HYUNDAI AUTOEVER

AUTOSAR Os User Manual

DOC. NO

SCOPE OF APPLICATION All Project/Engineering
Responsibility: Classic AUTOSAR Team

File Name	Os_UM.docx
Creation	Yangjin Kim
Check	Seunghoon Yoo
Approval	Jiseok Song
Edition Date	2024-11-29
Document Management System	

HYUNDAI AutoEver

Document Change History

Date (YYYY-MM-DD)	Ver.	Editor	Chap	Description (before -> after revision)
2019-12-24	1.0.0.0	MJ.Woo	All	Initial Version
2020-03-10	1.1.0.0	MJ.Woo	10.9	Added NMI callback function
2020-07-03	2.0.0.0	JH.Cho	4 5.1 10.10	Update Change Log and Limitation Change OsSystemTimerClock category Add Stack configuration
2020-08-14	2.0.1.1	JH.Cho	4	Update Change Log
2020-12-11	2.1.0.0	JH.Cho	4 5.10 5.13 5.14 6.9 6.16 6.17 6.18 10.6 10.11	Update Change Log Support configuartion of os-application Support configuartion of spinlock Support configuartion of ioc Support API of os-application Support API of spinlock Support API of multicore Support API of ioc Support appendix of os-application configuration Add Multi-Core configuration
2021-01-06	2.2.0.0	JH.Cho	4.3.1 4.3.2	Update Change Log Add cautions and limitation for multicore
2021-02-19	2.3.0.0	YH.Han	4.3.1 4.3.2 5.11 6.10 10.7	Update Change Log Add limitation for SC2 Support configuartion of timing protection Support API of timing protection Support appendix of timing protection configuration
2021-09-02	2.4.0.0	YH.Han	All 4.3.1 4.3.2 5.1.1 5.1.3 6.12	Apply the new template Update Change Log Add limitation for SC3 and FPU Add OsStackFrame and OsFpuSupport Add OsMpuRegion container Support Memory Protection
2022-01-21	2.4.1.0	YH.Han	4.3.1 5.1.1	Update Change Log Change category for OsFpuSupport
2022-03-18	2.4.2.0	YH.Han	4.2 4.3.1 4.3.2 5.2.1 5.10.1	Update Change Log Change category for OsFpuSupport Add stack related limitation when using SC3/4.
2022-08-10	2.4.3.0	JC.Kim	4.2 4.3.1	Update Change Log
2022-11-24	2.4.4.0	YH.Han	4.2 4.3.1	Update Change Log

2022-12-19	2.4.4.1	JC.Kim	4.3.1 4.3.2 5.1.3	Add limitation Udpate Change Log
2023-03-21	2.4.4.2	YH.Han	4.3.1	Updagte Change Log
2024-06-28	2.4.5.0	KhanhPQ8	4.3.1	Update Change Log
2024-11-29	2.4.5.0_HF1	Yangjin Kim	4.3.1	Update Change Log

Table of Contents

1. OVERVIEW ..... - 8 -

2. REFERENCE ..... - 8 -

3. AUTOSAR SYSTEM ..... - 9 -

3.1 OSEK OS AND AUTOSAR OS ..... - 9 -
3.2 START UP SEQUENCE ..... - 10 -
3.3 INTERACTION BETWEEN OS MODULE WITH OTHER MODULES ..... - 11 -
3.3.1 Interaction between OS Module and RTE ..... - 11 -
3.3.2 Interaction between OS Module and BSW Scheduler ..... - 11 -
3.4 CONFORMANCE CLASSES ..... - 12 -
3.5 SCALABILITY CLASSES ..... - 13 -
3.6 APPLICATION PROGRAM INTERFACE ..... - 14 -
3.7 IMPORTED TYPES ..... - 15 -
3.7.1 Standard Types ..... - 15 -
3.8 OPERATING SYSTEM EXECUTION CONTROL ..... - 15 -
3.8.1 StartOS ..... - 15 -
3.8.2 ShutdownOS ..... - 16 -

4. PRODUCT RELEASE NOTES ..... - 17 -

4.1 OVERVIEW ..... - 17 -
4.2 SCOPE OF THE RELEASE ..... - 17 -
4.3 MODULE RELEASE NOTES ..... - 17 -
4.3.1 Change Log ..... - 17 -
4.3.2 Limitation ..... - 23 -

5. CONFIGURATION GUIDE ..... - 24 -

5.1 GENERAL ..... - 24 -
5.1.1 OsOS ..... - 24 -
5.1.2 OsOs - OsHooks ..... - 25 -
5.1.3 OsOs - OsMpuRegion ..... - 26 -
5.2 TASK MANAGEMENT ..... - 28 -
5.2.1 OsTask ..... - 28 -
5.2.2 OsTask - OsTaskAutostart ..... - 29 -
5.2.3 OsTask - OsTaskTimingProtection ..... - 29 -
5.2.4 OsTask - OsTaskTimingProtection - OsTaskResourceLock ..... - 29 -
5.3 INTERRUPT PROCESSING ..... - 30 -
5.3.1 OsIsr ..... - 30 -
5.3.2 OsIsr - OsIsrTimingProtection ..... - 30 -
5.3.3 OsIsr - OsIsrTimingProtection - OsIsrResourceLock ..... - 31 -

