the Task/Category 2 OsIsr in error.

Note that for systems using a MPU and scalability class 3 or 4 a stack overflow may cause a memory exception before the stack monitoring is able to detect the fault.

7.5.2 Requirements

OS067: The Operating System shall offer a stack monitoring which detects possible stack faults of Task(s)/Category 2 OsIsr(s).

OS068: If a stack fault is detected by stack monitoring AND the configured scalability class is 1 or 2, the Operating System shall call the `ShutdownOS()` service with the status `E_OS_STACKFAULT`.

OS396: If a stack fault is detected by stack monitoring AND the configured scalability class is 3 or 4, the Operating System shall call the `ProtectionHook()` with the status `E_OS_STACKFAULT`.

7.6 OS-Application

7.6.1 Background & Rationale

An AUTOSAR OS must be capable of supporting a collection of OS objects (Tasks, OsIsrs, Alarms, Schedule tables, Counters, Resources) that form a cohesive functional unit. This collection of object is termed an *OS-Application*.

The OS is responsible for scheduling the available processing resource between the OS-Applications that share the processor. If OS-Application(s) are used, all Tasks, OsIsrs, Resources, Counters, Alarms and Schedule tables must belong to an OS-Application. Events belong to the OSApplications of the tasks that use them. All objects which belong to the same OS-Application have access to each other. The right to access objects from other OS-Applications may be granted during configuration. An event is accessible if the task for which the event can be set is accessible. Access means that these OS objects are allowed as parameters to API services.

There are two classes of OS-Application:

(1) Trusted OS-Applications are allowed to run with monitoring or protection features disabled at runtime. They may have unrestricted access to memory, the OS API, and need not have their timing behaviour enforced at runtime. They are allowed to run in privileged mode when supported by the processor.

(2) Non-Trusted OS-Applications are not allowed to run with monitoring or protection features disabled at runtime. They have restricted access to memory, restricted access to the OS API and have their timing behaviour enforced at runtime. They are not allowed to run in privileged mode when supported by the processor.

It is assumed that the OS itself is trusted.

There are services offered by the AUTOSAR OS which give the caller information about the access rights and the membership of objects. These services are intended to be used in case of an inter-OS-Application call for checking access rights and arguments.

The running OS-Application is defined as the OS-Application to which the currently running Task or OsIsr belongs. In case of a hook routine the Task or OsIsr which caused the call of the hook routine defines the running OS-Application.

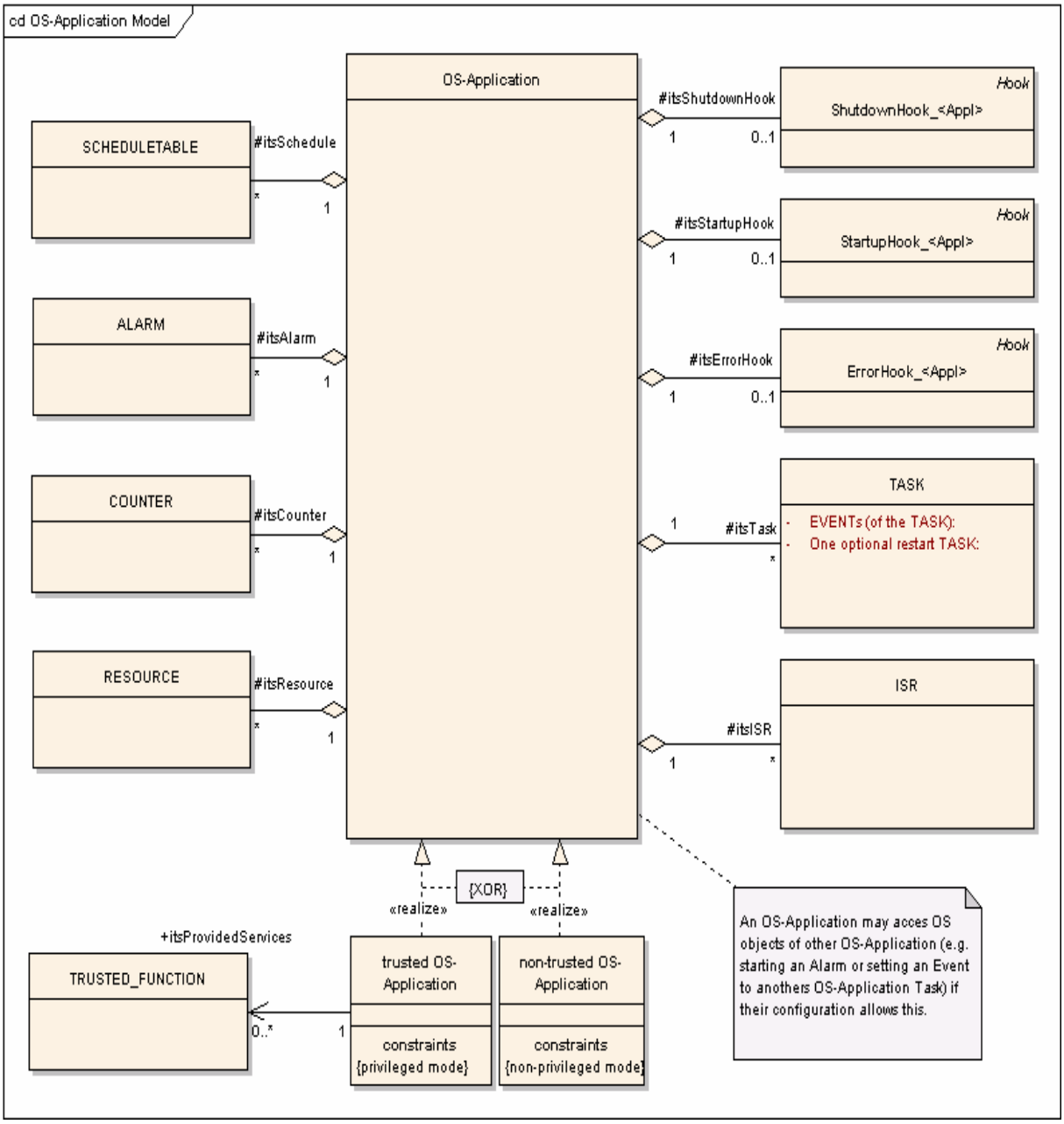


Figure 7.9: UML-model of OS-Application

7.6.2 Requirements

OS445: The Operating System shall support OS-Applications which are a composition of (at least one Task OR OsIsr) AND (zero or more Alarms, Schedule tables, Counters or Resources) AND (zero or one hooks for startup, error and shutdown).

OS446: The Operating System shall support the notion of trusted and not trustet OS-Applications.

OS464: Trusted OS-Applications may offer services (“trusted services”) to other (even non-trusted) OS-Applications.

OS016: The Operating System shall provide a service to determine the currently running OS-Application (a unique identifier shall be allocated to each application).

OS017: The Operating System shall provide a service to determine to which OS-Application a given Task, OsIsr, Resource, Counter, Alarm or Schedule Table belongs.

OS256: The Operating System shall provide a service to determine which OS-Applications are allowed to use the IDs of a Task, OsIsr, Resource, Counter, Alarm or Schedule Table in API calls.

OS258: The Operating System shall provide a service to terminate the OS-Application to which the calling Task/Category 2 OsIsr/application specific error hook belongs. (This is an OS-Application level variant of the `TerminateTask()` service)

OS447: Terminating an OS-Application means to:

- terminate all running, ready and waiting Tasks/OsIsrs of the OS-Application AND
- disabling all interrupts of the OS-Application AND
- stop all active alarms of the OS-Applications AND
- stop all schedule tables of the OS-Application.

OS448: OS-Applications, trusted or non-trusted, shall by default have only access rights to objects belonging to this OS-Application. Access rights from other OS-Applications shall be granted explicitely by configuration.

7.7 Protection Facilities

Protection is only possible for OS managed objects. This means that:

- It is not possible to provide protection during runtime of Category 1 OsIsrs, because the operating system is not aware of any Category 1 OsIsrs being invoked. Therefore, if any protection is required, Category 1 OsIsrs shall be

avoided. If Category 1 interrupts AND OS-Applications are used together then all Category 1 OsIsr must belong to a trusted OS-Application.

- It is not possible to provide protection between functions called from the body of the same Task/Category 2 OsIsr.

7.7.1 Memory Protection

7.7.1.1 Background & Rationale

Memory protection will only be possible on processors that provide hardware support for memory protection.

The memory protection scheme is based on the (data, code and stack) sections of the executable program.

Stack: An OS-Application comprises a number of Tasks and OsIsrs. The stack for these objects, by definition, belongs only to that OS-Application and there is therefore no need to share stack data between OS-Applications. The stacks of Task(s)/Category 2 OsIsr(s) belonging to different OS-Applications need to be protected.
The smallest unit managed by the OS is the stack of Task(s)/Category 2 OsIsr(s). Memory protection for the stack for each Task/Category 2 OsIsr within the same OS-Application is sometimes useful, mainly for two reasons:

(1) Provide a better (more immediate) detection of a stack overflow for the Task/Category 2 OsIsr compared to stack monitoring.
(2) Provide protection between the constituent parts of an OS-Application (e.g. to satisfy some safety constraints).

Data: OS-Applications can have private data sections and Tasks/OsIsrs may have private data sections. OS-Application's private data sections are shared by all Tasks/OsIsrs belonging to that OS-Application.

Code: Code sections are either private to an OS-Application or can be shared between all OS-Applications (to use shared libraries). In the case where code protection is not used, executing incorrect code will eventually result in a memory, timing or service violation.

7.7.1.2 Requirements

Data Sections and Stack

OS198: The Operating System shall prevent write access to its own data sections and its own stack from non-trusted OS-Applications.

Private data of an OS-Application

OS026: The Operating System may prevent read access to an OS-Application's data section attempted by other non-trusted OS-Applications.

OS086: The Operating System shall permit an OS-Application read and write access to that OS-Application's own private data sections.

OS207: The Operating System shall prevent write access to the OS-Application's private data sections from other non-trusted OS-Applications.

Private Stack of Task/OsIsr

OS196: The Operating System shall permit a Task/Category 2 OsIsr read and write access to that Task's/Category 2 OsIsr's own private stack.

OS208: The Operating System may prevent write access to the private stack of Tasks/Category 2 OsIsrs of a non-trusted application from all other Tasks/OsIsrs in the same OS-Application.

OS355: The Operating System shall prevent write access to all private stacks of Tasks/Category 2 OsIsrs of an OS-Application from other non-trusted OS-Applications.

Private data of a Task/OsIsr

OS087: The Operating System shall permit a Task/Category 2 OsIsr read and write access to that Task's/Category 2 OsIsr's own private data sections.

OS195: The Operating System may prevent write access to the private data sections of a Task/Category 2 OsIsr of a non-trusted application from all other Tasks/OsIsrs in the same OS-Application.

OS356: The Operating System shall prevent write access to all private data sections of a Task/Category 2 OsIsr of an OS-Application from other non-trusted OS-Applications.

Code Sections

OS027: The Operating System may provide an OS-Application the ability to protect its code sections against executing by non-trusted OS-Applications.

OS081: The Operating System shall provide the ability to provide shared library code in sections that are executable by all OS-Applications.

Peripherals

OS209: The Operating System shall permit trusted OS-Applications read and write access to peripherals.

OS083: The Operating System shall allow non-trusted OS-Applications to write to their assigned peripherals only (incl. reads that have the side effect of writing to a memory location).

Memory Access Violation

OS044: If a memory access violation is detected, the Operating System shall call the Protection Hook with status code `E_OS_PROTECTION_MEMORY`.

7.7.2 Timing Protection

7.7.2.1 Background & Rationale

A timing fault in a real-time system occurs when a task or interrupt misses its deadline at runtime.

AUTOSAR OS does not offer deadline monitoring for timing protection. Deadline monitoring is insufficient to correctly identify the Task/OsIsr causing a timing fault in an AUTOSAR system. When a deadline is violated this may be due to a timing fault introduced by an unrelated Task/OsIsr that interferes/blocks for too long. The fault in this case lies with the unrelated Task/OsIsr and this will propagate through the system until a Task/OsIsr misses its deadline. The Task/OsIsr that misses a deadline is therefore not necessarily the Task/OsIsr that has failed at runtime, it is simply the earliest point that a timing fault is detected.

If action is taken based on a missed deadline identified with deadline monitoring this would potentially use false evidence of error to terminate a correct OS-Application in favour of allowing an incorrect OS-Application to continue running. The problem is best illustrated by example. Consider a system with the following configuration:

TaskID	Priority	Execution Time	Deadline (=Period)
A	High	1	5
B	Medium	3	10
C	Low	5	15

Assuming that all tasks are ready to run at time zero, the following execution trace would be expected and all tasks would meet their respective deadlines.

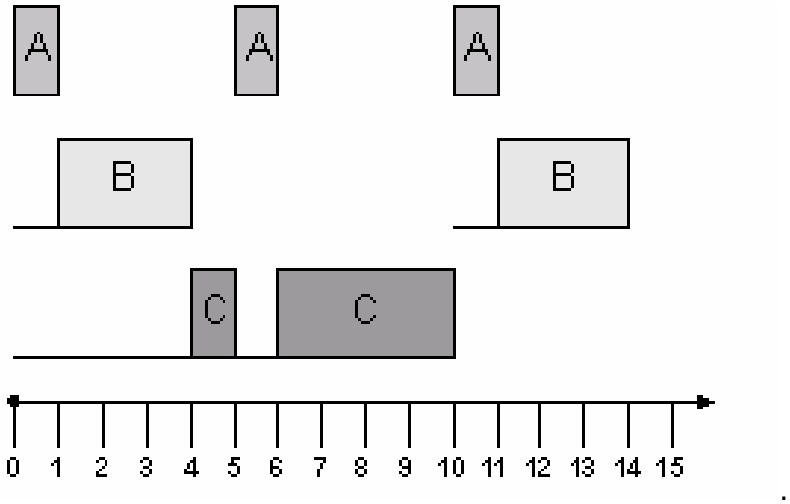


Figure 7.10: Example execution trace

Now consider the case when tasks A and B behave incorrectly. The figure below shows both task A and task B executing for longer than specified and task B arriving 2 ticks earlier than specified. Both tasks A and B meet their deadlines. Task C however, behaves correctly but it fails to meet its deadline because of the incorrect execution of Tasks A and B. This is fault propagation – a fault in an unrelated part of the system is causing a correctly functionality part of the system to fail.

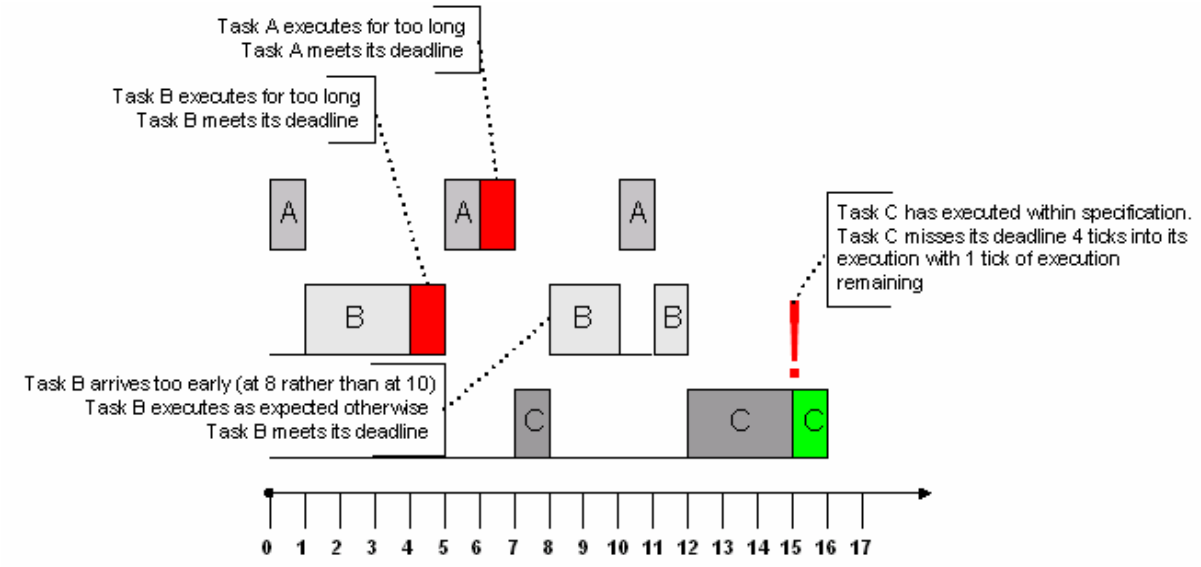


Figure 7.11: Insufficiency of Deadline Monitoring

Whether a task or OsIsr meets its deadline in a fixed priority preemptive operating system like AUTOSAR OS is determined by the following factors:

(1) the execution time of Task/OsIsrs in the system

(2) the blocking time that Task/OsIsrs suffers from lower priority Tasks/OsIsrs locking shared resources or disabling interrupts

(3) the interarrival rate of Task/OsIsrs in the system

For safe and accurate timing protection it is necessary for the operating system to control these factors at runtime to ensure that Tasks/OsIsrs can meet their respective deadlines.