5.4 RESOURCE MANAGEMENT - 31 -
5.4.1 OsResource - 31 -
5.5 COUNTER - 33 -
5.5.1 OsCounter - 33 -
5.5.2 OsCounter - OsDriver - 34 -
5.5.3 OsCounter - OsTimeConstant - 34 -
5.6 ALARM - 34 -
5.6.1 OsAlarm - 34 -
5.6.2 OsAlarm - OsAlarmAction - 35 -
5.6.3 OsAlarm – OsAlarmAction - OsAlarmActivateTask - 35 -
5.6.4 OsAlarm – OsAlarmAction - OsAlarmCallback - 35 -
5.6.5 OsAlarm – OsAlarmAction - OsAlarmIncrementCounter - 35 -
5.6.6 OsAlarm – OsAlarmAction - OsAlarmSetEvent - 35 -
5.6.7 OsAlarm - OsAlarmAutostart - 36 -
5.7 EVENT MECHANISM - 36 -
5.7.1 OsEvent - 36 -
5.8 SCHEDULE TABLE - 37 -
5.9 APPLICATION MODES - 37 -
5.9.1 OSAppMode - 37 -
5.10 OSAPPLICATION - 37 -
5.10.1 OsApplication - 37 -
5.10.2 OsApplication - OsApplicationHooks - 38 -
5.10.3 OsApplication - OsApplicationTrustedFunction - 38 -
5.10.4 OsApplication - OsMemoryBlock - 38 -
5.11 TIMING PROTECTION - 39 -
5.11.1 OsTaskTimingProtection - 39 -
5.11.2 OsTaskResourceLock - 39 -
5.11.3 OsIsrTimingProtection - 39 -
5.11.4 OsIsrResourceLock - 39 -
5.12 STACK MONITORING - 39 -
5.12.1 OsStackMonitoring - 39 -
5.13 SPINLOCK - 40 -
5.13.1 OsSpinlock - 40 -
5.14 IOC - 40 -
5.14.1 Osloc - 40 -
5.14.2 Osloc - OslocCommunication - 40 -
5.14.3 Osloc - OslocCommunication - OslocDataProperties - 41 -
5.14.4 Osloc - OslocCommunication - OslocReceiverProperties - 41 -
5.14.5 Osloc - OslocCommunication - OslocSenderProperties - 42 -
5.15 DEVIATION - 42 -

6. APPLICATION PROGRAMMING INTERFACE (API) ..... - 43 -
6.1 TASK MANAGEMENT ..... - 43 -
6.1.1 General ..... - 43 -
6.1.2 Data Types ..... - 45 -
6.1.3 Run Time Services ..... - 45 -
6.2 SCHEDULER ..... - 51 -
6.2.1 General ..... - 51 -
6.2.2 Data types ..... - 53 -
6.2.3 Run Time Services ..... - 53 -
6.3 INTERRUPT PROCESSING ..... - 53 -
6.3.1 General ..... - 53 -
6.3.2 Data Types ..... - 54 -
6.3.3 Run Time Services ..... - 54 -
6.4 RESOURCE MANAGEMENT ..... - 62 -
6.4.1 General ..... - 62 -
6.4.2 Data Types ..... - 63 -
6.4.3 Run Time services ..... - 63 -
6.5 COUNTERS ..... - 65 -
6.5.1 General ..... - 65 -
6.5.2 Data Types ..... - 65 -
6.5.3 Run Time services ..... - 65 -
6.6 ALARM ..... - 69 -
6.6.1 General ..... - 69 -
6.6.2 Data Types ..... - 70 -
6.6.3 Run Time services ..... - 70 -
6.7 EVENT MECHANISM ..... - 76 -
6.7.1 General ..... - 76 -
6.7.2 Data Types ..... - 77 -
6.7.3 Run Time services ..... - 77 -
6.8 APPLICATION MODES ..... - 82 -
6.8.1 General ..... - 82 -
6.8.2 Data Types ..... - 82 -
6.8.3 Run Time services ..... - 82 -
6.9 OS-APPLICATION ..... - 83 -
6.9.1 General ..... - 83 -
6.9.2 Data Types ..... - 84 -
6.9.3 Run Time services ..... - 87 -
6.10 TIMING PROTECTION ..... - 95 -
6.10.1 General ..... - 95 -
6.10.2 Data Types ..... - 95 -

6.10.3 Run Time services - 95 -
6.11 SERVICE PROTECTION - 96 -
6.11.1 General - 96 -
6.11.2 Data Types - 96 -
6.11.3 Run Time services - 96 -
6.12 MEMORY PROTECTION - 98 -
6.12.1 General - 98 -
6.12.2 Data Types - 98 -
6.12.3 Run Time servicesNA - 99 -
6.13 STACK MONITORING - 99 -
6.13.1 General - 99 -
6.13.2 Data TypesNA - 99 -
6.13.3 Run Time services - 99 -
6.14 HOOK ROUTINE - 99 -
6.14.1 General - 99 -
6.14.2 Data Types - 100 -
6.14.3 Run Time services - 103 -
6.15 PLATFORM SPECIFIC EXTENSIONS - 106 -
6.16 SPINLOCK - 106 -
6.16.1 General - 106 -
6.16.2 Data Types - 106 -
6.16.3 Run Time services - 107 -
6.17 MULTICORE - 111 -
6.17.1 General - 111 -
6.17.2 Data Types - 111 -
6.17.3 Run Time services - 111 -
6.18 IOC - 116 -
6.18.1 General - 116 -
6.18.2 Data Types - 116 -
6.18.3 Run Time services - 117 -

7. GENERATOR - 127 -

7.1 GETTING STARTED - 127 -
7.2 SAMPLE USAGE - 129 -
7.3 AUTOSAR OS GENERATION TOOL OVERVIEW - 131 -
7.4 TOOL INSTALLATION REQUIREMENTS - 131 -
7.4.1 Hardware requirements - 132 -
7.4.2 Software requirements - 132 -
7.4.3 Limitations - 132 -
7.5 TOOL INSTALLATION - 132 -
7.5.1 Pre Requisite - 132 -

7.5.2 Installation Steps ..... - 132 -
7.6 TOOL UNINSTALLATION ..... - 132 -
7.7 INPUT FILES ..... - 133 -
7.8 PRECAUTIONS ..... - 133 -
7.9 USER CONFIGURATION VALIDATION ..... - 134 -
8. BSWMD ..... - 135 -
8.1 BSW MDT PARAMETER CONFIGURATION ..... - 135 -
9. EXCLUSIVE AREAS ..... - 135 -
10. APPENDIX ..... - 136 -
10.1 TASK CONFIGURATION ..... - 136 -
10.2 ISR CONFIGURATION ..... - 138 -
10.3 ALARM CONFIGURATION ..... - 139 -
10.4 RESOURCE CONFIGURATION ..... - 141 -
10.5 EVENT CONFIGURATION ..... - 142 -
10.6 OS-APPLICATION CONFIGURATION ..... - 143 -
10.7 TIMING-PROTECTION CONFIGURATION ..... - 145 -
10.8 PROTECTIONHOOK CONFIGURATION ..... - 146 -
10.9 NMI CALLBACK FUNCTION ..... - 147 -
10.10 STACK CONFIGURATION ..... - 148 -
10.11 MULTI-CORE CONFIGURATION ..... - 151 -

1. Overview

This document is created based on AUTOSAR standard SRS/SWS.
For details functional description, please refer to the Reference Documents.

The source code of this module is permitted to be used only in projects contracted with Hyundai Autoever, and any other use is prohibited.
If you use it for other purposes or change the source code, you may take legal responsibility.
In this case, There is no warranty and technical support.

The following terms mean:

- Changeable : Can config by user
- Fixed : Can not change this configuration by user
- Not Supported : Can not use this configuration

SC is the abbreviation of 'Scalability Class'. Please refer to 3.5 Scalability Classes for the detailed information.

2. Reference

Sl. No.	Title	Version
1.	SRS_OS.pdf	V4.0.3
2.	SWS_OS.pdf	V4.0.3
3.	Os223.pdf (OSEK Os)	2.2.3

3. AUTOSAR System

3.1 OSEK OS and AUTOSAR OS

The relation between OSEK OS and AUTOSAR OS is shown in figure 1. The AUTOSAR OS includes all the functions of OSEK OS and basically has upward compatibility.

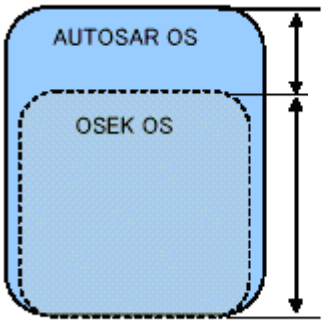


Figure 1: Relation between OSEK OS and AUTOSAR OS

3.2 Start Up Sequence

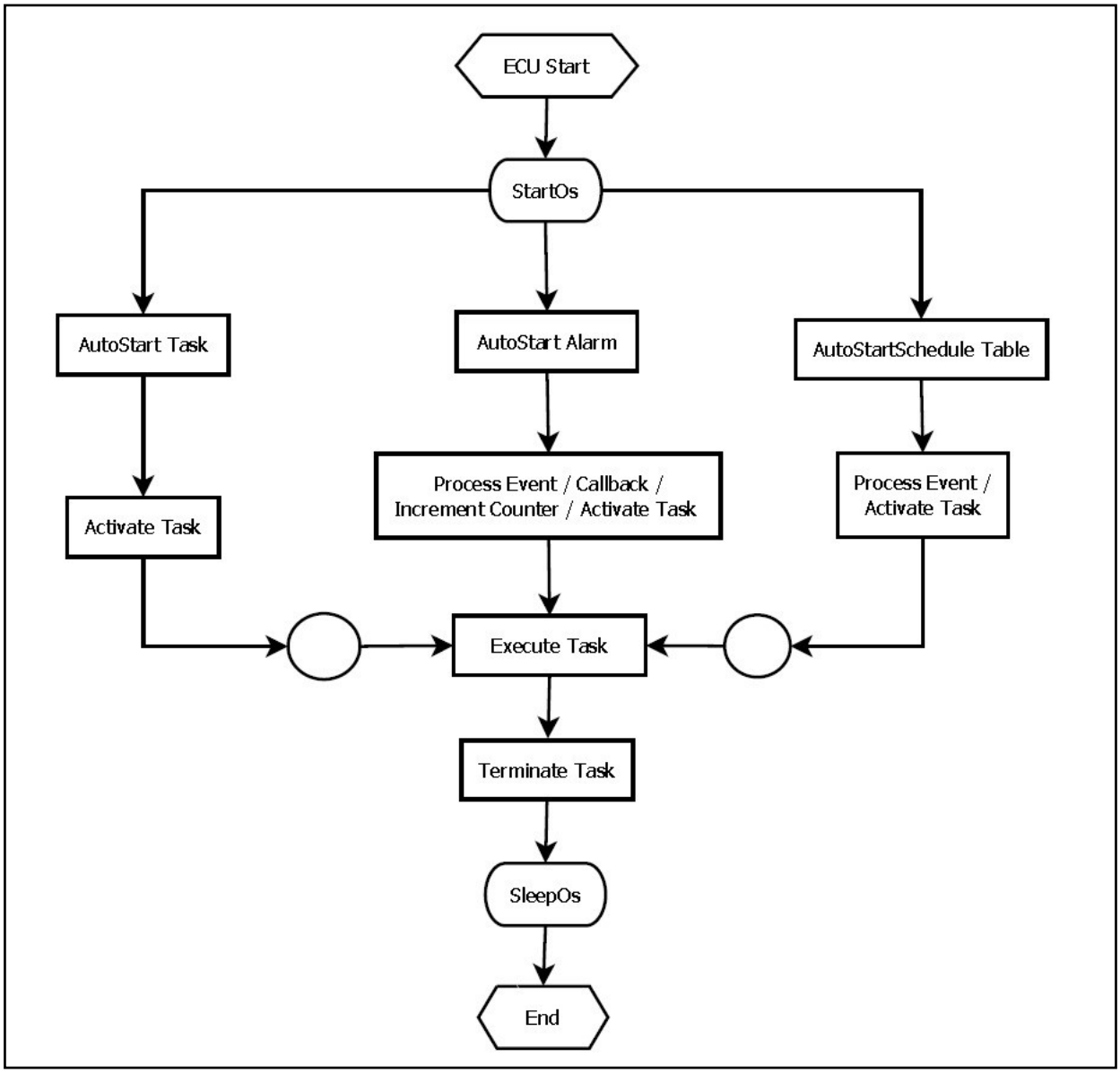


Figure 2: Startup Sequence

3.3 Interaction Between OS Module With Other Modules

The overview of the AUTOSAR OS software architecture is given in Figure 4. The OS Module interacts with the Run time environment (RTE), and BSW Scheduler.