AUTOSAR OS prevents timing errors from (1) by using *execution time protection* to guarantee a statically configured upper bound, called the Execution Budget, on the execution time of:

- Tasks
- Category 2 OsIsrs

AUTOSAR OS prevents timing errors from (2) by using *locking time protection* to guarantee a statically configured upper bound, called the Lock Budget, on the time that:

- Resources are held by Tasks/Category 2 OsIsrs
- OS interrupts are suspended by Tasks/Category 2 OsIsrs
- ALL interrupts are suspended/disabled by Tasks/Category 2 OsIsrs

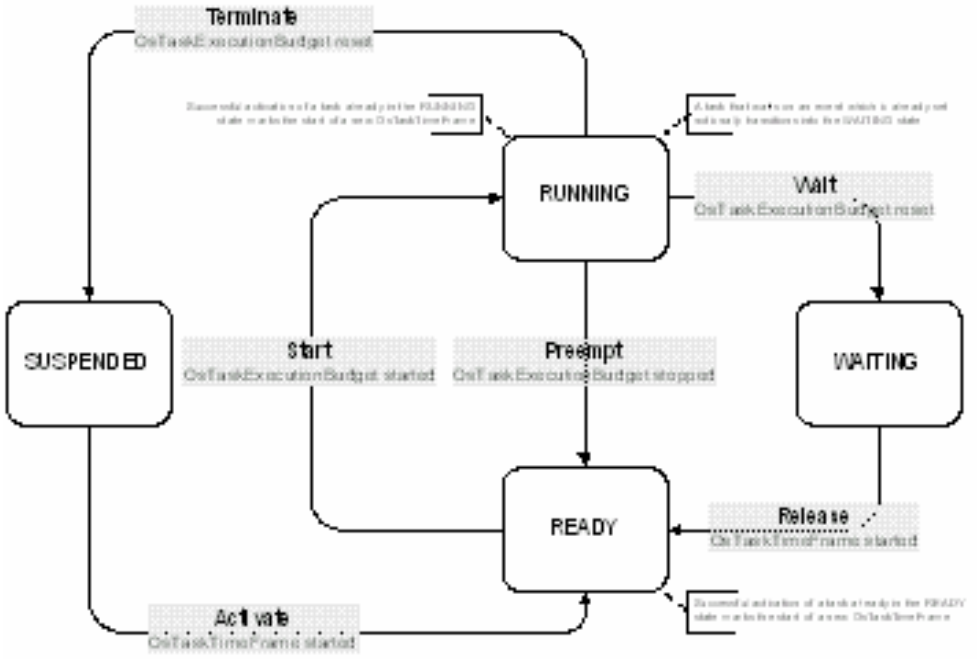
AUTOSAR OS prevents timing errors from (3) by using *inter-arrival time protection* to guarantee a statically configured lower bounds, called the Time Frame, on the time between:

- A task being permitted to transition into the `READY` state due to:
  - Activation (the transition from the `SUSPENDED` to the `READY` state)
  - Release (the transition from the `WAITING` to the `READY` state)
- A Category 2 OsIsr arriving
  An arrival occurs when the Category 2 OsIsr is recognized by the OS

Inter-arrival time protection for basic tasks controls the time between successive activations, irrespective of whether activations are queued or not. In the case of queued activations, activating a basic task which is in the `READY` or `RUNNING` state is a new activation because it represents the activation of a new instance of the task. Inter-arrival time protection therefore interacts with queued activation to control the rate at which the queue is filled.

Inter-arrival time protection for extended tasks controls the time between successive activations *and* releases. When a task is in the `WAITING` state and multiple events are set with a single call to `SetEvent()` this represents a single release. When a task waits for one or more events which are already set this represents a notional Wait/Reklease/Start transition and therefore is considered as a new release.

The following figure shows how execution time protection and inter-arrival time protection interact with the task state transition model for AUTOSAR OS.



Notes:

1. Inter-arrival time enforcement on Category 2 OsIsrs can be used to protect an ECU from a “babbling idiot” source of interrupts (e.g. a CAN controller taking an interrupt each time a frame is received from another ECU on the network) and provides the type of protection given by the OSEKtime Interrupt re-enable schedule event [13].
2. Timing protection only applies to Tasks or Catgeory 2 OsIsrs. There is no protection for Category 1 OsIsrs
3. Timing protection does not apply before the OS is started.
4. In the case of trusted OS-Applications it is essential that all timing information is correct, otherwise the system may fail at run-time. For a non-trusted OS-Application, timing protection can be used to enforce timing boundaries between executable objects.

7.7.2.2 Requirements

OS028: In a non-trusted OS-Application, the Operating System shall apply timing protection to every Task/Category 2 OsIsr of this non-trusted OS-Application.

OS089: In a trusted OS-Application, the Operating System shall offer the ability to apply timing protection to Tasks/Category 2 OsIsrs of this OS-Application.

OS397: If no OS-Application is configured, the Operating System shall be able to apply timing protection to Tasks/Category 2 OsIsrs.

Timing Protection: Tasks

OS064: If a task’s OsTaskExecutionBudget is reached then the Operating System shall call the `ProtectionHook()` with `E_OS_PROTECTION_TIME`.

OS473: The Operating System shall reset a task's OsTaskExecutionBudget on a transition to the `SUSPENDED` or `WAITING` states.

OS465: The Operating System shall limit the inter-arrival time of tasks to one per OsTaskTimeFrame.

OS469: The Operating System shall start an OsTaskTimeFrame when a task is activated successfully.

OS472: The Operating System shall start an OsTaskTimeFrame when a task is released successfully.

OS466: If an attempt is made to activate a task before the end of an OsTaskTimeFrame then the Operating System shall not perform the activation AND shall call the `ProtectionHook()` with `E_OS_PROTECTION_ARRIVAL`.

OS467: If an attempt is made to release a task before the end of an OsTaskTimeFrame then the Operating System shall not perform the release AND shall call the `ProtectionHook()` with `E_OS_PROTECTION_ARRIVAL`.

Timing Protection: OsIsrs

OS210: If a Category 2 OsIsr's OsIsrExecutionBudget is reached then the Operating System shall call the `ProtectionHook()` with `E_OS_PROTECTION_TIME`.
OS474: The Operating System shall rest an OsIsr's OsIsrExecutionBudget when the OsIsr returns control to the Operating terminates.

OS470: The Operating System shall limit the inter-arrival time of Category 2 OsIsrs to one per OsIsrTimeFrame.

OS471: The Operating System shall measure the start of an OsIsrTimeFrame from the point at which it recognises the interrupt (i.e. in the Operating System interrupt wrapper).

OS048: If Category 2 interrupt occurs before the end of the OsIsrTimeFrame then the Operating System shall not execute the user provided OsIsr AND shall call the `ProtectionHook()` with `E_OS_PROTECTION_ARRIVAL`.

Timing Protection: Resource Locking and Interrupt Disabling

OS033: If a Task/Category 2 OsIsr holds an OSEK Resource and exceeds the Os[Task|Isr]ResourceLockBudget, the Operating System shall call the `ProtectionHook()` with `E_OS_PROTECTION_LOCKED`.

OS037: If a Task/Category 2 OsIsr disables interrupts (via `Suspend/Disable|All/OS|Interrupts()`) and exceeds the configured Os[Task|Isr][All|OS]InterruptLockBudget, the Operating System shall call the `ProtectionHook()` with `E_OS_PROTECTION_LOCKED`.

7.7.2.3 Implementation Notes

Execution time enforcement requires hardware support, e.g. a timing enforcement interrupt. If an interrupt is used to implement the time enforcement, the priority of this interrupt shall be high enough to “interrupt” the supervised tasks or OsIsrs.

7.7.3 Service Protection

Background & Rationale

As OS-Applications may interact with the OS through services, it is essential that the service calls will not corrupt the OS itself. Service Protection guards against such corruption at runtime.

There are a number of cases to consider with Service Protection: An OS-Application makes an API call

(1) with an invalid handle or out of range value.

(2) in the wrong context, e.g. calling `ActivateTask()` in the `StartupHook()`.

(3) or fails to make an API call that results in the OSEK OS being left in an undefined state, e.g. it terminates without a `ReleaseResource()` call

(4) that impacts on the behaviour of every other OS-Application in the system, e.g. `ShutdownOS()`

(5) to manipulate OS objects that belong to another OS-Application (to which it does not have the necessary permissions), e.g. an OS-Application tries to execute `ActivateTask()` on a task it does not own.

The OSEK OS already provides some service protection through the status codes returned from service calls and this will provide the basis for service protection. This means that service protection will only apply for the extended status of OSEK OS.

However, OSEK OS does not cover all the cases outlined above. The following sections describe – besides the mandatory extended status – the additional protection requirements to be applied in each of these cases.

7.7.3.1 Invalid Object Parameter or Out of Range Value

7.7.3.1.1 Background & Rationale

The current OSEK OS’ service calls already return `E_OS_ID` on invalid objects (i.e. objects not defined in the OIL file) and `E_OS_VALUE` for out of range values (e.g. setting an alarm cycle time less than `OsCounterMinCycle`).

7.7.3.1.2 Requirements

OS051: If an invalid address (address is not writable by this OS-Application) is passed as an out-parameter to an OS service, the Operating System shall return the status code `E_OS_ILLEGAL_ADDRESS`.

7.7.3.2 Service Calls Made from Wrong Context

7.7.3.2.1 Background & Rationale

The current OSEK OS defines the valid calling context for service calls ([12], Fig. 12-1), however protects against only a small set of these invalid calls, e.g. calling `TerminateTask()` from a Category 2 OsIsr.

Service	Task	Cat1 OsIsr	Cat2 OsIsr	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	✓		✓							

Service	Task	Cat1 OsIsr	Cat2 OsIsr	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
CheckISRMemoryAccess	✓		✓	✓						✓
CheckTaskMemoryAccess	✓		✓	✓						✓
CheckObjectAccess	✓		✓	✓						✓
CheckObjectOwnership	✓		✓	✓						✓
StartScheduleTableRel	✓		✓							
StartScheduleTableAbs	✓		✓							
StopScheduleTable	✓		✓							
NextScheduleTable	✓		✓							
StartScheduleTableSynchron	✓		✓							
SyncScheduleTable	✓		✓							
GetScheduleTableStatus	✓		✓							
SetScheduleTableAsync	✓		✓							
IncrementCounter	✓		✓							
GetCounterValue	✓		✓							
GetElapsedCounterValue	✓		✓							
TerminateApplication	✓		✓	✓[1]						

Tab. 1: Allowed Calling Context for OS Service Calls

In the table above “C” indicates that validity is only “Checked in Extended status by E_OS_CALLEVEL”.

7.7.3.2.2 Requirements

OS088: If an OS-Application makes a service call from the wrong context AND is currently not inside a Category 1 OsIsr the Operating System shall not perform the requested action (the service call shall have no effect), and return E_OS_CALLEVEL or the “invalid value” of the service.

7.7.3.3 Services with Undefined Behaviour

7.7.3.3.1 Background & Rationale

There are a number of situations where the behaviour of OSEK OS is undefined in extended status. This is unacceptable when protection is required as it would allow the OS to be corrupted through its own service calls. The implementation of service protection for the OS must therefore describe and implement a behaviour that does

[1] Only in application specific ErrorHooks.

not jeopardise the integrity of the system or of any OS-Application which did not cause the specific error.

7.7.3.3.2 Requirements

Tasks ends without calling a `TerminateTask()` or `ChainTask()`

OS052: If a task returns from its entry function without making a `TerminateTask()` or `ChainTask()` call, the Operating System shall terminate the task (and call the `PostTaskHook()` if configured).

OS069: If a task returns from its entry function without making a `TerminateTask()` or `ChainTask()` call AND the error hook is configured, the Operating System shall call the `ErrorHook()` (this is done regardless of whether the task causes other errors, e.g. `E_OS_RESOURCE`) with status `E_OS_MISSINGEND` before the task leaves the `RUNNING` state.

OS070: If a task returns from the entry function without making a `TerminateTask()` or `ChainTask()` call and still holds OSEK Resources, the Operating System shall release them.

OS239: If a task returns from the entry function without making a `TerminateTask()` or `ChainTask()` call and interrupts are still disabled, the Operating System shall enable them.

Category 2 OsIsr ends with locked interrupts or allocated resources

OS368: If a Category 2 OsIsr calls `DisableAllInterupts()` / `SuspendAllInterrupts()` / `SuspendOSInterrupts()` and ends (returns) without calling the corresponding `EnableAllInterrupts()` / `ResumeAllInterrupts()` / `ResumeOSInterrupts()`, the Operating System shall perform the missing service and shall call the `ErrorHook()` (if configured) with the status `E_OS_DISABLEDINT`.

OS369: If a Category 2 OsIsr calls `GetResource()` and ends (returns) without calling the corresponding `ReleaseResource()`, the Operating System shall perform the `ReleaseResource()` call and shall call the `ErrorHook()` (if configured) with the status `E_OS_RESOURCE`.

PostTaskHook called during ShutdownOS()

OS071: If the `PostTaskHook()` is configured, the Operating System shall not call the hook if `ShutdownOS()` is called.

Tasks/OsIsrs calls EnableAllInterrupts/ResumeAllInterrupts/ResumeOSInterrupts without a corresponding disable

OS092: If `EnableAllInterrupts()` / `ResumeAllInterrupts()` / `ResumeOSInterrupts()` are called and no corresponding `DisableAllInterupts()` / `SuspendAllInterrupts()` / `SuspendOSInterrupts()` was done before, the Operating System shall not perform this OS service.

Tasks/OsIsrs calling OS services when DisableAllInterupts/SuspendAllInterrupts/SuspendOSInterrupts called

OS093: If interrupts are disabled/suspended by a Task/OsIsr and the Task/OsIsr calls any OS service (excluding the interrupt services) then the Operating System shall ignore the service AND shall return `E_OS_DISABLEDINT` if the service returns a `StatusType` value.

7.7.3.4 Service Restrictions for Non-Trusted OS-Applications

7.7.3.4.1 Background & Rationale

The OS service calls available are restricted according to the calling context (see Section 7.7.3.2). In a protected system, additional constraints need to be placed to prevent non-trusted OS-Applications executing API calls that can have a global effect on the system. Each level of restriction is a proper subset of the previous level as shown in the figure below.

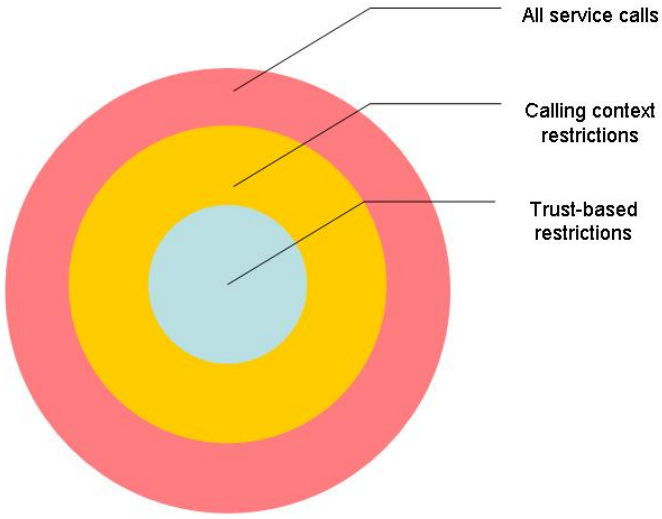


Figure 7.12: API Restrictions

There are two defined integrity levels:

1. Trusted
2. Non-Trusted

that correspond exactly with trusted and non-trusted OS-Applications.

7.7.3.4.2 Requirements

OS054: The Operating System shall ignore calls to `ShutdownOS()` from non-trusted OS-Applications.

7.7.3.5 Service Calls on Objects in Different OS-Applications

7.7.3.5.1 Background

Section 7.7.3.1 stated that `E_OS_ID` is returned by OSEK OS service calls when the object is invalid. Under the protection scheme a service call can be invalid because the caller does not have valid permissions for the object (a new meaning for multi-OS-Application systems).
This is a similar case to an object not being accessible in OSEK OS (for example, when a task tries to get a resource which exists in the system but has not been configured as used by the task).

7.7.3.5.2 Requirements

OS056: If an OS-object identifier is the parameter of a system service, and no sufficient access rights have been assigned at configuration time (Parameter `Os[...]AccessingApplication`) to the calling Task/Category 2 OsIsr, the system service shall return E_OS_ACCESS.

OS449: CheckTaskMemoryAccess and CheckIsrMemoryAccess check the memory access. Memory access checking is possible for all OS-Applications and from all OS-Applications and does not need granted rights (This is an exception to OS056).

OS450: CheckObjectAccess checks the access rights for OS objects. Checking object access is possible for all OS-Applications and from all OS-Applications and does not need granted rights (This is an exception to OS056).

7.7.4 Protecting the Hardware used by the OS

7.7.4.1 Background & Rationale

Where a processor supports privileged and non-privileged mode it is usually the case that certain registers, and the instructions to modify those registers, are inaccessible outside the privileged mode.

On such hardware, executing the OS in privileged mode and Tasks/OsIsrs in non-privileged mode protects the registers fundamental to OS operation from inadvertent corruption by the objects executing in non-privileged mode. The OS services will need to execute in privileged mode as they will need to modify the registers that are protected outside this mode.

The OS may use the control registers of the MPU, timer unit(s), interrupt controller, etc. and therefore it is necessary to protect those registers against non-trusted OS-Applications.

7.7.4.2 Requirements

OS058: If supported by hardware, the Operating System shall execute non-trusted OS-Applications in non-privileged mode.

OS096: As far as supported by hardware, the Operating System shall not allow non-trusted OS-Applications to access control registers managed by the Operating System.

OS245: If an instruction exception occurs (e.g. division by zero) the operating system shall call the protection hook with `E_OS_PROTECTION_EXCEPTION`.

7.7.4.3 Implementation Notes

When the OS is running non-trusted OS-Applications, the OS treatment of interrupt entry and hook routines must be carefully managed.

Interrupt handling: Where the MCU is moded (as discussed in this section) OsIsrs will require the OS to do extra work in the `OsIsr()` wrapper. OsIsrs will typically be entered in privileged mode. If the handler is part of a non-trusted OS-Application then the `OsIsr()` wrapper must make sure that a switch to non-privileged mode occurs before the handler executes.

7.7.5 Providing »Trusted Functions«

7.7.5.1 Background & Rationale

An OS-Application may invoke a Trusted Function provided by (another) trusted OS-Application. That may require a switch from non-privileged to privileged mode. This is typically achieved by these operations:

(1) Each trusted OS-Application may export services which are callable from other OS-Applications.
(2) During configuration these trusted services must be configured to be called from a non-trusted OS-Application.
(3) The call from the non-trusted OS-Application to the trusted service is using a mechanism (e.g. trap/software interrupt) provided by the OS. The service is passed as an identifier that is used to determine, in the trusted environment, if the service can be called.
(4) The OS offers services to check if a memory region is write/read/execute accessible from an OS-Application. It also returns information if the memory region is part of the stack space.

The system does not support »non-trusted services«.

7.7.5.2 Requirements

OS451: The Operating System shall allow to export services from trusted OS-Applications.

OS097: The Operating System shall provide a mechanism to call a trusted function from a (trusted or non-trusted) OS-Application.