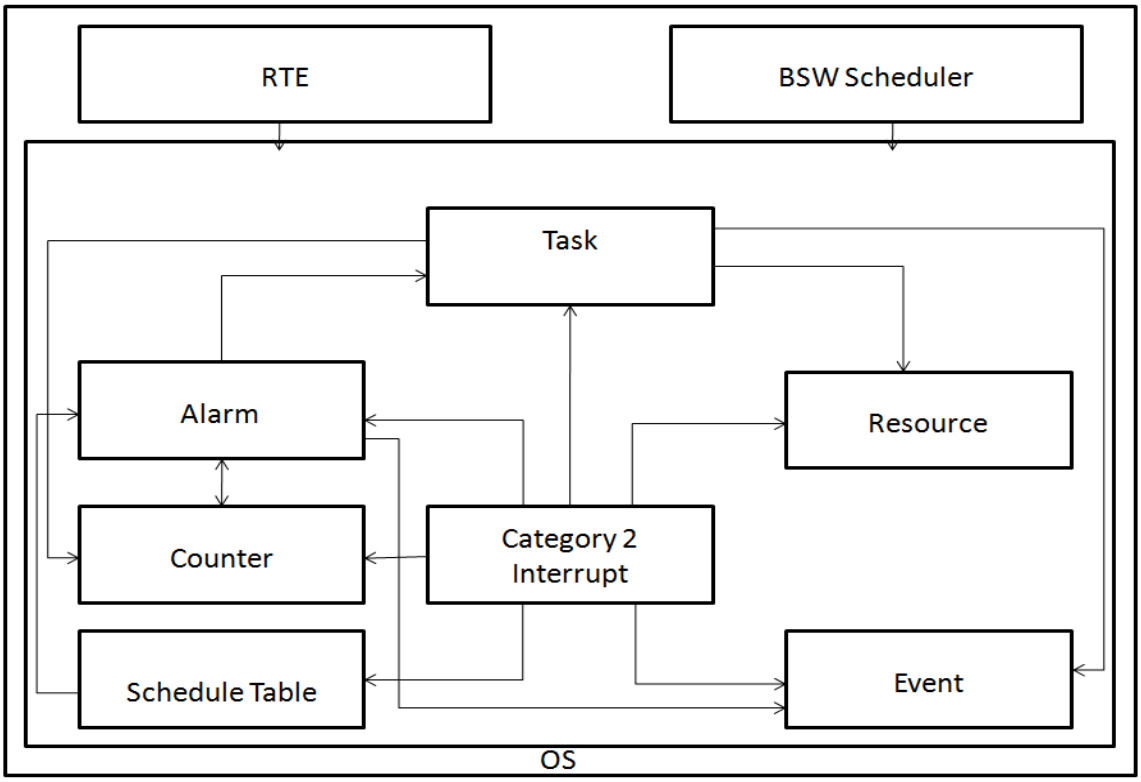


Figure 3: OS Module High Level Design

3.3.1 Interaction between OS Module and RTE

The configuration of an AUTOSAR system maps the runnables of a software Module to (one or more) tasks that are scheduled by the operating system. 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 Oslsrs that are scheduled by the operating system.

3.3.2 Interaction between OS Module and BSW Scheduler

Only the BSW scheduler uses the OS objects or OS services which include:

- Mapping of the scheduling objects to the OS tasks
- Specification of scheduling objects within tasks
- Specification of task sequences
- Specification of scheduling strategy

3.4 Conformance classes

Various requirements of the application software for the system and various capabilities of a specific system (e.g. processor, memory) demand different features of the operating system. In the following description, these operating system features are described as "conformance classes" (CC).
Conformance classes exist to support the following objectives:

- To provide convenient groups of operating system features for easier understanding and discussion of the OSEK operating system
- To allow partial implementations along pre-defined lines. These partial implementations may be certified as OSEK compliant
- To create an upgrade path from classes of lesser functionality to classes of higher functionality with no changes to the application using OSEK related features

Conformance classes are determined by the following attributes:

- Multiple requesting of task activation
- Task types
- Number of tasks per priority

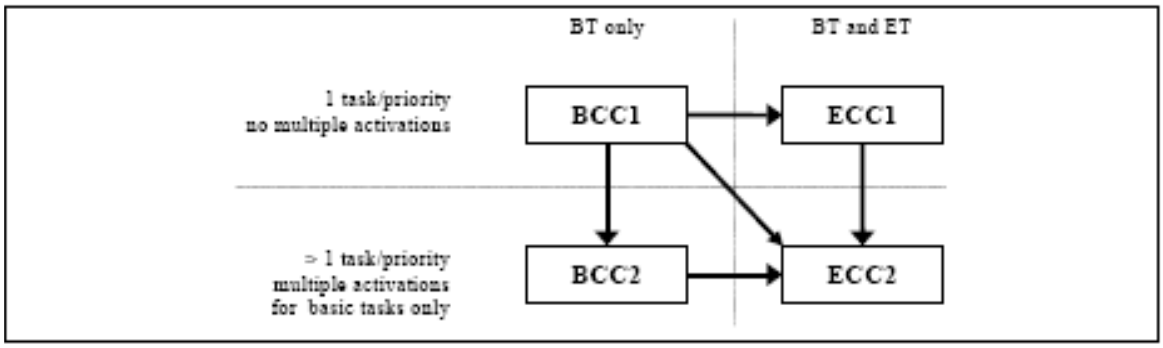


Figure 4: Restricted upward compatibility for conformance classes

The following conformance classes are defined:

- BCC1 (only basic tasks, limited to one activation request per task and one task per priority, while all tasks have different priorities)
- BCC2 (like BCC1, plus more than one task per priority possible and multiple requesting of task activation allowed)
- ECC1 (like BCC1, plus extended tasks)
- ECC2 (like ECC1, plus more than one task per priority possible and multiple requesting of task activation allowed for basic tasks)