OS100: If a called trusted function is not configured the Operating System shall call the ErrorHook with `E_OS_SERVICEID`.

OS099: The Operating System shall offer OS-Applications a service to check if a memory region is write/read/execute accessible from a Task/Category 2 OsIsr and also return information if the memory region is part of the stack space.

7.8 Protection Error Handling

7.8.1 Background & Rationale

The OS can detect protection errors based on statically configured information on what the constituent parts of an OS-Application can do at runtime. See Section 7.7.

Unlike monitoring, protection facilities will trap the erroneous state at the point the error occurs, resulting in the shortest possible time between transition into an erroneous state and detection of the fault. The different kinds of protection errors are described in the glossary. If a protection error occurs before the operating system is started the behaviour is not defined. If a protection error happens during shutdown, e.g. in the application-specific shutdown hook, an endless loop between the shutdown service and the protection hook may occur.

In the case of a protection error, the OS calls a user provided Protection Hook for the notification of protection errors at runtime. The Protection Hook runs in the context of the OS and must therefore be trusted code.

The OS itself needs only to detect an error and provide the ability to act. The Protection Hook can select one out of four options the OS provides, which will be performed after returning from the Protection Hook, depending on the return value of the Protection Hook. The options are:

- do nothing

1. forcibly terminate the faulty Task/Category 2 OsIsr
2. forcibly terminate all tasks and OsIsrs in the faulty OS-Application
    a. without restart of the OS-Application
    b. with restart of the OS-Application

3. shutdown the OS.
4.

Requirements OS243 and OS244 define the order of the default reaction if no faulty Task/Category 2 OsIsr or OS-Application can be found, e.g. in the system specific hook routines. Also OS-Applications are only mandatory in Scalability Classes 3 and 4, therefore in other Scalability Classes OS-Applications need not be defined.

Note that forcibly terminating interrupts is handled differently in “forcibly terminate the faulty OsIsr” and “forcibly terminate the OS-Application”. If a faulty OsIsr is forcibly terminated, the current invocation of the OsIsr is terminated. A subsequent invocation is allowed. If the OS-Application is forcibly terminate, then the interrupt source is also disabled, preventing subsequent interrupts.

7.8.2 Requirements

OS211: The Operating System shall execute the `ProtectionHook()` with the same permissions as the Operating System.

OS106: The Operating System shall perform one of the following reactions depending on the return value of the `ProtectionHook()`:

- Do nothing
- Forcibly terminate the faulty Task/Category 2 OsIsr OR
- Forcibly terminate the faulty OS-Application OR
- Forcibly terminate the faulty OS-Application and restart the OS-Application. OR
- Call `ShutdownOS()`.

OS107: If no `ProtectionHook()` is configured and a protection error occurs, the Operating System shall call `ShutdownOS()`.

OS475: If the reaction is to do nothing and the `ProtectionHook()` was not called with `E_OS_PROTECTION_ARRIVAL` then the Operating System shall call `ShutdownOS()`

OS243: If the reaction is to forcibly terminate the Task/Category 2 OsIsr and no Task or OsIsr can be associated with the error, the running OS-Application is forcibly terminated by the Operating System.

OS244: If the reaction is to forcibly terminate the faulty OS-Application and no OS-Application can be assigned, `ShutdownOS()`is called.

OS108: If the Operating System forcibly terminates a task, it terminates the task (no `PostTaskHook()` for the task will be called), releases all allocated OSEK resources and calls `EnableAllInterrupts()`/ `ResumeOSInterrupts()` / `ResumeAllInterrupts()` if the Task called `DisableAllInterrupts()` / `SuspendOSInterrupts()` / `SuspendAllInterrupts()` before without the corresponding `EnableAllInterrupts()`/ `ResumeOSInterrupts()` / `ResumeAllInterrupts()` call.

OS109: If the Operating System forcibly terminates an interrupt service routine, it clears the interrupt request, aborts the interrupt service routine (The interrupt source stays in the current state.) and releases all OSEK resources the interrupt service routine has allocated and calls `EnableAllInterrupts()` / `ResumeOSInterrupts()` / `ResumeAllInterrupts()` if the interrupt called `DisableAllInterrupts()` / `SuspendOSInterrupts()` / `SuspendAllInterrupts()` before without the corresponding `EnableAllInterrupts()`/ `ResumeOSInterrupts()` / `ResumeAllInterrupts()` call.

OS110: If the Operating System forcibly terminates an OS-Application, it:
- forcibly terminates all Tasks/OsIsrs of the OS-Application AND
- cancels all alarms of the OS-Application AND
- stops schedule tables of the OS-Application AND
- disables interrupt sources of Category 2 OsIsrs belonging to the OS-Application

OS111: When the Operating System restarts an OS-Application it activates the configured `OsRestartTask`.

7.9 System Scalability

7.9.1 Background & Rationale

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	Described in Section	Scalability Class 1	Scalability Class 2	Scalability Class 3	Scalability Class 4	Hardware requirements
OSEK OS (all conformance classes)	7.1	✓	✓	✓	✓	
Counter Interface	8.4.16	✓	✓	✓	✓	
SWFRT Interface	8.4.17, 8.4.18	✓	✓	✓	✓	Optional feature to support GPT driver
Schedule Tables	7.3	✓	✓	✓	✓	
Stack Monitoring	7.5	✓	✓	✓	✓	
ProtectionHook	7.8		✓	✓	✓	
Timing Protection	7.7.2		✓		✓	Timer(s) with high priority interrupt
Global Time /Synchronization Support	7.4		✓		✓	Global time source
Memory Protection	7.7.1, 7.7.4			✓	✓	MPU

OS-Applications	7.6, 7.10			✓	✓	
Service Protection	7.7.3			✓	✓	
CallTrustedFunction	7.7.5			✓	✓	(Non-)privileged Modes

Tab. 2: Scalability classes

Feature	Scalability Class 1	Scalability Class 2	Scalability Class 3	Scalability Class 4
Minimum number of Schedule Tables supported	2	8	2	8
Minimum number of OS-Applications supported	0	0	2	2
Minimum number of software Counters supported	8	8	8	8

Tab. 3: Minimum requirements of scalability classes

7.9.2 Requirements

OS240: If an implementation of a lower scalability class supports features of higher classes then the interfaces for the features must comply with this specification.

OS241: The operating system shall support the features according to the configured scalability class. (See Tab. 2)

OS327: The operating system shall always use extended status in Scalability Class 3 and 4.

7.10 Hook Functions

7.10.1 Background & Rationale

Hook routines as defined in OSEK OS run at the level of the OS and therefore can only belong to the trusted environment. Furthermore, these hook routines are global to the system (system-specific) and will probably be supplied by the ECU integrator.

In AUTOSAR however, each OS-Application may have the need to execute application specific code e.g. initialize some hardware in its own additional (application-specific) startup hook. These are called application specific hook routines. In general the application specific hooks have the same properties as the hook routines described in the OSEK OS specification. Differences are described below.

7.10.2 Requirements

OS439: The Operating System shall offer the OSEK error macros (`OSError…()`) to all configured error hooks AND there shall be two (like in OIL) global configuration parameter to switch these macros on or off.

StartupHook

OS060: If an application-specific startup hook is configured for an OS-Application `<App>`, the Operating System shall call `StartupHook_<App>` on startup of the OS.

OS226: The Operating System shall execute an application-specific startup hook with the access rights of the associated OS-Application.

OS236: If both a system-specific and one (or more) application specific startup hook(s) are configured, the Operating System shall call the system-specific startup hook before the application-specific startup hook(s).

ShutdownHook

OS112: If an application-specific shutdown hook is configured for an OS-Application `<App>`, the Operating System shall call `ShutdownHook_<App>` on shutdown of the OS.

OS225: The Operating System shall execute an application-specific shutdown hook with the access rights of the associated OS-Application.

OS237: If both a system-specific and one (or more) application specific shutdown hook(s) are configured, the Operating System shall call the system-specific shutdown hook after the application-specific shutdown hook(s).

Error Hook

OS246: When an error occurs AND an application-specific error hook is configured for the faulty OS-Application <App>, the Operating System shall call that application-specific error hook ErrorHook_<App> after the system specific error hook is called (if configured).

OS085: The Operating System shall execute an application-specific error hook with the access rights of the associated OS-Application.

OS367: Operating System services which do not return a StatusType shall not raise the error hook(s).

7.11 Error classification

Instead of specifying two versions for production and development errors the AUTOSAR OS provides a finer granularity to adjust the error handling to specific needs, e.g. Scalability Classes, standard and extended status, switching on/off of hook routines.

Type or error	*Relevance*	*Related error code*	*Value*
Service can not be called.	Production	E_OS_SERVICEID	Assigned by implementation
An invalid address is given as a parameter to a service.	Production	E_OS_ILLEGAL_ADDRESS	Assigned by implementation
Tasks terminates without a TerminateTask() or ChainTask() call.	Production	E_OS_MISSINGEND	Assigned by implementation
A service of the OS is called inside an interrupt disable/enable pair.	Production	E_OS_DISABLEDINT	Assigned by implementation
A stack fault detected via stack monitoring by the OS	Production	E_OS_STACKFAULT	Assigned by implementation
A memory access violation occurred	Production	E_OS_PROTECTION_MEMORY	Assigned by implementation
A Task exceeds its execution time budget	Production	E_OS_PROTECTION_TIME	Assigned by implementation
A Category 2 OsIsr exceeds its execution time budget			
A Task/Category 2 arrives before its timeframe has expired	Production	E_OS_PROTECTION_ARRIVAL	Assigned by implementation
A Task/Category 2 OsIsr blocks for too long	Production	E_OS_PROTECTION_LOCKED	Assigned by implementation
A trap occurred	Production	E_OS_PROTECTION_EXCEPTION	Assigned by implementation

8 API specification

8.1 Constants

8.1.1 Error codes of type StatusType

See Section 7.11 and [12].

8.2 Macros

```
OSMEMORY_IS_READABLE(<AccessType>)
OSMEMORY_IS_WRITEABLE(<AccessType>)
OSMEMORY_IS_EXECUTABLE(<AccessType>)
OSMEMORY_IS_STACKSPACE(<AccessType>)
```

These macros return a value not equal to zero if the memory is readable / writable / executable or stack space.

8.3 Type definitions

8.3.1 ApplicationType (for OS-Applications)

Type:	Scalar
Description:	This data type identifies the OS-Application.
Constants of this Type:	`INVALID_OSAPPLICATION`

8.3.2 TrustedFunctionIndexType

Type:	Scalar
Description:	This data type identifies a trusted function.

8.3.3 TrustedFunctionParameterRefType

Type:	Pointer
Description:	This data type points to a structure which holds the arguments for a call to a trusted function.

8.3.4 AccessType

Type:	Integral
Description:	This type holds information how a specific memory region can be accessed.

8.3.5 ObjectAccessType

Type:	Scalar
Description:	This data type identifies if an OS-Application has access to an object.
Constants of this Type:	ACCESS NO_ACCESS

8.3.6 ObjectTypeType

Type:	Scalar
Description:	This data type identifies an object.
Constants of this Type:	OBJECT_TASK OBJECT_ISR OBJECT_ALARM OBJECT_RESOURCE OBJECT_COUNTER OBJECT_SCHEDULETABLE

8.3.7 MemoryStartAddressType

Type:	Pointer
Description:	This data type is a pointer which is able to point to any location in the MCU address space.

8.3.8 MemorySizeType

Type:	Scalar
Description:	This data type holds the size (in bytes) of a memory region.

8.3.9 ISRType

Type:	Scalar
Description:	This data type identifies an interrupt service routine (OsIsr).
Constants of this Type:	INVALID_ISR

8.3.10 ScheduleTableType

Type:	Scalar
Description:	This data type identifies a schedule table.

8.3.11 ScheduleTableStatusType

Type:	Scalar
Description:	This type describes the status of a schedule. The status can be one of the following: o The schedule table is not started (SCHEDULETABLE_STOPPED) o The schedule table will be started after the end of currently running schedule table (schedule table was used in NextScheduleTable( ) service) (SCHEDULETABLE_NEXT) o The schedule table uses implicit synchronization, has been started and is waiting for the global time. (SCHEDULETABLE_WAITING) o The schedule table is running, but is currently not synchronous to a global time source (SCHEDULETABLE_RUNNING) o The schedule table is running and is synchronous to a global time source (SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS)
Constants of this Type:	SCHEDULETABLE_STOPPED SCHEDULETABLE_NEXT SCHEDULETABLE_WAITING SCHEDULETABLE_RUNNING SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS

8.3.12 ScheduleTableStatusRefType

Type:	Pointer
Description:	This data type points to a variable of the data type ScheduleTableStatusType.

8.3.13 CounterType

Type:	Scalar
Description:	This data type identifies a counter.

8.3.14 ProtectionReturnType

Type:	Scalar
Description:	This data type identifies a value which controls further actions of the OS on return from the protection hook.
Constants of this Type:	PRO_IGNORE PRO_TERMINATETASKISR PRO_TERMINATEAPPL PRO_TERMINATEAPPL_RESTART PRO_SHUTDOWN

8.3.15 RestartType

Type:	Scalar
Description:	This data type defines the use of a Restart Task after terminating an OS-Application.
Constants of this Type:	RESTART NO_RESTART

8.3.16 PhysicalTimeType

Type:	Scalar
Description:	This data type is used for values returned by the conversion macro (see OS393()) `OS_TICKS2<Unit>_<Counter>()`.

8.4 Function definitions

The availability of the following services is defined in Tab. 2. The use of these services may be restricted depending on the context they are called from. See Tab. 1 for details.

8.4.1 GetApplicationID

Service name:	GetApplicationID
Syntax:	`ApplicationType GetApplicationID (void)`
Service ID:	`OSServiceId_GetApplicationID`
Sync/Async:	Sync
Reentrancy:	Yes
Parameters (in):	None
Parameters (out):	None
Return value:	<Identifier of running OS-Application> Or
	`INVALID_OSAPPLICATION`
Description:	OS261: `GetApplicationID()` shall return the application identifier to which the executing Task/OsIsr/hook belongs. OS262: If no OS-Application is running, `GetApplicationID()` shall return `INVALID_OSAPPLICATION`.
Caveats:	None
Configuration:	Available in Scalability Classes 3 and 4

8.4.2 GetISRID

Service name:	GetISRID
Syntax:	`ISRType GetISRID (void)`
Service ID:	`OSServiceId_GetISRID`
Sync/Async:	Sync
Reentrancy:	Yes
Parameters (in):	None
Parameters (out):	None
Return value:	<Identifier of running OsIsr> Or
	`INVALID_ISR`
Description:	OS263: If called from category 2 OsIsr (or Hook routines called inside a category 2 OsIsr), `GetISRID()` shall return the identifier of the currently executing OsIsr. OS264: If its caller is not a category 2 OsIsr (or Hook routines called inside a

	category 2 OsIsr), `GetISRID()` shall return `INVALID_ISR`.
Caveats:	None
Configuration:	Available in all Scalability Classes.

8.4.3 CallTrustedFunction

Service name:	CallTrustedFunction	
Syntax:	`StatusType CallTrustedFunction ( TrustedFunctionIndexType FunctionIndex, TrustedFunctionParameterRefType FunctionParams )`	
Service ID:	`OSServiceId_CallTrustedFunction`	
Sync/Async:	Depends on called function. If called function is synchronous then service is synchronous. May cause rescheduling.	
Reentrancy:	Yes	
Parameters (in):	`FunctionIndex`	Index of the function to be called.
	`FunctionParams` or `NULL`	Pointer to the parameters for the function – specified by the FunctionIndex - to be called. If no parameters are provided, a `NULL` pointer has to be passed.
Parameters (out):	None	
Return value:	`E_OK`	No Error
	`E_OS_SERVICEID`	No function defined for this index

Description:	OS265: If <FunctionIndex> is a defined function index, `CallTrustedFunction()` shall switch the processor into privileged mode AND shall call the function <FunctionIndex> out of a list of implementation specific trusted functions AND shall return `E_OK` after completion. OS312: The called trusted function must conform to the following C prototype: `void TRUSTED_<name_of_the_trusted_service>( TrustedFunctionIndexType,TrustedFunctionParameterRefType);` (The arguments are the same as the arguments of CallTrustedFunction). OS266: When the function <FunctionIndex> is called, it shall get the same permissions (access rights) as the associated trusted OS-Application. OS364: If the trusted function is called from a Category 2 OsIsr context it shall continue to run on the same interrupt priority and be allowed to call all system services defined for Category 2 OsIsr (see table in chapter 7.7.3.2). OS365: If the trusted function is called from a task context it shall continue to run on the same priority and be allowed to call all system services defined for tasks (see table in chapter 7.7.3.2). OS292: If the function index <FunctionIndex> is undefined, `CallTrustedFunction()` shall return `E_OS_SERVICEID`.
Caveats:	Normally, a user will not directly call this service, but it will be part of some standard interface, e.g. a standard I/O interface. It is the duty of the called trusted function to check rights of passed parameters, especially if parameters are interpreted as out parameters.
Configuration:	Available in Scalability Classes 3 and 4

8.4.4 CheckISRMemoryAccess

Service name:	CheckISRMemoryAccess	
Syntax:	`AccessType CheckISRMemoryAccess ( ISRType ISRID, MemoryStartAddressType Address, MemorySizeType Size )`	
Service ID:	`OSServiceId_CheckISRMemoryAccess`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ISRID`	OsIsr reference
	`Address`	Start of memory area
	`Size`	Size of memory area
Parameters (out):	None	
Return value:	Value which contains the access rights to the memory area.	

Description:	OS267: If the OsIsr reference <ISRID> is valid, `CheckISRMemoryAccess()` shall return the access rights of the OsIsr on the specified memory area. OS313: If an access right (e.g. “read”) is not valid for the whole specified memory area `CheckISRMemoryAccess()` shall yield no access regarding this right. OS268: If the OsIsr reference <ISRID> is not valid, `CheckISRMemoryAccess()` shall yield no access rights.
Caveats:	None
Configuration:	Available in Scalability Classes 3 and 4

8.4.5 CheckTaskMemoryAccess

Service name:	CheckTaskMemoryAccess	
Syntax:	`AccessType CheckTaskMemoryAccess ( TaskType TaskID, MemoryStartAddressType Address, MemorySizeType Size )`	
Service ID:	`OSServiceId_CheckTaskMemoryAccess`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`TaskID`	Task reference
	`Address`	Start of memory area
	`Size`	Size of memory area
Parameters (out):	None	
Return value:	Value which contains the access rights to the memory area.	
Description:	OS269: If the Task reference <TaskID> is valid, `CheckTaskMemoryAccess()` shall return the access rights of the task on the specified memory area. OS314: If an access right (e.g. “read”) is not valid for the whole specified memory area `CheckTaskMemoryAccess()` shall yield no access regarding this right. OS270: If the Task reference <TaskID> is not valid, `CheckTaskMemoryAccess()` shall yield no access rights.	
Caveats:	None	
Configuration:	Available in Scalability Classes 3 and 4	

8.4.6 CheckObjectAccess

Service name:	CheckObjectAccess	
Syntax:	`ObjectAccessType CheckObjectAccess ( ApplicationType ApplID, ObjectTypeType ObjectType, … )`	
Service ID:	`OSServiceId_CheckObjectAccess`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ApplID`	OS-Application identifier

	ObjectType	Type of the following parameter
	…	The object to be examined
Parameters (out):	None	
Return value:	ACCESS	if the ApplID has access to the object
	NO_ACCESS	Otherwise
Description:	OS271: If the OS-Application <ApplID> has access to the queried object, CheckObjectAccess() shall return ACCESS. OS272: If the OS-Application <ApplID> has no access to the queried object, CheckObjectAccess() shall return NO_ACCESS. OS423: If the object to be examined is not a valid object OR <ApplID> is invalid OR <ObjectType> is invalid THEN the the CheckObjectAccess() shall return NO_ACCESS. OS318: If the object type is OBJECT_RESOURCE AND the object to be examined is the RES_SCHEDULER CheckObjectAccess() shall always return ACCESS.	
Caveats:	None	
Configuration:	Available in Scalability Classes 3 and 4	

8.4.7 CheckObjectOwnership

Service name:	CheckObjectOwnership	
Syntax:	ApplicationType CheckObjectOwnership (ObjectTypeType ObjectType, …)	
Service ID:	OSServiceId_CheckObjectOwnership	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	ObjectType	Type of the following parameter
	…	The object to be examined
Parameters (out):	None	
Return value:	<OS-Application>	The service returns the OS-Application to which the object ObjectType belongs or
	INVALID_OSAPPLICATION	If the object does not exists the service returns:
Description:	OS273: If the specified object ObjectType exists, CheckObjectOwnership() shall return the identifier of the OS-Application to which the object belongs. OS274: If the specified object ObjectType is invalid OR the argument of the type (the "…") is invalid , CheckObjectOwnership() shall return INVALID_OSAPPLICATION. OS319: If the object to be examined is the RES_SCHEDULER CheckObjectOwnership() shall always return INVALID_OSAPPLICATION.	
Caveats:	None	
Configuration:	Available in Scalability Classes 3 and 4	

8.4.8 StartScheduleTableRel

Service name:	StartScheduleTableRel	
Syntax:	`StatusType StartScheduleTableRel ( ScheduleTableType ScheduleTableID, TickType Offset )`	
Service ID:	`OSServiceId_StartScheduleTableRel`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ScheduleTableID`	Schedule table to be started
	`Offset`	Number of ticks on the counter before the the schedule table processing is started
Parameters (out):	None	
Return value:	`E_OK`	No Error
	`E_OS_ID` (only in `EXTENDED` status)	`ScheduleTableID` not valid.
	`E_OS_VALUE` (only in `EXTENDED` status)	`Offset` is greater than (`OsCounterMaxAllowedValue – InitialOffset`) or is equal to 0.
	`E_OS_STATE`	Schedule table was already started.
Description:	OS275: If the schedule table <ScheduleTableID> is not valid, `StartScheduleTableRel()` shall return `E_OS_ID`. **OS452**: If the schedule table <ScheduleTableID> is implicitely synchronized (`OsScheduleTblSyncStrategy = IMPLICIT`), `StartScheduleTableRel()` shall return `E_OS_ID`. OS332: If <Offset> is zero `StartScheduleTableRel()` shall return `E_OS_VALUE`. OS276: If the offset <Offset> is greater than `OsCounterMaxAllowedValue` of the underlying counter minus the Initial Offset, `StartScheduleTableRel()` shall return `E_OS_VALUE`. OS277: If the schedule table <ScheduleTableID> is not in the state `SCHEDULETABLE_STOPPED`, `StartScheduleTableRel()` shall return `E_OS_STATE`. OS278: If its input parameters are valid and the state of schedule table <ScheduleTableID> is `SCHEDULETABLE_STOPPED`, then `StartScheduleTableRel()` shall start the processing of a schedule table <ScheduleTableID>. The Initial Expiry Point shall be processed after <Offset> + Initial Offset ticks have elapsed on the underlying counter. The state of <ScheduleTableID> to `SCHEDULETABLE_RUNNING`.	
Caveats:	None	
Configuration:	Available in all Scalability Classes.	

8.4.9 StartScheduleTableAbs

Service name:	StartScheduleTableAbs
Syntax:	`StatusType StartScheduleTableAbs`

	(
	`ScheduleTableType ScheduleTableID,`
	`TickType Start`
	)	

Service ID:	`OSServiceId_StartScheduleTableAbs`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ScheduleTableID`	Schedule table to be started
	`Start`	Absolute counter tick value at which the schedule table is started.
Parameters (out):	None	
Return value:	`E_OK`	No Error
	`E_OS_ID` (only in `EXTENDED` status)	`ScheduleTableID` not valid.
	`E_OS_VALUE` (only in `EXTENDED` status)	`Tickvalue` is greater than `OsCounterMaxAllowedValue`.
	`E_OS_STATE`	Schedule table was already started.
Description:	OS348: If the schedule table <ScheduleTableID> is not valid, `StartScheduleTableAbs()` shall return `E_OS_ID`. OS349: If the <Tickvalue> is greater than the `OsCounterMaxAllowedValue` of the underlying counter, `StartScheduleTableAbs()` shall return `E_OS_VALUE`. OS350: If the schedule table <ScheduleTableID> is not in the state `SCHEDULETABLE_STOPPED`, `StartScheduleTableAbs()` shall return `E_OS_STATE`. OS351: If its input parameters are valid and <ScheduleTableID> is in the state `SCHEDULETABLE_STOPPED`, `StartScheduleTableAbs()` shall start the processing of schedule table <ScheduleTableID> at count value <Start> and shall set the state of <ScheduleTableID> to `SCHEDULETABLE_RUNNING`. The Initial Expiry Point will be processed when the underlying counter equals <Start>+Initial Offset.	
Caveats:	None	
Configuration:	Available in all Scalability Classes.	

8.4.10 StopScheduleTable

Service name:	StopScheduleTable	

Syntax:	`StatusType StopScheduleTable`
	(
	`ScheduleTableType ScheduleTableID`
	)	

Service ID:	`OSServiceId_StopScheduleTable`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ScheduleTableID`	Schedule table to be stopped
Parameters (out):	None	
Return value:	`E_OK`	No Error
	`E_OS_ID` (only in `EXTENDED` status)	`ScheduleTableID` not valid.

	E_OS_NOFUNC Schedule table was already stopped
Description:	OS279: If the schedule table identifier <ScheduleTableID> is not valid, StopScheduleTable() shall return E_OS_ID. OS280: If the schedule table with identifier <ScheduleTableID> is in state SCHEDULETABLE_STOPPED, StopScheduleTable() shall return E_OS_NOFUNC. OS281: If its input parameters are valid, StopScheduleTable()shall set the state of <ScheduleTableID> to SCHEDULETABLE_STOPPED and (stop the schedule table <ScheduleTableID> from processing any further expiry points and) shall return E_OK.
Caveats:	None
Configuration:	Available in all Scalability Classes.

8.4.11 NextScheduleTable

Service name:	NextScheduleTable	
Syntax:	`StatusType NextScheduleTable ( ScheduleTableType ScheduleTableID_From, ScheduleTableType ScheduleTableID_To )`	
Service ID:	`OSServiceId_NextScheduleTable`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ScheduleTableID_From`	Schedule table
	`ScheduleTableID_To`	Schedule table that provides its series of expiry points
Parameters (out):	None	
Return value:	`E_OK`	No error
	`E_OS_ID` (only in `EXTENDED` status)	`ScheduleTableID_From` or `ScheduleTableID_To` not valid.
	`E_OS_NOFUNC`	`ScheduleTableID_From` not started.
	`E_OS_STATE` (only in `EXTENDED` status)	`ScheduleTableID_To` is started or next.
Description:	OS282: If the input parameter <ScheduleTableID_From> or <ScheduleTableID_To> is not valid, `NextScheduleTable()` shall return `E_OS_ID`. OS330: If schedule table <ScheduleTableID_To> is driven by different counter than schedule table <ScheduleTableID_From> then `NextScheduleTable()` shall return an error `E_OS_ID`. OS283: If the schedule table <ScheduleTableID_From> is in state `SCHEDULETABLE_STOPPED` OR in state `SCHEDULETABLE_NEXT`, `NextScheduleTable()` shall leave the state of <ScheduleTable_From> and <ScheduleTable_To> unchanged and return `E_OS_NOFUNC`.. OS309: If the schedule table <ScheduleTableID_To> is not in state `SCHEDULETABLE_STOPPED`, `NextScheduleTable()` shall leave the state of <ScheduleTable_From> and <ScheduleTable_To> unchanged and return `E_OS_STATE`. OS284: If the input parameters are valid then `NextScheduleTable()` shall start the processing of schedule table <ScheduleTableID_To> <ScheduleTableID_From>.FinalDelay ticks after the Final Expiry Point on <ScheduleTableID_From> is processed and shall return `E_OK`. The Initial Expiry Point on <ScheduleTableID_To> shall be processed at <ScheduleTableID_From>.Final Delay + <ScheduleTable_To>.Initial Offset ticks after the Final Expiry Point on <ScheduleTableID_From> is processed . OS324: If the input parameters are valid AND the <ScheduleTableID_From> already has a “next” schedule table then the <ScheduleTableID_To> shall replace the previous “next” schedule table and the old “next” schedule table state becomes `SCHEDULETABLE_STOPPED`. OS363: The synchronization strategy of the <ScheduleTableID_To> shall come into effect when the Operating System processes the first expiry point of <ScheduleTableID_To>.	

Caveats:	**OS453**: If the <ScheduleTableID_From> is stopped, the "next" schedule table does not start and its state changes to SCHEDULETABLE_STOPPED.
Configuration:	Available in all Scalability Classes.

8.4.12 StartScheduleTableSynchron

Service name:	StartScheduleTableSynchron	
Syntax:	`StatusType StartScheduleTableSynchron ( ScheduleTableType ScheduleTableID )`	
Service ID:	`OSServiceId_StartScheduleTableSynchronhron`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`ScheduleTableID`	Schedule table to be started
Parameters (out):	None	
Return value:	`E_OK`	No Error
	`E_OS_ID` (only in EXTENDED status)	`ScheduleTableID` not valid.
	`E_OS_STATE`	Schedule table was already started.
Description:	OS387: If the schedule table <ScheduleTableID> is not valid OR the schedule table <ScheduleTableID> is not explicitly synchronized (`OsScheduleTblSyncStrategy = EXPLICIT`) `StartScheduleTableSynchron()` shall return `E_OS_ID`. OS388: If the schedule table <ScheduleTableID> is not in the state SCHEDULETABLE_STOPPED, the service shall return `E_OS_STATE`. OS389: If <ScheduleTableID> is valid, `StartScheduleTableSynchron()` shall set the state of <ScheduleTableID> to SCHEDULETABLE_WAITING and start the processing of schedule table <ScheduleTableID> after the synchronization count of the schedule table is set via `SyncScheduleTable()`. The Initial Expiry Point shall processed when (Duration-SyncValue)+InitialOffset ticks have elapsed on the synchronization counter where: • Duration is <ScheduleTableID>.OsScheduleTableDuration • SyncValue is the <Value> parameter passed to the SyncScheduleTable() • InitialOffset is the shortest expiry point offset in <ScheduleTableID> .	
Caveats:	None	
Configuration:	Available in Scalability Classes 2 and 4.	

8.4.13 SyncScheduleTable

Service name:	SyncScheduleTable
Syntax:	`StatusType SyncScheduleTable ( ScheduleTableType ScheduleTableID, TickType Value )`

Service ID:	OSServiceId_SyncScheduleTable	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	ScheduleTableID	Schedule table
	Value	The current value of the synchronization counter.
Parameters (out):	None	
Return value:	E_OK	No errors
	E_OS_ID (only in EXTENDED status)	The ScheduleTableID was not valid or schedule table can not be synchronized (OsScheduleTblSyncStrategy not set or OsScheduleTblSyncStrategy = IMPLICIT).
	E_OS_VALUE (only in EXETENDED status)	The <Value> is out of range.
	E_OS_STATE (only in EXTENDED status)	The state of schedule table <ScheduleTableID> is equal to SCHEDULETABLE_STOPPED.
Description:	OS454: If the <ScheduleTableID> is not valid OR schedule table can not be explicitely synchronized (OsScheduleTblSyncStrategy is not equal to EXPLICIT) the service shall return E_OS_ID. OS455: If the <Value> is greater than the OsScheduleTableDuration, SyncScheduleTableAbs() shall return E_OS_VALUE. OS456: If the state of the schedule table <ScheduleTableID> is equal to SCHEDULETABLE_STOPPED or SCHEDULETABLE_NEXT the service shall return E_OS_STATE. OS457: If the parameters are valid, the service provides the operating system with the current synchronization count for the given schedule table. (It is used to synchronize the processing of the schedule table to the synchronization counter.)	
Caveats:	None	
Configuration:	Available in Scalability Classes 2 and 4.	

8.4.14 SetScheduleTableAsync

Service name:	SetScheduleTableAsync	
Syntax:	StatusType SetScheduleTableAsync (ScheduleTableType ScheduleTableID)	
Service ID:	OSServiceId_SetScheduleTableAsync	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	ScheduleTableID	Name of schedule for which status is requested
Parameters (out):	None	
Return value:	E_OK	No Error
	E_OS_ID (only in EXTENDED status)	Invalid ScheduleTableID
Description:	OS300: If OsScheduleTblSyncStrategy of <ScheduleTableID> equals to EXPLICIT then SetScheduleTableAsync() shall set the status of <ScheduleTableID> to SCHEDULETABLE_RUNNING.	

	OS362: If SetScheduleTableAsync() is called for a running schedule table the OS shall stop further synchronization until a SyncScheduleTable() call is made. OS323: If SetScheduleTableAsync() is called for a running schedule table the OS shall continue to process expiry points on the schedule table. OS458: If OsScheduleTblSyncStrategy of <ScheduleTableID> is not equal to EXPLICIT then the service call shall return E_OS_ID
Caveats:	None
Configuration:	Available in Scalability Classes 2 and 4.

8.4.15 GetScheduleTableStatus

Service name:	GetScheduleTableStatus	
Syntax:	StatusType GetScheduleTableStatus (ScheduleTableType ScheduleTableID, ScheduleTableStatusRefType ScheduleStatus)	
Service ID:	OSServiceId_GetScheduleTableStatus	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	ScheduleTableID	Schedule table for which status is requested
Parameters (out):	ScheduleStatus	Reference to ScheduleStatusType
Return value:	E_OK	No Error
	E_OS_ID (only in EXTENDED status)	Invalid ScheduleTableID
Description:	OS289: If the schedule table <ScheduleTableID> is NOT started, GetScheduleTableStatus() shall pass back SCHEDULETABLE_STOPPED via the reference parameter <ScheduleStatus> AND shall return E_OK. OS353: If the schedule table <ScheduleTableID> was used in a NextScheduleTable() call AND waits for the end of the current schedule table, GetScheduleTableStatus() shall return SCHEDULETABLE_NEXT via the reference parameter <ScheduleStatus> AND shall return E_OK. OS354: If the schedule table <ScheduleTableID> is configured with explicit synchronization AND no synchronization count was provided to the Operating System, GetScheduleTableStatus() shall return SCHEDULETABLE_WAITING via the reference parameter <ScheduleStatus> AND shall return E_OK. OS290: If the schedule table <ScheduleTableID> is started AND synchronous, GetScheduleTableStatus() shall pass back SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS via the reference parameter <ScheduleStatus> AND shall return E_OK. OS291: If the schedule table <ScheduleTableID> is started AND NOT synchronous (deviation is not within the precision interval OR the schedule table has been set asynchronous), GetScheduleTableStatus() shall pass back SCHEDULETABLE_RUNNING via the reference parameter ScheduleStatus AND shall return E_OK.	

	OS293: If the identifier <ScheduleTableID> is NOT valid, `GetScheduleTableStatus()` shall return `E_OS_ID`.
Caveats:	None
Configuration:	Available in all Scalability Classes.

8.4.16 IncrementCounter

Service name:	IncrementCounter	
Syntax:	`StatusType IncrementCounter ( CounterType CounterID )`	
Service ID:	`OSServiceId_IncrementCounter`	
Sync/Async:	Sync, may cause rescheduling.	
Reentrancy:	Yes	
Parameters (in):	`CounterID`	The Counter to be incremented
Parameters (out):	None	
Return value:	`E_OK`	No errors
	`E_OS_ID` (only in `EXTENDED` status)	The `CounterID` was not valid or counter is implemented in hardware and can not be incremented by software.
Description:	OS285: If the input parameter <CounterID> is not valid OR the counter is a hardware counter, `IncrementCounter()` shall return `E_OS_ID`. OS286: If its input parameter is valid, `IncrementCounter()` shall increment the counter <CounterID> by one (if any alarm connected to this counter expires, the given action, e.g. task activation, is done) and shall return `E_OK`. OS321: If an error happens during the execution of an alarm action, e.g. `E_OS_LIMIT` caused by a task activation, the error hook(s) are called, but the `IncrementCounter()` service itself will return `E_OK`.	
Caveats:	If called from a task, rescheduling may take place.	
Configuration:	Available in all Scalability Classes.	