3.5 Scalability Classes

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

- **Scalability Class 1 (SC1):** Deterministic RTOS baseline (tasks, events, counters, alarms, resources, schedule table)
- **Scalability Class 2 (SC2):** Timing based task determinism (low-latency, precise timing for periodic tasks, global time synchronization) & Synchronization of Schedule table (used for FlexRay)
- **Scalability Class 3 (SC3):** Protected memory for tasks avoids memory collisions for safety systems
- **Scalability Class 4 (SC4):** Timing and memory protected tasks, utilizes the full capabilities of the silicon for secure and protected RTOS designed specifically for the automotive

Feature	SC1	SC2	SC3	SC4	Hardware Requirements
OSEK OS (All Conformance Classes)	●	●	●	●	
Counter Interface	●	●	●	●	
Schedule Tables	●	●	●	●	
Stack Monitoring	●	●	●	●	
Protection Hook		●	●	●	
Timing Protection		●		●	Timers with high priority interrupt
Global Time / Synchronization support		●		●	Global Time Source
Memory Protection			●	●	MPU
OS-Applications	●*	●*	●	●	
Service Protection			●	●	
CallTrustedFunction			●	●	(non-) privilege modes

Table 1: Scalability classes

***Note:** The OS module shall support OS-Applications in case of Multi-Core for SC1 and SC2.

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

Table 2: Minimum requirements of scalability classes

3.6 Application Program Interface

The AUTOSAR Operating System establishes the Application Program Interface (API) which must be used for all users' actions connected with system calls and system objects. This API defines data types used by the system, the syntax of all run-time service calls, declarations and definitions of the system.
The current OSEK OS defines the valid calling context for service calls. Following table describes the allowance for OS Service Calls for respective OS.

Service	Task	Cat1 ISR	Cat2 ISR	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
ActivateTask	✓		✓							
TerminateTask	✓		C							
ChainTask	✓		C							
Schedule	✓		C							
GetTaskID	✓		✓	✓	✓	✓				✓
GetTaskState	✓		✓	✓	✓	✓				
DisableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EnableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SuspendAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ResumeAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SuspendOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ResumeOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GetResource	✓		✓							
ReleaseResource	✓		✓							
SetEvent	✓		✓							
ClearEvent	✓			C						
GetEvent	✓		✓	✓	✓	✓				
WaitEvent	✓			C						
GetAlarmBase	✓		✓	✓	✓	✓				
GetAlarm	✓		✓	✓	✓	✓				
SetRelAlarm	✓		✓							
SetAbsAlarm	✓		✓							
CancelAlarm	✓		✓							
GetActiveApplicationMode	✓		✓	✓	✓	✓	✓	✓		
StartOS										
ShutdownOS	✓		✓	✓			✓			
GetApplicationID	✓		✓	✓	✓	✓	✓	✓		✓
GetISRID	✓		✓	✓						✓
CallTrustedFunction	✓		✓							
CheckISRMemoryAccess	✓		✓	✓						✓
CheckTaskMemoryAccess	✓		✓	✓						✓
CheckObjectAccess	✓		✓	✓						✓
CheckObjectOwnership	✓		✓	✓						✓
StartScheduleTableRel	✓		✓							
StartScheduleTableAbs	✓		✓							
StopScheduleTable	✓		✓							
NextScheduleTable	✓		✓							
StartScheduleTableSynchron	✓		✓							
SyncScheduleTable	✓		✓							
GetScheduleTableStatus	✓		✓							
SetScheduleTableAsync	✓		✓							
IncrementCounter	✓		✓							
GetCounterValue	✓		✓							
GetElapsedValue	✓		✓							
TerminateApplication	✓		✓	✓[1]						
AllowAccess	✓		✓							
GetApplicationState	✓		✓	✓	✓	✓	✓	✓		✓

Table 3: Allowed Calling Context for OS Service Calls

3.7 Imported Types

This section explains the Data types imported by the Os Module and lists its dependency on other modules.

3.7.1 Standard Types

The following list shows all types of Std_Types.h that are used by the Os Module

- Std_ReturnType
- Std_VersionInfoType
- StatusType

3.8 Operating System Execution Control

3.8.1 StartOS

StartOS		
Prototype:	void StartOS (AppModeType <Mode>)	
Service ID:	OSServiceId_StartOS	
Sync/Async:	NA	
Reentrancy:	Non Re-entrant	
Parameters (In):	**Type**	**Parameter**
	AppModeType	Mode
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	**Type**	**Possible Return Values**
	void	None
Description:	The user can call this system service to start the operating system in a specific mode.	
Configuration Dependency:	None	
Pre-conditions:	None	

3.8.2 ShutdownOS

ShutdownOS		
Prototype:	void ShutdownOS (StatusType <Error>)	
Service ID:	OSServiceId_ShutdownOS	
Sync/Async:	NA	
Reentrancy:	Non-Reentrant	
Parameters (In):	**Type**	**Parameter**
	StatusType	Error
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	**Type**	**Possible Return Values**
	None	None
Description:	The user can call this system service to abort the overall system (e.g. emergency off).	
Configuration Dependency:	None	
Pre-conditions:	None	

4. Product Release Notes

4.1 Overview

This chapter describes release information.

4.2 Scope of the release

The scope of this document is restricted to the below Os module.

Module	Module version
Os	2.4.5

※ Module version means sw version of BswModule Description(Bswmd)

4.3 Module release notes

4.3.1 Change Log

4.3.1.1 Version 2.4.5.0_HF1

➢ **Bug**

- Context Save/Recovery Failed in Os_ISRHandler

Cause	Context Save/Recovery Failed
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Add critical sections in Os_InterCoreInterruptHandler

Cause	Critical section is needed
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

➢ **Improvement**

- Invalid ISR handling code added

Cause	Invalid ISR handling code is needed
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.2 Version 2.4.5.0

➢ **Improvement**

- ORTI file generate for Stack attribute wrong.

Cause	* To improve ORTI Stack attribute
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.3 Version 2.4.4.2

➢ **Task**

- An UM document template is changed

Cause	* To apply changed document template.
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.4 Version 2.4.4.1

➢ **Task**

- Modify pdf file to support 4GB in Osos/Mpu/Region Size

Cause	* To prevent ECC errors due to the speculative access function of the ARM M7 core, a 4GB Region Mpu must be supported.
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.5 Version 2.4.4.0

➢ **Bug**

- Fix user stack bottom address value calculation error

Cause	* Error due to invalid pointer operation when calculating user stack bottom
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Fix an error when checking the stack size value in SC3/4

Cause	* Invalid index access when reading stack size from user configuration file
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.6 Version 2.4.3.0

➢ **Improvement**

- Improvement of coding for UNECE Cyber Security

Cause	* Client request
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.7Version 2.4.2.0

➢ **Improvement**

- Improvement of coding for UNECE Cyber Security

Cause	* Client request
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Improvement of sorting the input arxml comment of the generaged file

Cause	* Difficult to track history due to changes in generated file comments for every generation
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Adding validation for invalid stack size when using SC3/4

Cause	* Adding stack size validation according to MCU restrictions
Operation Impact	N/A
Configuration Impact	* stack size should be changed according to 5.2.1 and 5.10.1 when using SC3/4.
Required measure of ASW	N/A

4.3.1.8Version 2.4.1.0

➢ **Improvement**

- Fix a bug for FPU related function

Cause	* invalid stack pointer could be calculated by wroing context size due to FPU
Operation Impact	* occasionally results in an invlid float point calculation when using FPU * occasionally unexpected stack fault
Configuration Impact	* bulid option should be changed according to OsFpuSupport (refer to 5.1.1)
Required measure of ASW	N/A

4.3.1.9Version 2.4.0.0

➢ **Feature**

- Support of SC3 and SC4

Cause	* Support of SC3 and SC4
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Support of FPU

Cause	* Support of FPU
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

➢ **Improvement**

- Fix for handling of OsCounter in case of MultiOs

Cause	* OsCounter should be handled with each CoreId in case of MultiOs
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.10 Version 2.3.0.0

➢ **Feature**

- Support of SC2

Cause	* Support of SC2
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.11 Version 2.2.0.0

➢ **Feature**

- Support of Traveo II 4BF Series

Cause	* Support of Traveo II 4BF Series
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.12 Version 2.1.0.0

➢ **Feature**

- Support of Multicore

Cause	* Support of Multicore
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.13 Version 2.0.1.1

➢ **Improvement**

- Fix a bug for Entry Nesting Interrupt

Cause	* Stack was violated by entry nesting interrupt. And, It was solved.
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Polyspace (Runtime Error) measurement

Cause	To handle Runtime Error of Polyspace
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.14 Version 2.0.0.0

➢ **Feature**

- Support 4M for Traveo II

Cause	* Support 4M for Traveo II (M7 Core)
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.15 Version 1.1.0.0

➢ **Feature**

- Added NMI Handler

Cause	* Add NMI handling function for user to handle additional processing when NMI is used
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	If NMI exception is used, Os_CallBackNMInterrupt callback function should be implemented

➢ **Improvement**

- Expand count of ScheduleTable ExpiryPoint(2^8 -> 2^32)

Cause	* Expiry Point is required more than 255
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Fix Stack struct generation

Cause	* Invalid stack struct generated when no Extended Task and no OsApplication config
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Change UNUSED macro to OS_UNUSED macro

Cause	* To use OS module macro
Operation Impact	N/A
Configuration Impact	N/A

Required measure of ASW	N/A

- Move Debug Feature to Os_Imp

Cause	* To move the Os Debug Features on Os_Imp
Operation Impact	N/A
Configuration Impact	OsOS/OsDebug should be removed
Required measure of ASW	N/A

- Clear running TP time in case of stopping of a running TP

Cause	* To modify missing incorrect interrupt lock budget value when using Enter/Exit
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Change calculation of stack size

Cause	* To align stack frame size
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Change of OS_ALARM_COUNT

Cause	* Invalid Alarm ID is determined valid because OS_ALARM_COUNT is number of Alarms and ScheduleTables.
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Change location of flag indicating that OS has started

Cause	* Change of OS start flag setting location to support some MCU OS (like multicore chorus).
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

- Fix the Os_GetCurPrivMode function

Cause	* It was fixed that superviosr mode and user mode can be returned correctly.
Operation Impact	N/A
Configuration Impact	N/A
Required measure of ASW	N/A