8.4.17 GetCounterValue

Service name:	GetCounterValue	
Syntax:	`StatusType GetCounterValue ( CounterType CounterID, TickRefType Value )`	
Service ID:	`OSServiceId_GetCounterValue`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`CounterID`	The Counter which tick value should be read
Parameters (out):	`Value`	Contains the current tick value of the counter
Return value:	`E_OK`	No errors

	`E_OS_ID` (only in `EXTENDED` status)	The <CounterID> was not valid.
Description:	OS376: If the input parameter <CounterID> is not valid, the service should return `E_OS_ID`. OS377: If its input parameter is valid, `GetCounterValue()` shall return the current tick value of the counter via <Value> and return `E_OK`.	
Caveats:	Note that for counters of `OsCounterType` = `HARDWARE` the real timer value (the – possibly adjusted - hardware value, see OS384) is returned, whereas for counters of `OsCounterType` = `SOFTWARE` the current "software" tick value is returned.	
Configuration:	Available in all Scalability Classes.-	

8.4.18 GetElapsedCounterValue

Service name:	GetElapsedCounterValue	
Syntax:	`StatusType GetElapsedCounterValue ( CounterType CounterID, TickRefType Value, TickRefType ElapsedValue )`	
Service ID:	`OSServiceId_GetElapsedCounterValue`	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`CounterID`	The Counter to be read
	`Value`	The previously read tick value of the counter
Parameters (out):	`Value`	Contains the current tick value of the counter
	`ElapsedValue`	The difference to the previous read value
Return value:	`E_OK`	No errors
	`E_OS_ID` (only in `EXTENDED` status)	The CounterID was not valid
	`E_OS_VALUE` (only in `EXTENDED` status)	The given Value was not valid
Description:	OS381: If the input parameter <CounterID> is not valid the service will return E_OS_ID. OS391: If the <Value> is larger than the max allowed value of the <CounterID>, the service will return E_OS_VALUE. OS382: If its input parameter are valid, `GetElapsedCounterValue()` shall return the number of elapsed ticks since the given <Value> value via <ElapsedValue> and shall return E_OK. [OS460: In the <Value> parameter the current tick value of the counter is returned.	
Caveats:	If the timer already passed the <Value> value a second (or multiple) time, the result returned is wrong. The reason is that the service can not detect such a relative overflow.	
Configuration:	Available in all Scalability Classes.	

8.4.19 TerminateApplication

Service name:	TerminateApplication	
Syntax:	`StatusType TerminateApplication(RestartType RestartOption)`	
Service ID:	`OSServiceId_TerminateApplication`	
Sync/Async:	Normally does not return to the caller, except called in the wrong context: sync.	
Reentrancy:	Yes	
Parameters (in):	`RestartOption`	Either RESTART for doing a restart of the OS-Application or NO_RESTART if OS-Application shall not be restarted.
Parameters (out):	None	
Return value:	`E_OS_CALLEVEL`	Called in the wrong context.
	`E_OS_VALUE`	RestartOption is neither RESTART nor NO_RESTART.
Description:	OS287: If called from allowed context – see table [7.7.3.2.1] --, `TerminateApplication()` shall terminate the calling OS-Application (i.e. to kill all tasks, disable the interrupt sources of those OsIsrs which belong to the OS-Application and free all other OS resources associated with the application). OS288: If called from wrong context, `TerminateApplication()` shall return `E_OS_CALLEVEL`. OS459: If the <RestartOption> is invalid, the service shall retrun `E_OS_VALUE`. OS346: If RestartOption equals RESTART, `TerminateApplication()` shall activate the configured `OsRestartTask` of the terminated OS-Application.	
Caveats:	If no applications are configured the implementation shall make sure that this service is not available.	
Configuration:	Available in Scalability Classes 3 and 4.	

8.5 Hook functions

Hook functions are called by the operating system if specific conditions are met. They are provided by the user. Besides the ProtectionHook below, the hooks from [14] and/or extensions from 7.10 may be called by the OS.

8.5.1 Protection Hook

Service name:	ProtectionHook	
Syntax:	`ProtectionReturnType ProtectionHook ( StatusType Fatalerror )`	
Service ID:	Not a user service, so no ID.	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`Fatalerror`	The error which caused the call to the protection hook
Parameters (out):	None	
Return value:	`PRO_IGNORE`	The return value defines the action the OS shall take after the protection hook
	`PRO_TERMINATETASKISR`	
	`PRO_TERMINATEAPPL`	
	`PRO_TERMINATEAPPL_RESTART`	
	`PRO_SHUTDOWN`	
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: • forcibly terminate the Task/Category 2 OsIsr which causes the problem OR • forcibly terminate the OS-Application the Task/Category 2 OsIsr belong (optional with restart) OR shutdown the system OR • do nothing	
Caveats:	OS308: If an invalid value is returned the Operating System shall take the same action as if no protection hook is configured.	
Configuration:	Available in Scalability Classes 2, 3 and 4.	

8.5.2 Application specific StartupHook

Service name:	StartupHook_<App>
Syntax:	`void StartupHook_<App>(void)`
Service ID:	Not a user service, so no ID.
Sync/Async:	Sync
Reentrancy:	Yes
Parameters (in):	None
Parameters (out):	None
Return value:	None
Description:	The application specific startup hook is called during the start of the OS (after the

	user has started the OS via `StartOS()`). The hook is always called after the standard StartupHook(). If more than one OS-Application is configured which use startup hooks, the order of calls to the startup hooks of the different OS-Applications is not defined.
Caveats:	
Configuration:	Available in Scalability Classes 3 and 4.

8.5.3 Application specific ErrorHook

Service name:	ErrorHook_<App>	
Syntax:	`void ErrorHook_<App>(StatusType Error)`	
Service ID:	Not a user service, so no ID.	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`Error`	The error which caused the call to the error hook
Parameters (out):	None	
Return value:	None	
Description:	The application specific error hook is called whenever a Task or Category 2 OsIsr which belongs to the OS-Application causes an error. If the general `ErrorHook()` is configured, the general `ErrorHook()` is called before the application specific error hook is called.	
Caveats:		
Configuration:	Available in Scalability Classes 3 and 4.	

8.5.4 Application specific ShutdownHook

Service name:	ShutdownHook_<App>	
Syntax:	`void ShutdownHook_<App>(StatusType Fatalerror)`	
Service ID:	Not a user service, so no ID.	
Sync/Async:	Sync	
Reentrancy:	Yes	
Parameters (in):	`Fatalerror`	The error which caused the action to shut down the operating system.
Parameters (out):	None	
Return value:	None	
Description:	The application specific shutdown hook is called whenever the system starts the shut down of itself. If the general `ShutdownHook()` is configured, the general `ShutdownHook()` is called after all application specific shutdown hook(s) are called. If there exist more OS-Applications with an application specific shutdown hook the order of calls to these application specific shutdown hooks is not defined.	
Caveats:	Since a shutdown hook may not return to the caller it is recommended that at least all application specific shutdown hooks return to the caller in order to allow a execution of all shutdown hooks.	
Configuration:	Available in Scalability Classes 3 and 4.	

9 Sequence diagrams

9.1 Sequence chart for calling trusted functions

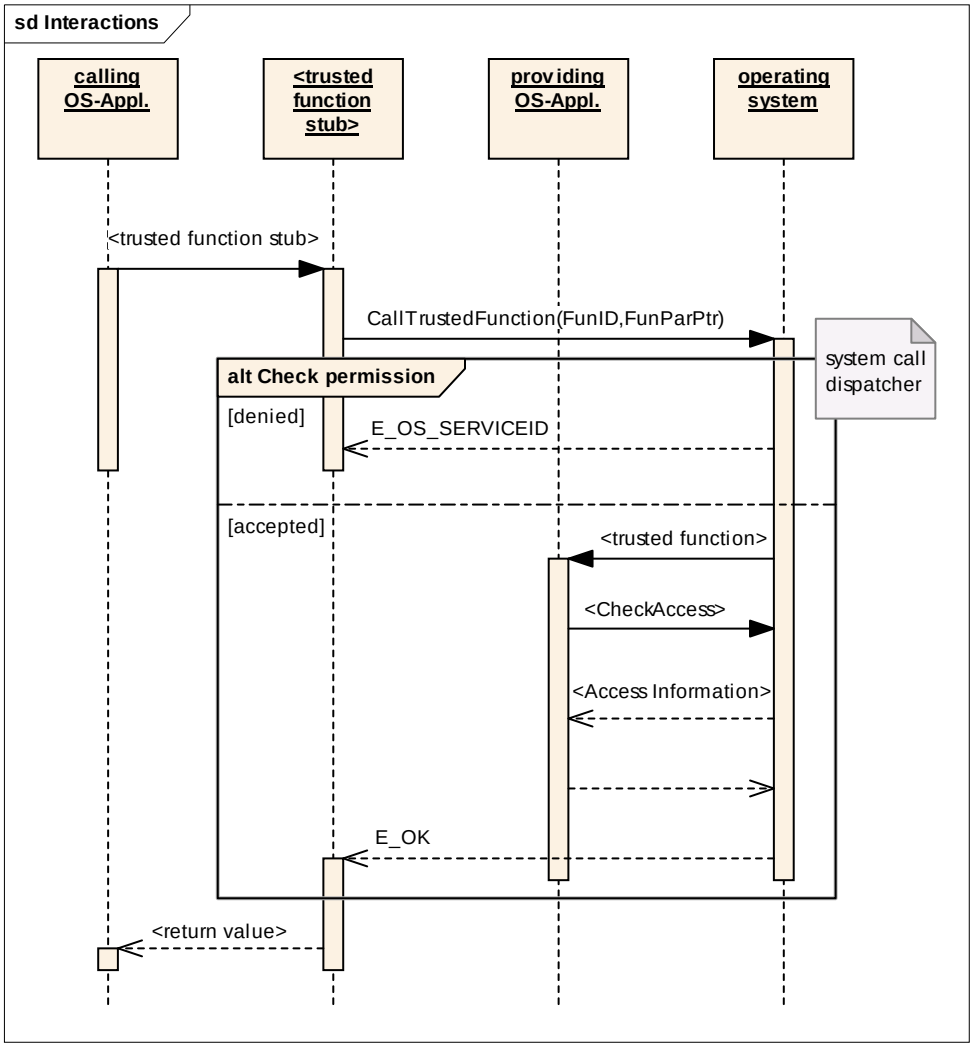


Figure 9.1: System Call sequence chart

The above sequence describes a call to the CallTrustedFunction service. It starts with a user who calls a service which requires itself a call to a trusted function. The service then packs the argument for the trusted function into a structure and calls CallTrustedFunction with the ID and the pointer as arguments. Afterwards the OS checks if the access to the requested service is valid. If no access is granted E_OS_SERVICEID is returned. Otherwise the trusted service itself is called and the function checks the arguments for access right, etc.

9.2 Sequence chart for usage of ErrorHook

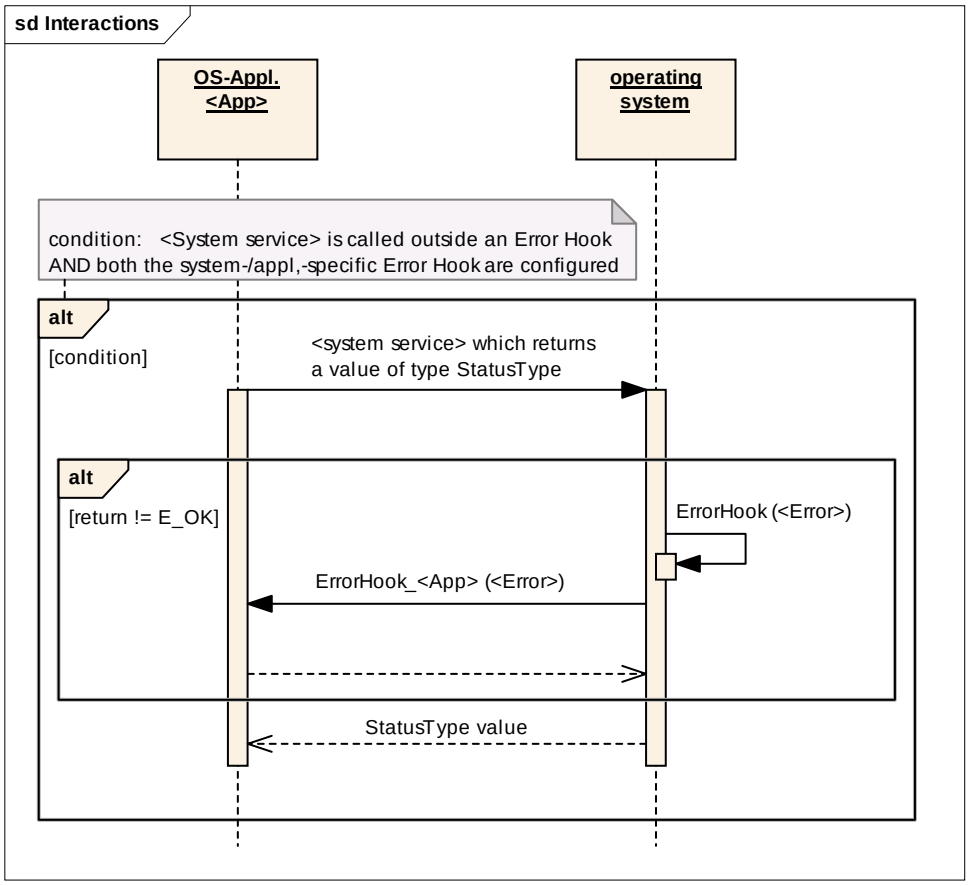


Figure 9.2: Error Hook sequence chart

The above sequence chart shows the sequence of error hook calls in case a service does not return with E_OK. Note that in this case the general error hook and the OS-Application specific error hook are called.

9.3 Sequence chart for ProtectionHook

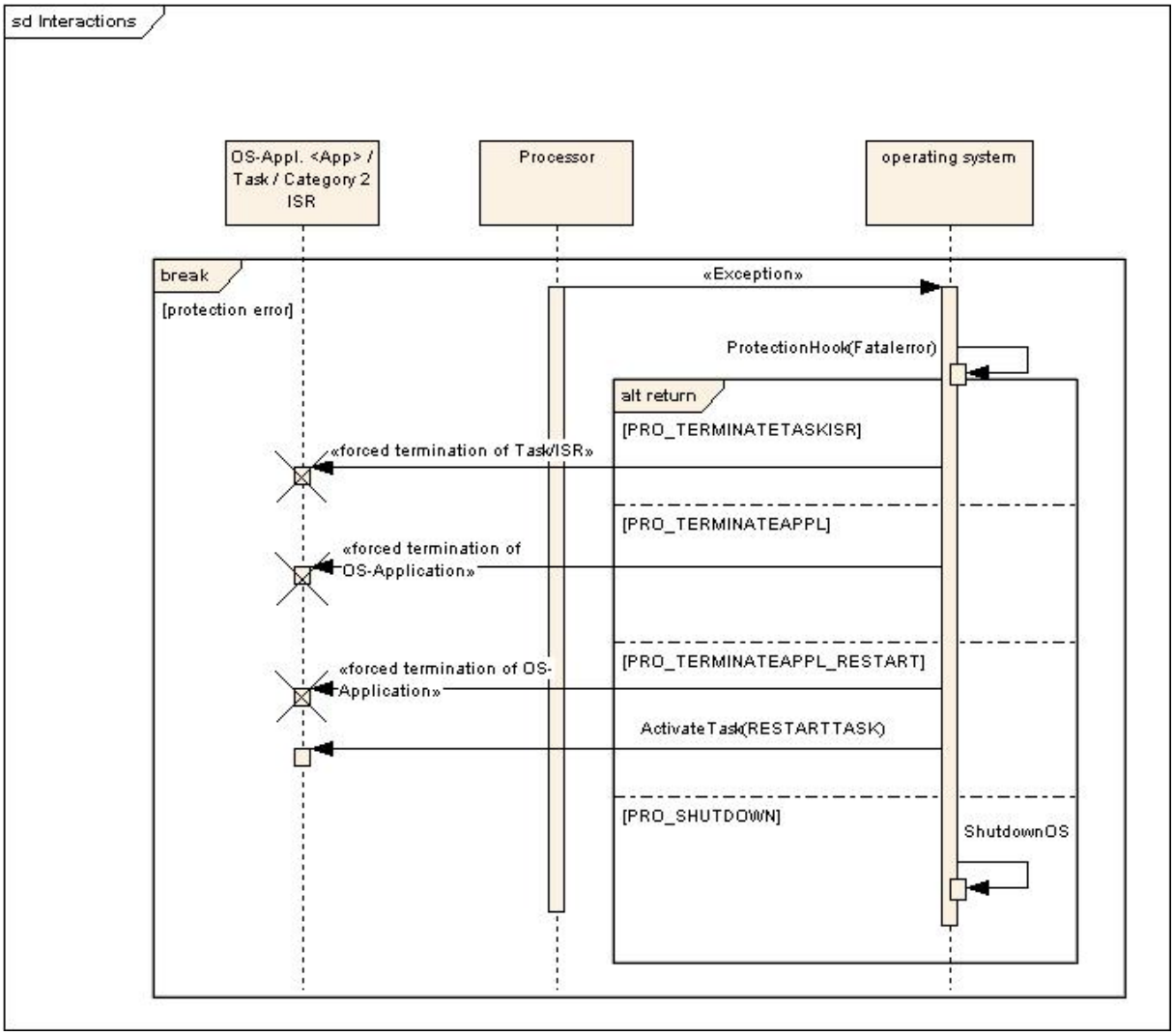


Figure 9.3: Protection Hook sequence chart

The sequence shows the flow of control if a protection error occurs. Depending on the return values of the ProtectionHook, either the faulty Task/OsIsr is forcibly terminated or the OS-Application is forcibly terminated or the system is shut down. If the action is to terminate the faulty OS-Application an option is to start afterwards the restart task, which can do a cleanup, etc.