4.3.1.16 Version 1.0.0.0

➢ **Initial Release**
- N/A

4.3.2 Limitation

- Supported Scalability Class : SC1, SC2, SC3 and SC4
  Only single-core supports SC3/4
- If SC2 is used, TCPWM0 Group#2, Counter#1 and #2 can't be used for anything other than timing protection. And TCPWM0 Group#2, Counter#3 and #4 for CYT4BB and TCPWM1 Group#2, Counter#1 and #2 can't be used for CYT4BF.
- If a Timing Protection Error occurs, it is required to clear the error by taking reset.
- Available Hardware counter number : 1 (SYSTICK is used for counter driving interrupt)
- Supported target MCU : Traveo-II series (CYT2B7, CYT2B9, CYT2BL, CYT4BB, CYT4BF)
- If SC3/4 is used, Category 2 ISRs belonging to non-trusted OS-Applications also operate as trusted (Memory Protection is not applicable in Category 2 ISRs). CheckTaskMemoryAccess() and CheckIsrMemoryAccess() can not be used for MPU regions where OsMpuRegionEnabledOnStart is true.
- If Memory Protection is required, the MPU regions should be configured in Osos.
  The MPU regions must be configured by user according to the system.
- In orders to use the Memory Protection, an understanding of the linker description is required.
- SWP only provides sample MPU region configurations.
  However, in the case of ARM M7 core(CYT3XX/4XX), if you change the configurations of MPU regions from 0 to 7 in Osos, you must also change the registers(EcuM_Rbar and EcuM_Rasr) related to the MPU region in the integration_EcuM/EcuM_Callout.c
  → refer confluence page https://swpfaq.hyundai-autoever.com/x/U4pKAg
- The base address of MPU region must be set considering align.
  The base address of sections such as code, data, and stack must be aligned to 2^n set above.
  The size of MPU region must be set in the form of 2^n.
  All stacks must be same size. As a result, memory usage could be increased.
- After every build, user should check memory usage and allocation.
  Since the base address and size of each section can be changed at every build,
  So, it is necessary to check if they are properly arranged through a map file after building.
- If external flash is existed in the system, consider adding region for external flash.
- It is necessary to design a section that can be accessed from the OS application.
- Up to 16 ARM M7 MPUs can be used, up to 8 ARM M4 MPUs can be used, and since last number of MPU is allocated as a stack area, so it cannot be used.
  * A total of 13 MPU regions, including 12 MPU regions and stack regions, are the minimum number of MPU regions required for memory protection in CYT3XX/4XX, so arbitrary deletion is not possible.
  * In CYT2XX, user can additionally use only 4 MPU regions.
  → Since the number of MPUs is insufficient, a system design considering the number of MPUs is required.
- If you want to use FPU, OsFpuSupport needs to be 'true', and you must compile with the '-fsingle' option. And floating point operations should not be used in OS Hooks.
- By AUTOSAR specification, synchronization between multi-cores is provided within StartOS function, and after that, if needed, the application must implement the synchronization mechanism itself for tasks activation and events setting, etc. (Refer OS579 of AUTSOAR_SWS_OS)

5. Configuration Guide

5.1 General

5.1.1 OsOS

This container contains the configuration parameters to configure general properties of Os.
The description of the parameters present in the container is as below

Parameter Name	Value	
OsNumberOfCores	From SRS	Fixed
OsScalabilityClass	From SRS	Fixed
OsStackMonitoring	True	Fixed
OsStatus	EXTENDED	Fixed
OsUseGetServiceId	True	Fixed
OsUseParameterAccess	True	Fixed
OsUseResScheduler	True	Fixed
OsProcessor	From SRS	Fixed
OsProcessorSeries	From SRS	Fixed
OsIncludeHeaderFile	-	Fixed
OsUserStackSize	User Defined	Changeable
OsStackFrame	StackFrame8	Fixed
OsWdgSecondsPerTick	SC1,3: - / SC2, 4: From SRS	Fixed
OsSystemTimerClock	From SRS	Changeable
OsFpuSupport	User Defined	Changeable

1) OsNumberOfCores
   - Maximum number of cores that are controlled by the OS. The OS uses the value internally. It depends on the ECU HW.
2) OsScalabilityClass
   - 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.
   - SC1: Basic functionalities of OS
   - SC2: Functionality related to timing protection and synchronization
   - SC3: Functionality related to memory protection
   - SC4: Combination of SC1, SC2 and SC3 scalability classes.
3) OsStackMonitoring
   - This parameter selects the stack monitoring of Tasks/Category 2 ISRs.
   - true: Stacks are monitored
   - false: Stacks are not monitored
4) OsStatus
   - This parameter specifies whether a system with standard or extended status has to be used. Automatic assignment is not supported for this attribute.
   - EXTENDED: The Standard status does the normal checks on calling operating system services.
   - STANDARD: The extended status allows for enhanced plausibility checks on calling operating system services.
5) OsUseGetServiceId

- This parameter enables or disables the GetServiceId function.
  - True: GetServiceId function exists.
  - False: GetServiceId function does not exists.

6) OsUseParameterAccess
   - This parameter enables or disables the parameter accessing.
     - True: Parameter accessing exists.
     - False: Parameter accessing does not exists
7) OsUseResScheduler
   - This parameter specifies whether the Resource Scheduler is used within the application.
     - True: The task that takes this resource cannot be pre empted.
     - False: The task that takes this resource can be pre empted.
8) OsProcessor
   - This parameter specifies the Os processor name.
9) OsProcessorSeries
   - This parameter gives Processor Series name.
10) OsIncludeHeaderFile
    - This parameter gives name of the Header file to include in Os source file.
11) OsUserStackSize
    - This parameter gives User Stack size. The OS uses the value internally. It depends on the ECU HW.
12) OsStackFrame
    - This parameter gives frame size of the Stack in bytes. The OS uses the value internally. It depends on the ECU HW.
      Note: StackFrame8 (SC1, SC2)
13) OsWdgSecondsPerTick
    - This parameter specifies Time of one hardware tick in seconds for Timing Protection.
14) OsFpuSupport
    - The OsFpuSupport defines whether to support saving and restoring FPU registers during context switching.
    - If OsFpuSupport is 'true', 136 bytes of memory per Task or CAT2 ISR will be used additionally. Compile option '-fsingle' should be added. And if any library is used, floating-point suffix '_sd' shobule be added. For example, if libmath is used, you should add 'math_sd' for it.
      - true: Save and restore FPU registers during context switching
      - false: Do not handle FPU registers

5.1.2 OsOs - OsHooks

This container contains the configuration parameters to configure Os Hooks.

Parameter Name	Value	
OsErrorHook	True	Fixed
OsPostTaskHook	False	Fixed
OsPreTaskHook	False	Fixed
OsProtectionHook	SC1: False / SC2, 3, 4: True	Fixed
OsShutdownHook	True	Fixed
OsStartupHook	True	Fixed
OsProfilingHook	True	Fixed