9.4 Sequence chart for StartupHook

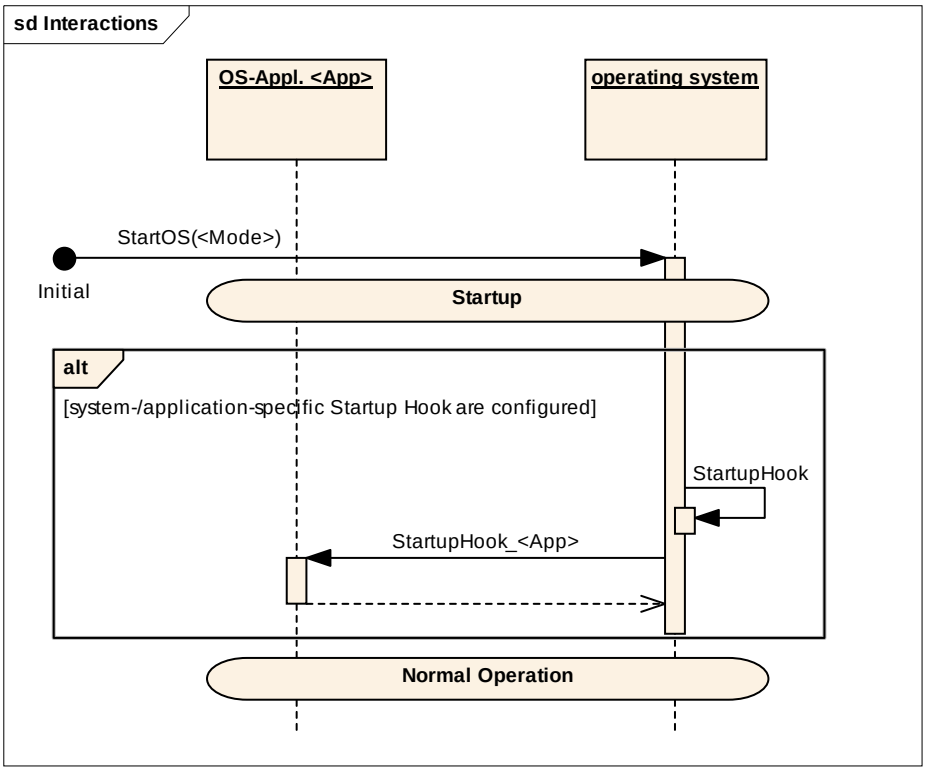


Figure 9.4: StartupHook sequence chart

The above sequence shows the flow of control during the startup of the OS. Like in OSEK OS the user calls the StartOS() service to start the OS. During the startup the startup hooks are called in the above order. The rest of the startup sequence is identical to the defined behaviour of OSEK OS.

9.5 Sequence chart for ShutdownHook

The next sequence shows the behaviour in case of a shut down. The flow is the same as in OSEK OS with the exception that the shut down hooks of the OS-Applications are called before the general ShutdownHook is called. Note that the specific shutdown hooks of the application are not allowed to block, they must return to the caller.

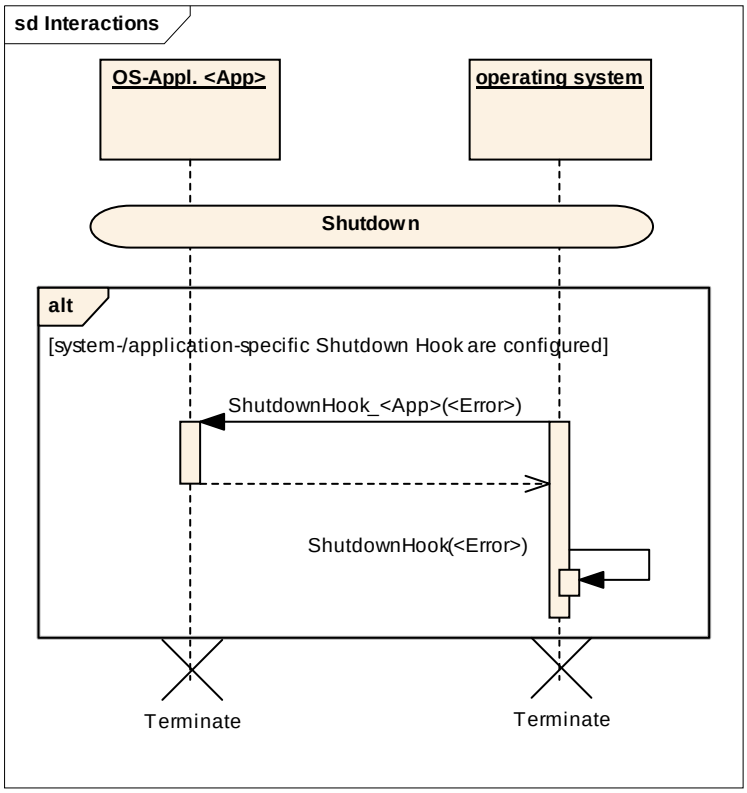


Figure 9.5: ShutdownHook sequence chart

10 Configuration Specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Os.

Chapter 10.3 specifies published information of the module Os.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [1]
- AUTOSAR ECU Configuration Specification [10]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

Note that not all attributes may be available in all scalability class.

Memory protection configuration is not standardized and therefore not part of this specification.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.4 Rules for paramters

Some configuration parameters are configured as floating point values and sometimes these values must be rounded in order to be used. The following rules define the rounding of specific parameters:

- Execution times (for the timing protection) are “round down”
- Timeframes are “round down”

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters and their containers. The detailed meanings of the parameters describe Chapters 7 and 8.

For better readability OIL names of the 2.1 OS specification are given in curly braces in the namefield of configuration parameters.

10.2.1 Os

Module Name	**Os**
Module Description	Configuration of the Os (Operating System) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsAlarm	0..*	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..*	OsAppMode is the object used to define OSEK OS properties for an OSEK OS application mode. No standard attributes are defined for AppMode. In a CPU, at least one AppMode object has to be defined. [source: OSEK OIL Spec. 2.5] An OsAppMode called OSDEFAULTAPPMODE must always be there for OSEK compatilbility.
OsApplication	0..*	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.
OsCounter	0..*	Configuration information for the counters that belong to the OsApplication.

OsEvent	0..*	Representation of OS events in the configuration context. Adopted from the OSEK OIL specification.
OsIsr	0..*	The OsIsr container represents an OSEK interrupt service routine.
OsOS	1	OS is the object used to define OSEK OS properties for an OSEK application. Per CPU exactly one OS object has to be defined.
OsResource	0..*	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.
OsScheduleTable	0..*	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..*	This container represents an OSEK task.

10.2.2 OsAlarmSetEvent

SWS Item	**--**
Container Name	OsAlarmSetEvent{SETEVENT}
Description	This container specifies the parameters to set an event
Configuration Parameters	

SWS Item	**--**		
Name	OsAlarmSetEventRef {EVENT}		
Description	Reference to the event that will be set by that alarm action		
Multiplicity	1		
Type	Reference to OsEvent		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsAlarmSetEventTaskRef {TASK}		
Description	Reference to the task that will be activated by that event		
Multiplicity	1		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.3 OsAlarm

SWS Item	**--**
Container Name	OsAlarm{ALARM}
Description	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.
Configuration Parameters	

SWS Item	**--**		
Name	OsAlarmAccessingApplication {ACCESSING_APPLICATION}		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsAlarmCounterRef {COUNTER}		
Description	Reference to the assigned counter for that alarm		
Multiplicity	1		
Type	Reference to OsCounter		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsAlarmAction	1	This container defines which type of notification is used when the alarm expires.
OsAlarmAutostart	0..1	If present this container defines if an alarm is started automatically at system start-up depending on the application mode.

10.2.4 OsAlarmAction

SWS Item	**--**
Choice Container Name	OsAlarmAction{ACTION}
Description	This container defines which type of notification is used when the alarm expires.

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
OsAlarmActivateTask	0..1	This container specifies the parameters to activate a task.
OsAlarmCallback	0..1	This container specifies the parameters to call a callback OS alarm action.
OsAlarmIncrementCounter	0..1	This container specifies the parameters to increment a counter.
OsAlarmSetEvent	0..1	This container specifies the parameters to set an event

10.2.5 OsAlarmActivateTask

SWS Item	**--**
Container Name	OsAlarmActivateTask{ACTIVATETASK}

Description	This container specifies the parameters to activate a task.
Configuration Parameters	

SWS Item	**--**		
Name	OsAlarmActivateTaskRef {TASK}		
Description	Reference to the task that will be activated by that alarm action		
Multiplicity	1		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.6 OsAlarmAutostart

SWS Item	**--**
Container Name	OsAlarmAutostart{AUTOSTART}
Description	If present this container defines if an alarm is started automatically at system start-up depending on the application mode.
Configuration Parameters	

SWS Item	**--**		
Name	OsAlarmAlarmTime {ALARMTIME}		
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		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsAlarmAutostartType		
Description	This specifies the type of autostart for the alarm..		
Multiplicity	1		
Type	EnumerationParamDef		
Range	ABSOLUTE	The alarm is started on startup via SetAbsAlarm().	
	RELATIVE	The alarm is started on startup via SetAbsAlarm.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**
Name	OsAlarmCycleTime {CYCLETIME}
Description	Cycle time of a cyclic alarm in ticks. If the value is 0 than the alarm is not cyclic.

Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsAlarmAppModeRef {APPMODE}		
Description	Reference to the application modes for which the AUTOSTART shall be performed		
Multiplicity	1..*		
Type	Reference to OsAppMode		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.7 OsAlarmCallback

SWS Item	**--**
Container Name	OsAlarmCallback{ALARMCALLBACK}
Description	This container specifies the parameters to call a callback OS alarm action.
Configuration Parameters	

SWS Item	**--**		
Name	OsAlarmCallbackName {ALARMCALLBACKNAME}		
Description	Name of the function that is called when this alarm callback is triggered.		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.8 OsAlarmIncrementCounter

SWS Item	**OS302 :**
Container Name	OsAlarmIncrementCounter{INCREMENTCOUNTER}
Description	This container specifies the parameters to increment a counter.
Configuration Parameters	

SWS Item	**--**
Name	OsAlarmIncrementCounterRef {COUNTER}

Description	Reference to the counter that will be incremented by that alarm action		
Multiplicity	1		
Type	Reference to OsCounter		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.9 OsApplication

SWS Item	**OS114 :**
Container Name	OsApplication{APPLICATION}
Description	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.
Configuration Parameters	

SWS Item	**OS115 :**		
Name	OsTrusted {TRUSTED}		
Description	Parameter to specify if an OS-Application is trusted or not. true: OS-Application is trusted false: OS-Application is not trusted (default)		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS231 :**		
Name	OsAppAlarmRef		
Description	Specifies the OsAlarms that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsAlarm		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4		

SWS Item	**OS234, OS317 :**		
Name	OsAppCounterRef		
Description	References the OsCounters that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsCounter		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS221 :**		
Name	OsAppIsrRef		
Description	references which OsIsrs belong to the OsApplication		
Multiplicity	0..*		
Type	Reference to OsIsr		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS248, OS252 :**		
Name	OsAppResourceRef		
Description	References the OsResources that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS230, OS249 :**		
Name	OsAppScheduleTableRef		
Description	References the OsScheduleTables that belong to the OsApplication.		
Multiplicity	0..*		
Type	Reference to OsScheduleTable		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS116, OS250 :**		
Name	OsAppTaskRef		
Description	references which OsTasks belong to the OsApplication		
Multiplicity	0..*		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4		

SWS Item	**OS120 :**
Name	OsRestartTask {RESTARTTASK}
Description	Optionally one task of an OS-Application may be defined as Restart Task. Multiplicity = 1: Restart Task is activated by the Operating System if the protection hook requests it. Multiplicity = 0: No task is automatically started after a protection error happened.

Multiplicity	0..1		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsApplicationHooks	1	Container to structure the OS-Application-specific hooks
OsApplicationTrustedFunction	0..*	Container to structure the configration parameters of trusted functions

10.2.10 OsApplicationHooks

SWS Item	**--**
Container Name	OsApplicationHooks
Description	Container to structure the OS-Application-specific hooks
Configuration Parameters	

SWS Item	**OS213 :**		
Name	OsAppErrorHook {ERRORHOOK}		
Description	Select the OS-Application error hook. true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS125 :**		
Name	OsAppShutdownHook {SHUTDOWNHOOK}		
Description	Select the OS-Application specific shutdown hook for the OS-Application. true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

SWS Item	**OS124 :**
Name	OsAppStartupHook {STARTUPHOOK}
Description	Select the OS-Application specific startup hook for the OS-Application. true: Hook is called false: Hook is not called
Multiplicity	1
Type	BooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4.		

No Included Containers

10.2.11 OsApplicationTrustedFunction

SWS Item	**--**
Container Name	OsApplicationTrustedFunction
Description	Container to structure the configration parameters of trusted functions
Configuration Parameters	

SWS Item	**OS254 :**		
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		
Type	FunctionNameDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 3 and 4 and in trusted OS-Applications.		

No Included Containers

10.2.12 OsAppMode

SWS Item	**--**
Container Name	OsAppMode{--}
Description	OsAppMode is the object used to define OSEK OS properties for an OSEK OS application mode. No standard attributes are defined for AppMode. In a CPU, at least one AppMode object has to be defined. [source: OSEK OIL Spec. 2.5] An OsAppMode called OSDEFAULTAPPMODE must always be there for OSEK compatilbility.
Configuration Parameters	

SWS Item	**--**
Name	OsAlarmRef {--}
Description	Optional References to autostarted OSAlarms.
Multiplicity	0..*
Type	Reference to OsAlarm

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsScheduleTableRef {--}		
Description	Optional References to autostarted OS Schedule Tables.		
Multiplicity	0..*		
Type	Reference to OsScheduleTable		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsTaskRef {--}		
Description	Optional References to autosarted Os Tasks.		
Multiplicity	0..*		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.13 OsCounter

SWS Item	**--**
Container Name	OsCounter{COUNTER}
Description	Configuration information for the counters that belong to the OsApplication.
Configuration Parameters	

SWS Item	**--**		
Name	OsCounterMaxAllowedValue {MAXALLOWEDVALUE}		
Description	Maximum possible allowed value of the system counter in ticks.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**
Name	OsCounterMinCycle {MINCYCLE}
Description	The MINCYCLE attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.
Multiplicity	1
Type	IntegerParamDef
Default value	--

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsCounterTicksPerBase {TICKSPERBASE}		
Description	The TICKSPERBASE attribute specifies the number of ticks required to reach a counterspecific unit. The interpretation is implementation-specific.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**OS255 :**		
Name	OsCounterType {TYPE}		
Description	This parameter contains the natural type or unit of the counter.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	HARDWARE	This counter is driven by some hardware e.g. a hardware timer unit.	
	SOFTWARE	The counter is driven by some software which calls the IncrementCounter service.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsSecondsPerTick		
Description	Time of one hardware tick in seconds.		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsCounterAccessingApplication {ACCESSING_APPLICATION}		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers

Container Name	Multiplicity	Scope / Dependency
OsDriver	0..1	This Container contains the information who will drive the counter. This configuration is only valid if the counter has OsCounterType set to HARDWARE. If the container does not exist (multiplicity=0) the timer is managed by the OS internally (OSINTERNAL). If the container exists the OS can use the GPT interface to manage the timer. The user have to supply the GPT channel. If the counter is driven by some other (external to the OS) source (like a TPU for example) this must be described as a vendor specific extension.
OsTimeConstant	0..*	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 later on accessed via the OsConstName. The conversation is done by rounding time values to the nearest fitting tick value.

10.2.14 OsDriver

SWS Item	OS371 :
Container Name	OsDriver{DRIVER}
Description	This Container contains the information who will drive the counter. This configuration is only valid if the counter has OsCounterType set to HARDWARE. If the container does not exist (multiplicity=0) the timer is managed by the OS internally (OSINTERNAL). If the container exists the OS can use the GPT interface to manage the timer. The user have to supply the GPT channel. If the counter is driven by some other (external to the OS) source (like a TPU for example) this must be described as a vendor specific extension.
Configuration Parameters	

SWS Item	--		
Name	OsGptChannelRef {GPTCHANNELNAME}		
Description	Reference to the GPT channel.		
Multiplicity	0..1		
Type	Reference to GptChannelConfiguration		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.15 OsEvent

SWS Item	--
Container Name	OsEvent{EVENT}
Description	Representation of OS events in the configuration context. Adopted from the OSEK OIL specification.
Configuration Parameters	

SWS Item	--
Name	OsEventMask {MASK}
Description	If event mask would be set to AUTO in OIL, this parameter should be

	omitted here.		
Multiplicity	0..1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.16 OsHooks

SWS Item	**--**
Container Name	OsHooks
Description	Container to structure all hooks belonging to the OS
Configuration Parameters	

SWS Item	**--**		
Name	OsErrorHook {ERRORHOOK}		
Description	Error hook as defined by OSEK true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsPostTaskHook {POSTTASKHOOK}		
Description	Post-task hook as defined by OSEK true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsPreTaskHook {PRETASKHOOK}		
Description	Pre-task hook as defined by OSEK true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**OS214 :**		
Name	OsProtectionHook {PROTECTIONHOOK}		
Description	Switch to enable/disable the call to the (user supplied) protection hook. true: Protection hook is called on protection error false: Protection hook is not called		
Multiplicity	0..1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2,3 and 4		

SWS Item	**--**		
Name	OsShutdownHook {SHUTDOWNHOOK}		
Description	Shutdown hook as defined by OSEK true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsStartupHook {STARTUPHOOK}		
Description	Startup hook as defined by OSEK true: Hook is called false: Hook is not called		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.17 OsIsr

SWS Item	**--**
Container Name	OsIsr{ISR}
Description	The OsIsr container represents an OSEK interrupt service routine.
Configuration Parameters	

SWS Item	**--**
Name	OsIsrCategory {CATEGORY}
Description	This attribute specifies the category of this ISR.

Multiplicity	1		
Type	EnumerationParamDef		
Range	CATEGORY_1	Interrupt is of category 1	
	CATEGORY_2	Interrupt is of category 2	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsIsrResourceRef {RESOURCE}		
Description	This reference defines the resources accessed by this ISR.		
Multiplicity	0..*		
Type	Reference to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsIsrTimingProtection	0..1	This container contains all parameters which are related to timing protection If the container exists, the timing protection is used for this interrupt. If the container does not exist, the interrupt is not supervised regarding timing violations.

10.2.18 OsIsrResourceLock

SWS Item	**OS229 :**
Container Name	OsIsrResourceLock{LOCKINGTIME}
Description	This parameter contains a list of times the interrupt uses resources.
Configuration Parameters	

SWS Item	**OS229 :**		
Name	OsIsrResourceLockBudget {MAXRESOURCELOCKTIME}		
Description	This parameter contains the maximum time the interrupt is allowed to hold the given resource (in seconds).		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS229 :**		
Name	OsIsrResourceLockResourceRef {RESOURCE}		
Description	Reference to the resource the locking time is depending on		
Multiplicity	1		
Type	Reference to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

No Included Containers

10.2.19 OsIsrTimingProtection

SWS Item	**OS326 :**		
Container Name	OsIsrTimingProtection{TIMING_PROTECTION}		
Description	This container contains all parameters which are related to timing protection If the container exists, the timing protection is used for this interrupt. If the container does not exist, the interrupt is not supervised regarding timing violations.		
Configuration Parameters			

SWS Item	**OS229 :**		
Name	OsIsrAllInterruptLockBudget {MAXALLINTERRUPTLOCKTIME}		
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	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS222 :**		
Name	OsIsrExecutionBudget {EXECUTIONTIME}		
Description	The parameter contains the maximum allowed execution time of the interrupt (in seconds).		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS229 :**		
Name	OsIsrOsInterruptLockBudget {MAXOSINTERRUPTLOCKTIME}		
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	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS223 :**		
Name	OsIsrTimeFrame {TIMEFRAME}		
Description	This parameter contains the minimum inter-arrival time between successive interrupts (in seconds).		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsIsrResourceLock	0..*	This parameter contains a list of times the interrupt uses resources.

10.2.20 OsOS

SWS Item	**--**
Container Name	OsOS{OS}
Description	OS is the object used to define OSEK OS properties for an OSEK application. Per CPU exactly one OS object has to be defined.
Configuration Parameters	

SWS Item	**OS259 :**		
Name	OsScalabilityClass {SCALABILITYCLASS}		
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. If the scalability class is omitted this translates to the OIL AUTO mechanism.		
Multiplicity	0..1		
Type	EnumerationParamDef		
Range	SC1	--	
	SC2	--	
	SC3	--	
	SC4	--	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**OS307 :**

Name	OsStackMonitoring {STACKMONITORING}		
Description	Select stack monitoring of Tasks/Category 2 ISRs true: Stacks are monitored false: Stacks are not monitored		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsStatus {STATUS}		
Description	The Status attribute specifies whether a system with standard or extended status has to be used. Automatic assigment is not supported for this attribute.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	EXTENDED	--	
	STANDARD	--	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsUseGetServiceId {USEGETSERVICEID}		
Description	As defined by OSEK		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsUseParameterAccess {USEPARAMETERACCESS}		
Description	As defined by OSEK		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**
Name	OsUseResScheduler {USERESSCHEDULER}
Description	The OsUseResScheduler attribute defines whether the resource RES_SCHEDULER is used within the application.
Multiplicity	1
Type	BooleanParamDef
Default value	true

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsHooks	1	Container to structure all hooks belonging to the OS

10.2.21 OsResource

SWS Item	**OS252 :**
Container Name	OsResource{RESOURCE}
Description	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.
Configuration Parameters	

SWS Item	**--**		
Name	OsResourceProperty {RESOURCEPROPERTY}		
Description	This specifies the type of the resource.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	INTERNAL	The resource is an internal resource.	
	LINKED	The resource is a linked resource (a second name for a existing resource).	
	STANDARD	The resource is a standard resource.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsResourceAccessingApplication {ACCESSING_APPLICATION}		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsResourceLinkedResourceRef {LINKEDRESOURCE}		
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 to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.22 OsScheduleTable

SWS Item	**OS141 :**
Container Name	OsScheduleTable{SCHEDULETABLE}
Description	An OsScheduleTable addresses the synchronization issue by providing an encapsulation of a statically defined set of alarms that cannot be modified at runtime.
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTableDuration		
Description	This parameter defines the duration (modulus) of the schedule table in ticks, i.e. the number of ticks that occur before the schedule table wraps.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**OS144 :**		
Name	OsScheduleTableRepeating {REPEATING}		
Description	true: first expiry point on the schedule table shall be processed at final expiry point delay ticks after the final expiry point is processed. false: the schedule table processing stops when the final expiry point is processed.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsSchTblAccessingApplication {ACCESSING_APPLICATION}		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**OS145 :**
Name	OsScheduleTableCounterRef {COUNTER}
Description	This parameter contains a reference to the counter which drives the schedule table.
Multiplicity	1

Type	Reference to OsCounter		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsScheduleTableAutostart	0..1	This parameter specifies if and how the schedule table is started on startup of the Operating System. The options to start a schedule table correspond to the API calls to start schedule tables during runtime.
OsScheduleTableExpiryPoint	1..*	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.

10.2.23 OsScheduleTableAutostart

SWS Item	**OS335 :**
Container Name	OsScheduleTableAutostart{AUTOSTART}
Description	This parameter specifies if and how the schedule table is started on startup of the Operating System. The options to start a schedule table correspond to the API calls to start schedule tables during runtime.
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTableAbsValue		
Description	Absolute autostart tick value when the schedule table starts. Only used if the OsScheduleTableAutostartType is ABSOLUTE.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsScheduleTableAutostartType		
Description	This specifies the type of the autostart for the schedule table.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	ABSOLUTE	The schedule table is started during startup with the StartScheduleTableAbs() service.	
	RELATIVE	The schedule table is started during startup with the StartScheduleTableRel() service.	
	SYNCHRON	The schedule table is started during startup with the StartScheduleTableSynchron() service.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsScheduleTableRelOffset		
Description	Relative offset in ticks when the schedule table starts. Only used if the OsScheduleTableAutostartType is RELATIVE.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	OsScheduleTableAppModeRef		
Description	Reference in which application modes the schedule table should be started during startup		
Multiplicity	1..*		
Type	Reference to OsAppMode		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.24 OsScheduleTableEventSetting

SWS Item	**--**
Container Name	OsScheduleTableEventSetting{SETEVENT}
Description	Event that is triggered by that schedule table.
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTableSetEventRef {EVENT}		
Description	Reference to event that will be set by action		
Multiplicity	1		
Type	Reference to OsEvent		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsScheduleTableSetEventTaskRef		
Description	--		
Multiplicity	1		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

No Included Containers

10.2.25 OsScheduleTableExpiryPoint

SWS Item	**OS143 :**
Container Name	OsScheduleTableExpiryPoint{ACTION}
Description	The point on a Schedule Table at which the OS activates tasks and/or sets events
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTblExpPointOffset		
Description	The offset from zero (in ticks) at which the expiry point is to be processed.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsScheduleTableEventSetting	0..*	Event that is triggered by that schedule table.
OsScheduleTableTaskActivation	0..*	Task that is triggered by that schedule table.
OsScheduleTblAdjustableExpPoint	0..1	Adjustable expiry point

10.2.26 OsScheduleTableTaskActivation

SWS Item	**--**
Container Name	OsScheduleTableTaskActivation{ACTIVATETASK}
Description	Task that is triggered by that schedule table.
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTableActivateTaskRef {TASK}		
Description	Reference to task that will be activated by action		
Multiplicity	1		
Type	Reference to OsTask		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.27 OsScheduleTblAdjustableExpPoint

SWS Item	**--**		
Container Name	OsScheduleTblAdjustableExpPoint		
Description	Adjustable expiry point		
Configuration Parameters			

SWS Item	**--**		
Name	OsScheduleTableMaxAdvance		
Description	The maximum positive adjustment that can be made to the expiry point offset (in ticks).		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsScheduleTableMaxRetard		
Description	The maximum negative adjustment that can be made to the expiry point offset (in ticks).		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.28 OsScheduleTableSync

SWS Item	**--**
Container Name	OsScheduleTableSync{LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION}
Description	This parameter specifies the synchronization parameters of the schedule table.
Configuration Parameters	

SWS Item	**--**		
Name	OsScheduleTblExplicitPrecision		
Description	This configuration is only valid if the explicit synchronisation is used.		
Multiplicity	0..1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: System		

SWS Item	**--**			
Name	OsScheduleTblSyncStrategy			
Description	AUTOSAR OS provides support for synchronisation in two ways: explicit and implicit.			
Multiplicity	1			
Type	EnumerationParamDef			
Range	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.		
	IMPLICIT	The counter driving the schedule table is the counter with which synchronisation is required.		
	NONE	No support for synchronisation. (default)		
ConfigurationClass	***Pre-compile time***	X	All Variants	
	Link time	--		
	Post-build time	--		
Scope / Dependency	scope: System			

No Included Containers

10.2.29 OsTask

SWS Item	**--**
Container Name	OsTask{TASK}
Description	This container represents an OSEK task.
Configuration Parameters	

SWS Item	**--**		
Name	OsTaskActivation {ACTIVATION}		
Description	This attribute defines the maximum number of queued activation requests for the task. A value equal to "1" means that at any time only a single activation is permitted for this task. Note that the value must be a natural number starting at 1.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsTaskPriority {PRIORITY}		
Description	The priority of a task is defined by the value of this attribute. This value has to be understood as a relative value, i.e. the values show only the relative ordering of the tasks. OSEK OS defines the lowest priority as zero (0); larger values correspond to higher priorities.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**--**		
Name	OsTaskSchedule {SCHEDULE}		
Description	The OsTaskSchedule attribute defines the preemptability of the task. If this attribute is set to NON, no internal resources may be assigned to this task.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FULL	Task is preemptable.	
	NON	Task is not preemptable.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsTaskAccessingApplication {ACCESSING_APPLICATION}		
Description	Reference to applications which have an access to this object.		
Multiplicity	0..*		
Type	Reference to OsApplication		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsTaskEventRef {EVENT}		
Description	This reference defines the list of events the extended task may react on.		
Multiplicity	0..*		
Type	Reference to OsEvent		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	OsTaskResourceRef {RESOURCE}		
Description	This reference defines a list of resources accessed by this task.		
Multiplicity	0..*		
Type	Reference to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsTaskAutostart	0..1	This container determines whether the task is activated during the system start-up procedure or not for some specific application modes. If the task shall be activated during the system start-up, this container is present and holds the

		references to the application modes in which the task is auto-started.
OsTaskTimingProtection	0..1	This parameter contains all parameters regarding timing protection of the task.

10.2.30 OsTaskAutostart

SWS Item	**--**
Container Name	OsTaskAutostart{AUTOSTART}
Description	This container determines whether the task is activated during the system start-up procedure or not for some specific application modes. If the task shall be activated during the system start-up, this container is present and holds the references to the application modes in which the task is auto-started.
Configuration Parameters	

SWS Item	**--**		
Name	OsTaskAppModeRef {APPMODE}		
Description	Reference to application modes in which that task is activated on startup of the OS		
Multiplicity	1..*		
Type	Reference to OsAppMode		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.31 OsTaskResourceLock

SWS Item	**--**
Container Name	OsTaskResourceLock{RESOURCELOCK}
Description	This parameter contains the worst case time between getting and releasing a given resource (in seconds).
Configuration Parameters	

SWS Item	**--**		
Name	OsTaskResourceLockBudget {RESOURCELOCKTIME}		
Description	This parameter contains the maximum time the task is allowed to lock the resource (in seconds)		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**--**

Name	OsTaskResourceLockResourceRef {RESOURCE}		
Description	Reference to the resource used by the task		
Multiplicity	1		
Type	Reference to OsResource		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

No Included Containers

10.2.32 OsTaskTimingProtection

SWS Item	**OS325 :**
Container Name	OsTaskTimingProtection{TIMING_PROTECTION}
Description	This parameter contains all parameters regarding timing protection of the task.
Configuration Parameters	

SWS Item	**--**		
Name	OsTaskAllInterruptLockBudget {MAXALLINTERRUPTLOCKTIME}		
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	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS185 :**		
Name	OsTaskExecutionBudget {EXECUTIONBUDGET}		
Description	This parameter contains the maximum allowed execution time of the task (in seconds).		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**--**
Name	OsTaskOsInterruptLockBudget {MAXOSINTERRUPTLOCKTIME}
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	FloatParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Required for scalability class 2 and 4		

SWS Item	**OS185 :**		
Name	OsTaskTimeFrame {TIMEFRAME}		
Description	The minimum inter-arrival time between activations and/or releases of a task (in seconds).		
Multiplicity	0..1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Only available in scalability class 2 and 4		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
OsTaskResourceLock	0..*	This parameter contains the worst case time between getting and releasing a given resource (in seconds).

10.2.33 OsTimeConstant

SWS Item	**OS386 :**
Container Name	OsTimeConstant{TIMECONSTANTS}
Description	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 later on accessed via the OsConstName. The conversation is done by rounding time values to the nearest fitting tick value.
Configuration Parameters	

SWS Item	**--**		
Name	OsConstName {CONSTNAME}		
Description	The name which is accessed by the application to get the above OsTimeValue.		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**
Name	OsTimeValue
Description	This parameter contains the value of the constant in seconds.
Multiplicity	1
Type	FloatParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [11] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

11 Generation of the OS

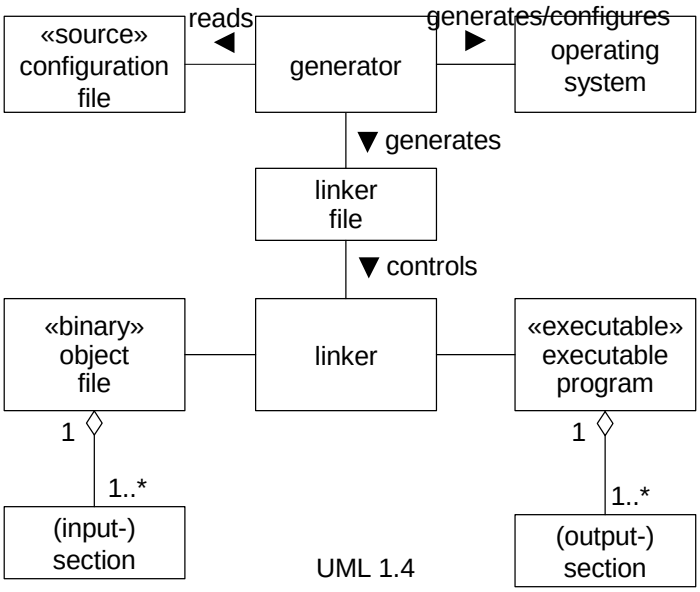


Figure 11.1: Generation Activities

11.1 Read in configuration

OS172: The generator shall provide the user the ability of reading the information of a selectable configuration file.

11.2 Consistency check

The conistency check can issue warnings or errors. Warnings mean that the generation is completed successfully, only indicating a not advisable configuration. Errors mean that the generation is not performed.

OS173: The generator shall provide the user the ability of performing a consistency check of the current configuration.

OS050: If service protection is required and `OsStatus` is not equal to `EXTENDED` (all the associated error handling is provided), the consistency check shall issue an error.

OS045: If timing protection is configured together with OSEK OS Category 1 interrupts, the consistency check shall issue a warning.

OS320: If configured attributes do not match the configured scalability class (e.g. defining an execution time budget in Tasks or Category 2 OsIsrs and selected scalability class is 1) the consistency check shall issue a warning.

OS311: If `OsScalabilityClass` is `SC3` or `SC4` AND a Task OR Category 2 OsIsr OR Resources OR Counters OR Alarms OR Schedule tables does not belong to exactly one OS-Application the consistency check shall issue an error.. RES_SCHEDULER as the only exception does not belong to any OS-Application.

OS361: If `OsScalabilityClass` is `SC3` or `SC4` AND a Category 1 OsIsr does not belong to exactly one trusted OS-Application the consistency check shall issue an error

OS177: If `OsScalabilityClass` is `SC3` or `SC4` AND an interrupt source that is used by the OS is assigned to an OS-Application, the consistency check shall issue an error.

OS303: If `OsAlarmIncrementCounter` is configured as action on alarm expiry AND the alarm is driven directly or indirectly (a cyclic chain of alarm actions with `OsAlarmIncrementCounter`) by that counter, the consistency check shall issue a warning..

OS328: If `OsStatus` is `STANDARD` and `OsScalabilityClass` is `SC3` or `SC4` the consistency check shall issue an error.

OS343: If `OsScalabilityClass` is `SC3` or `SC4` AND a task is referenced within a schedule table object AND the OS-Application of the schedule table has no access to the task, the consistency check shall issue an error.

OS344: If `OsScalabilityClass` is `SC3` or `SC4` AND a task is referenced within an alarm object AND the OS-Application of the alarm has no access to the task, the consistency check shall issue an error.

OS440: If a schedule table has `OsScheduleTblSyncStrategy = IMPLICIT` and the `OsCounterMaxAllowedValue+1` of the associated counter is not equal to the duration of the schedule table then the consitency check shall issue an error.

OS441: If a GPT channel is configured as a hardware counter driver the consitency check shall issue an error if the selected GPT channel is not configured for continuous mode.

OS461: If `OsScalabilityClass` is `SC2, SC3 or SC4 AND` Alarm Callbacks are configured the conistency check shall isuue an error.

11.3 Generating operating system

OS179: If the consistency check of the read-in configuration file has not run free of errors, the generator shall not generate/configure the operating system.

OS336: The generator shall generate a relocatable memory section containing the interrupt vector table.

OS370: The generator shall print out information about timers used internally by the OS during generation (e.g. on console, list file).

OS393: The generator shall create conversation macros to convert counter ticks (given as argument) into real time. The format of the macro is `OS_TICKS2<Unit>_<Counter>(ticks)` whereas `<Unit>` is one of `NS` (nanoseconds), `US` (microseconds), `MS` (milliseconds) or `SEC` (seconds) and `<Counter>` is the name of the counter; E.g. `OS_TICKS2MS_MyCounter()`)

12 Application Notes

12.1 Hooks

In OSEK OS, PreTask & PostTask Hooks run at the level of the OS with unrestricted access rights and therefore must be trusted. It is strongly recommended that these hook routines are only used during debugging and are not used in a final product.

When an OS-Application is killed the shutdown and startup hooks of the OS-Application are not called. Cleanup of OS-Application specific data can be done in the restart task.

All application-specific hook functions (startup, shutdown and error) must return (blocking or endless loops are not acceptable).

12.2 Providing Trusted Functions

Address checking shall be done before data is accessed. Special care must be taken if parameters passed by reference point to the stack space of a task or interrupt, because this address space might no longer belong to the task or interrupt when the address is used.

The following code fragment shows an example how a trusted function is called and how the checks should be done.

```
struct parameter_struct {type1 name1, type2 name2, StatusType
return_value};

/* This service is called by the user and uses a trusted function */

StatusType system_service(
              type1 parameter1,
              type2 parameter2)
{
    /* store parameters in a structure (parameter1 and parameter2) */
    struct  parameter_struct  local_struct;
    local_struct.name1 = parameter1;
    local_struct.name2 = parameter2;

    /* call CallTrustedFunction with appropriate index and
     * pointer to structure */
    if(CallTrustedFunction(SYSTEM_SERVICE_INDEX, &local_struct) !=
       E_OK)
         return(FUNCTION_DOES_NOT_EXIST);
    return(local_struct.return_value);
}

/* The CallTrustedFunction() service switches to the privileged
 * mode. Note that the example is only a fragment! */

StatusType CallTrustedFunction(
               TrustedFunctionIndexType        ix,
               TrustedFunctionParameterRefType ref)
{
    /* check for legal service index and return error if necessary */
    if(ix > MAX_SYSTEM_SERVICE)
        return(E_OS_SERVICEID);

    /* some implementation specific magic happens: the processor is
     * set to privileged mode */
    ….

    /*  indirectly call target function based on the index */
    (*(system-service_list[ix]))(ix, ref);

    /* some implementation specific magic happens: the processor is
     * set to non-privileged mode */
    ….

    return(E_OK);
}
```

```
/* This part of the system service is called by
 * CallTrustedFunction() */

void TRUSTED_system_service_part2 (TrustedFunctionIndexType a,
parameter_struct *local_struct)
{
    TaskRefType task;
    type1 parameter1;
    type2 parameter2;

    if (GetTaskID(&task) != E_OK)
        task = INVALID_TASK;

    /* get parameters out of the structure (parameter1 and
     * parameter2) */
    parameter1 = local_struct.name1;
    parameter2 = local_struct.name2;

    /* check the parameters if necessary */
    /* example is for parameter1 being an address and parameter2
     * being a size */
    /* example only for system_service called from tasks */
    if(GetISRID()!=INVALID_ISR)
    {
        /* error: not callable from ISR  */
        local_struct.return_value = E_OS_ACCESS;
    }
    else if(OSMEMORY_IS_WRITEABLE(CheckTaskMemoryAccess(
                task,parameter1,parameter2)))
    {
        /* system_service_part3() is now the function as it
         * would be if directly called in a non-protected
         * environment */
        local_struct.return_value =
              system_service_part3(parameter1,parameter2);
    }
    else
    {
        /* error handling */
        local_struct.return_value = E_OS_ACCESS;
    }
}
```

Note: Since the service of `CallTrustedFunction()` is very generic, it is needed to define a stub-interface which does the packing and unpacking of the arguments (as the example show). Depending on the implementation the stub interface may be (partly) generated by the generation tool.

12.3 Migration hints for OSEKtime OS users

All important OSEKtime OS features are supported in AUTOSAR OS and it should be relatively easy to port applications from OSEKtime OS to AUTOSAR OS.

However, most OSEKtime OS features are implemented slightly differently and requiring some porting effort. The following steps show how to proceed.

- Dispatcher tables can be implemented by using schedule tables provided by AUTOSAR OS. Synchronization to a global time base can be done in a similar way to OSEKtime by using the `SyncScheduleTable()` API call. A more elegant synchronization solution is also available by driving the schedule table directly from the global time source. However, the AUTOSAR OS implements priority based scheduling rather than the stack based scheduling of OSEKtime. Therefore, priorities have to be chosen for the tasks.
  If a given OSEKtime dispatcher table has to be converted, all tasks can be given the same priority as long as there are no task preemptions. If this cannot be guaranteed, in each case where a task could be pre-empted at a dispatch point, the pre-empting task must be allocated a strictly higher priority than the task it pre-empts. Usually, there are few preemptions in OSEKtime systems, so the priorities are easy to calculate – a simple monotonically increasing priority assignment relative to the tasks position in the schedule table should suffice in most cases. Caveat: In OSEKtime, it is theoretically possible that task A pre-empts task B at one point in the dispatcher table and task B pre-empts task A at another point (however, this is rarely used in practice). Such behaviour is not directly possible in AUTOSAR OS. It can, however, be emulated if required, either by constructing a simple state machine in the task bodies, or by adding two tasks A' and B' using the same code as tasks A and B respectively.
  - Deadline monitoring is not supported by AUTOSAR OS - instead, worst-case execution time enforcement is provided. Schedulability analysis can be used to calculate whether given deadlines are met in a system of periodic tasks with given worst-case execution times.
  - Reenabling of interrupts defined offline is not supported by AUTOSAR OS.
  - Tasks that have precedence over interrupt service routines are not supported by AUTOSAR OS, however, this behaviour can be easily emulated by activating a low-priority task from an OsIsr.
  - Smooth synchronization is achieved by adjusting the delay between adjacent expiry points, generalising OSEKtime OS' approach, where the synchronization of the local time to the global time is done during several dispatcher rounds by extending or shortening the last ground state of the dispatcher round.

The OSEK time specification allows dispatcher rounds to take 3 modes:

1. Synchronous
2. Asynchronous/Hard
3. Asynchronous/Smooth

Users of OSEKtime who are migrating the AUTOSAR OS can define a schedule table that has the same range/tick resolution as their global time source (with an accompanying AUTOSAR OS counter that has the same resolution as the global time) and can synthesise these modes as follows:

1. Synchronous: Define `OsScheduleTblSyncStrategy = IMPLICIT` and start using `StartScheduleTableAbs()`. Or define `OsScheduleTblSyncStrategy = EXPLICIT` and start using `StartScheduleTableSynchron()`

2. Asynchronous/Hard: Define `OsScheduleTblSyncStrategy = EXPLICIT` and specify that the final expiry point on the schedule table has a `OsScheduleTableMaxRetard = 1` and a `OsScheduleTableMaxAdvance = OsCounterMaxAllowedValue`. Start using `StartScheduleTableRel()`.

3. Asynchronous/Smooth: Define `OsScheduleTblSyncStrategy = EXPLICIT` and specify that each expiry point on the schedule table has `OsScheduleTableMaxRetard = 1` and a `OsScheduleTableMaxAdvance < OsCounterMaxAllowedValue`. Start using `StartScheduleTableRel()`.

12.4 Software Components and OS-Applications

Trusted OS-Applications can be permitted access to IO space. As software components can not be allowed direct access to the hardware, software components can not be trusted OS-Applications because this would violate this protection feature. The configuration process must ensure that this is the case.

The AUTOSAR Virtual Function Bus (VFB) specification places no restrictions on how runnables from software components are mapped to OS tasks. However, because the protection mechanisms in AUTOSAR OS apply only to OS managed objects. This means that all runnables in a task:

- Are not protected from each other at runtime
- Share the same protection boundary

If runnables need to be protected they must therefore be allocated to different tasks and those tasks protected accordingly.

A simple rule can suffice:

> *"When allocating runnables to tasks, only allocate runnables from the same software component into the same task or set of tasks."*

If multiple software components from the same application are to reside on the same processor, then, assuming protection is required between applications (or parts thereof) on the same processor, this rule could be modified to relax the scope of protection to the application:

> *"When allocating runnables to tasks, only allocate runnables from the same application into the same task or set of tasks."*

If an OS-Application is killed and the restart task is activated it can not assume that the startup of the OS-Application has finished. Maybe the fault happened in the application startup hook and no task of the application was started so far.

12.5 Global Time Synchronization

The OS currently assumes that the global time synchronization is done by the user (unless implicit synchronization is used). This allows maximum flexibility regarding the time source. For synchronization with e.g. FlexRay some glue code may be necessary which transfer the information from the time source to the OS.

12.6 Working with FlexRay

Schedule tables in the AUTOSAR OS may be synchronized with a global (network) time provided by FlexRay in essentially two ways:

1. Using the FlexRay interface's services for controlling timer interrupts related to global time to provide a "hardware" counter tick source to drive the processing of a schedule table (implicit synchronization)
2. Using the FlexRay interface's service for accessing the current global time and passing this into the OS through the SyncScheduleTable() OS service call

This section looks at the second option only.

In FlexRay time is presented as a tuple of a Cycle and a MacrotickOffset within the cycle. Cycle is an 8-bit value and MacrotickOffset is a 16-bit value.
In AUTOSAR OS a schedule table is associated with an underlying counter that has a notion of ticks. It is therefore possible to synchronize with either the Cycle or the tuple of Cycle/MacrotickOffset to give the resolution of synchronization required by the application.
If Cycle only resolution is require then an OS COUNTER object should be configured that has a OsCounterMaxAllowedValue equal to the maximum number of Cycles. If Cycle/MacrotickOffset is required then an OS COUNTER object should be configured with a OsCounterMaxAllowedValue of the maximum number of Cycles multiplied by the MacrotickOffset. This provides the OS with a time base against which a ScheduleTable can be synchronized.

Synchronization between the OS and an external global time source is provided by telling the OS the global time through the SyncScheduleTable() service call. This call takes a scalar parameter of TickType so to interface this to FlexRay's representation of time a small conversion needs to be done. The following example assumes a Cycle of 255 with 65535 Macroticks per Cycle. TickType is at least 24-bits wide.

```
#define OSTIME(x) (TickType)(x);
FrIf_GetGlobalTime(Controller, &Cycle, &Macrotick);
SyncScheduleTable(Tbl, ((OSTIME(Cycle) << 16)+(OSTIME(Macrotick))));
```

Telling the ScheduleTable that GlobalTime can be done when the application detects that the FlexRay controller has lost synchronization with the network (by polling the controller sync status). The following code indicates how this can be used to force an associated ScheduleTable into the `SCHEDULETABLE_RUNNING` state from the `SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS` state.

```
Fr_SyncStateType CurrentSyncStatus;
if (FrIf_GetSyncState(Controller, &CurrentSyncStatus) == E_OK) {
  if (CurrentSyncStatus == FR_ASYNC ) {
    SetScheduleTableAsync(Table);
  }
}
```

Of course, other actions are possible here, like stopping the ScheduleTable, as best fits user requirements.

12.7 Migration from OIL to XML

This version of the AUTOSAR OS specification does not directly support the configuration via OIL. The support for OIL was dropped in favour of XML because XML is the standard configuration language in AUTOSAR and is essential if configuration data has to be imported / exported from / to other AUTOSAR modules or between different tools during development.

Since OIL and XML are both ASCII formats a tool vendor may offer a possibility to import (old) OIL files and to store them as (AUTOSAR OS) XML files. Currently all known vendors support at least the import of existing OIL configurations.

Note that for showing conformance to the OSEK OS specification, each OSEK OS vendor must support OIL. This means that practically each AUTOSAR OS vendor will offer some sort of import of OIL configurations – at least to show the OSEK OS conformance.

13 AUTOSAR Service implemented by the OS

13.1 Scope of this Chapter

This chapter is an addition to the specification of the Operating System. Whereas the other parts of the specification define the behavior and the C-interfaces of the OS module, this chapter formally specifies the corresponding AUTOSAR Service in terms of the SWC Template. The interfaces described here will be visible on the VFB and are used by the RTE generator to create the glue code between the application software (SWC) and the OS.

13.1.1 Package

The following definitions are interpreted to be in
`ARPackage AUTOSAR/Services/Os`

13.2 Overview

The AUTOSAR Operating System is normally not used directly by SWCs. Even the other BSW modules which are below the RTE are using the BSW Scheduler to have access to OS services. The BSW Scheduler of courses uses the OS to implement its features, e.g. critical sections.
Nevertheless there is one use case, where it makes sense to allow SWCs access to services of the OS: timer services. Since the number of timers in an ECU is limited it make sense to share these units across several SWCs. The functionality of the timer services of the OS which are offered to the SWCs are:

- A service to get the current value of a – hardware or software – counter
- A service which calculates the time difference between the current timer value and a given (previouls read) timer value

13.3 Specification of the Ports and Port Interfaces

This chapter specifies the ports and port interfaces which are needed in order to operate the timer services of the OS over the VFB. Note that there are ports on both sides of the RTE: The SW-C description of the OS service will define the ports below the RTE. Each SW-Component, which uses the Service, must contain “service ports” in its own SW-C description which will be connected to the ports of the OS, so that the RTE can be generated.

13.3.1 Data Types and Port Interface

13.3.1.1 General Approach

It is appropriate to model the requests issued from a client to the timer services by ports using the client/server interfaces.

13.3.1.2 Data Types

This chapter describes the data types which will be used in the port interfaces for timer service requests. In general the timer interfaces are using the following types:

- `CounterType` – This type is the reference to the requested Counter
- `TickType` – This type holds a timer value
- `TickRefType` – This is a reference (pointer) to a `TickType`

The APIs of the timer services have a return type of `StatusType`. This means that a successful call returns 0 and a return value not equal 0 represents an error.

13.3.1.3 Port Interface

The operations correspond to the function calls of the OS C-API (notation in pseudo code; must be transferred into XML).

The notation of possible error codes resulting from server calls follows the approach in the meta-model. It is a matter of the RTE specification [9], how those error codes will be passed via the actual API.

```
ClientServerInterface OsService {
     PossibleErrors {
          E_NOT_OK <> 0
     };

     // The timer services

      GetCounterValue(       IN  CounterType  CounterID,
                             OUT TickRefType  Value);

      GetElapsedCounterValue(IN  CounterType  CounterID,
                             INOUT  TickType  PreviousValue,
                             OUT TickRefType  Value
};
```

13.3.1.4 Ports

We end up with the following structure for the AUTOSAR Interface of the OS:

```
Service Os
{
      ProvidePort OsService OsTimerService;
};
```

It is obvious that the existence of all these port definitions depends on the ECU.

14 Outlook on Memory Protection Configuration

As stated before, memory protection configuration is not standardized yet. Nevertheless it seems helpful to contribute a recommendation in this chapter, how the configuration might work.

14.1 Configuration Approach

Both, SW-Components and BSW modules, map code and variables to dedicated, disjoined memory sections (see meta-class»ObjectFileSection« in chapter 7.3 of »Software Component Template«, Version 2.0.1, and »module specific sections« in chapter 8.2 of »Specification of Memory Mapping«, Version 1.0.1).

This essential precondition (avoid an inseparable conglomeration of variables in the default section) can be used to support configuration of memory protection domains:

1. The generator can save for each OS-Application a (processor-specific) maximum number of output sections for data in a file (to be used in the linker file).

2. The generator can uniquely identify the address spaces of the data output sections with symbols using the naming convention (see »memory allocation keywords« `_STOP_SEC_VAR` and `_START_SEC_VAR` for start and stop symbols) in the specification mentioned above.

The input data sections in the object files of an OS-Application can then be assigned to the output sections (with potential tool support). Usually, this is one segment for global data, and one segment for code.

To archieve portability, the user shall group all variables belonging to a private data section (Task/OsIsr or OS-Application) in separate files.

15 Changes to Release 1

This chapter contains all major changes to the previous release. Changes made are either based on bugzilla entries or on proposals which were presented during work group meetings. Note that small changes or typos or reformatting are not listed.

15.1 Deleted SWS Items

SWS Item	***Rationale***
Chapter 4.5	Deleted completely
Chapter 7	OS260 (in chapter 7.3), OS305 (in chapter 7.7)
Chapter 8	Deleted `StartScheduleTable()`
Chapter 11	OS174 (in chapter 11.2), OS178 (in chapter 11.3)

15.2 Replaced SWS Items

SWS Item of Release 1	***replaced by SWS Item***	***Rationale***
Rational and background of chapter 7.3	New rational and background	Replaced complete description which also contains now an example how relative/absolute starts of schedule tables influences the synchronization

15.3 Changed SWS Items

SWS Item	***Rationale***
Glossary of Terms	Adapted some terms to the new specification/meaning.
Chapter 6	Updated tables to reflect new API and added missing requirements
Chapter 7	Clarified and extended synchronization of schedule tables (OS206, OS200). Changed handling of memory protection for Task/Category 2 OsIsr private stack/data within an OS-Application to an optional feature (OS208, OS195). Changed arrival rate handling: For Tasks the model changed from a state-based to a time budget behavior. For Category 2 OsIsrs two APIs were introduced to allow a save implementation. Exclude Category 1 Interrupts from some requirements (e.g. OS088) Restrict Scalability Class 3 and 4 to EXTENDED mode. Service protection makes no sense in standard mode. The interrupt locking time for Tasks/Category 2 OsIsrs is split into two timings: One for the time a Task/Category 2 OsIsr disables all interrupts and one time where only the Category 2 interrupts are disabled. Changed several requirments to fit to new configuration parameters.
Chapter 8	Improved wording of constants for schedule table status type. Added argument for `TerminateApplication()` to allow a restart. Removed 3rd argument of `SyncScheduleTable()` (now obsolete).

Chapter 12	Updated the OIL example to new attributes.

15.4 Added SWS Items

SWS Item	***Rationale***
Glossary of Terms	Added some new terms which are now covered by the SWS, e.g. Interrupt Vector Table.
Chapter 7	Added new figure to 7.2 explaining the start of a schedule table.
Chapter 8	Added new APIs: o `StartScheduleTableRel()` o `StartScheduleTableAbs()` o `DisableInterruptSource()` o `EnableInterruptSource()` Extended `GetScheduleTableStatus()`
Chapter 10	Added containers from new SWS template. These contain now also the OIL attribute and their meaning.
Chapter 11	Added some additional consistency checks (OS343, OS344, OS345).
Chapter 13	Added more explanations to 13.4 covering the migration from OSEKtime to AUTOSAR OS.

16 Changes to Release 2.1

- Rewrote schedule table and synchronization chapter to improve description and to make some requirments more explicit
- Support for XML, OIL is dropped
- Updates of figures
- Small correction (wording, typos, clarifications)
- Added service interface for SWCs calling the timer services