1) OsErrorHook
   This parameter enables or disables the error hook.
   - true: Hook is called
   - false: Hook is not called
2) OsPostTaskHook
   This parameter enables or disables the Post task hook.
   - true: Hook is called
   - false: Hook is not called
3) OsPreTaskHook
   This parameter enables or disables the Pre task hook.
   - true: Hook is called
   - false: Hook is not called
4) OsProtectionHook
   This parameter enables or disables the Protection task hook.
   - true: Protection hook is called on protection error
   - false: Protection hook is not called
5) OsShutdownHook
   This parameter enables or disables the Shutdown hook.
   - true: Hook is called
   - false: Hook is not called
6) OsStartupHook
   This parameter enables or disables the Startup hook.
   - true: Hook is called
   - false: Hook is not called
7) OsProfilingHook
   Provide following hook functions for profiling Task and Category 2 ISR state information.
   Os_IsrEntryHook()
   Os_IsrExitHook ()
   Os_IsrKillHook ()
   Os_TaskActivationHook()
   Os_TaskTerminationHook()
   Os_TaskPreemptionHook()
   Os_TaskRunningHook()
   Os_TaskWaitingHook()
   Os_TaskReleaseHook()
   Os_TaskKillHook()
   Os_IdleEntryHook()
   Os_IdleExitHook()
   - true: Hook is called
   - false: Hook is not called

5.1.3 OsOs – OsMpuRegion

This container contains the configuration parameters to configure MPU on CYTxxx. This container is only used on the CYTxxx.

Parameter Name	Value	
OsMpuRegionNumber	User Defined	Changeable
OsMpuRegionEnabledOnStart	User Defined	Changeable
OsMpuRegionBaseAddress	User Defined	Changeable
OsMpuRegionSize	User Defined	Changeable
OsMpuRegionRwPermission	User Defined	Changeable

Parameter Name	Value	
OsMpuRegionSubregionDisable	User Defined	Changeable
OsMpuRegionExecutionPermission	User Defined	Changeable
OsMpuRegionAttribute	User Defined	Changeable
OsMpuRegionLinkerSection	User Defined	Changeable
OsMpuRegionAccessingApplication	User Defined	Changeable

1) OsMpuRegionNumber
   - This parameter defines the MPU region number to be accessed.
     The available range is 0 to 6 in case of Cortex-M4F and 0 to 14 in case of Cortex-M7F. Number 7 in case of Cortex-M4F and Number 15 in case of Cortex-M7F is reserved for the stack area used by the OS.
   - The numbers from 0 to 7 in case of Cortex-M7F are reserved to prevent speculative access function from occurring ECC Ram Error.
2) OsMpuRegionEnabledOnStart
   - This parameter determines that enable this MPU region before the OS is started.
   - True: The MPU region is enabled before the OS is started.
   - False: The MPU region is enabled when the OS is started and the OS-Application object specified in OsMpuRegionAccessingApplication is executed.
3) OsMpuRegionBaseAddress
   - This parameter defines the start of the memory region. You must align this to a region-sized boundary. For example, if a region size of 8KB is programmed for a given region, the base address must be a multiple of 8KB.
4) OsMpuRegionSize
   - This parameter defines the size of the region specified by the Memory Region Number Register.
     The range that can be set is from 32bytes to 2GB, and it should be set to a power of 2.
5) OsMpuRegionRwPermission
   - This parameter defines the data access permissions.
     Each region can be given no access, read-only access, or read/write access permissions for Privileged or all modes.
6) OsMpuRegionSubregionDisable
   - This parameter defines the subregion disable.
     Each bit position represents a sub-region, 0-7. The meaning of each bit is:
     0 = address range is part of this region
     1 = address range is not part of this region.

     Each region can be split into eight equal sized non-overlapping subregions. An access to a memory address in a disabled subregion does not use the attributes and permissions defined for that region. Instead, it uses the attributes and permissions of a lower priority region or generates a background fault if no other regions overlap at that address. This enables increased protection and memory attribute granularity.
     All region sizes between 256 bytes and 2GB support eight subregions. Region sizes below 256 bytes do not support subregions.
7) OsMpuRegionExecutionPermission
   - This parameter determines if a region of memory is executable.
   - True: The MPU region is executable.
   - False: The MPU region is not executable.
8) OsMpuRegionAttribute
   - This parameter defines attributes of MPU region. Each region has a number of attributes associated with it. These control how a memory access is performed when the processor accesses an address that falls within a given region. The attributes are:
     • Memory type, one of:
       - Strongly Ordered
       - Device

- Normal.
- Shared or Non-shared
- Non-cacheable
- Write-through cacheable
- Write-back cacheable
- Read allocation
- Write allocation

9) OsMpuRegionLinkerSection
   - This parameter defines linker sections that should be located in MPU region.
   ex)
   Code section: ".text.ECUC_PARTITION_NAME*"
   Read only data section: ".rodata.ECUC_PARTITION_NAME*"
   Variable section: ".data.ECUC_PARTITION_NAME*" ".bss.ECUC_PARTITION_NAME*"
10) OsMpuRegionAccessingApplication

This parameter defines the reference to OS-Application which has access permission to this region.

5.2 Task Management

5.2.1 OsTask

This container contains the configuration parameter(s) for Os Tasks.

Parameter Name	Value	
OsTaskActivation	1	Fixed
OsTaskPriority	User Defined	Changeable
OsTaskSchedule	FULL	Fixed
OsTaskStackSize	User Defined	Changeable
OsTaskAccessingApplication	User Defined	Changeable
OsTaskEventRef	User Defined	Changeable
OsTaskResourceRef	User Defined	Changeable

1) OsTaskActivation
   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.
2) OsTaskPriority
   The priority of a task is defined by the value of this attribute. This value has to be understood a s a relative value, i.e. the values show only the relative ordering of the tasks. OSEK OS defines t he lowest priority as zero (0); larger values correspond to higher priorities.
3) OsTaskSchedule
   This parameter defines the preemptability of the task. If this attribute is set to NON, no internal resources may be assigned to this Task.
   - FULL: Task is preemptable
   - NON: Task is not preemptable
4) OsTaskStackSize
   This parameter specifies the stack size in extended task. For basic task, this configuration is not used.
   And if SC3/4 is used, the stack size of all os-applications and tasks should be the same. And the ld file should be modified so that the base address of the stack is aligned with the stack size.

5) OsTaskAccessingApplication
This parameter references the Application which has the access to this Task.
6) OsTaskEventRef
This parameter defines the list of Events the Extended Task may react on.
7) OsTaskResourceRef
This parameter defines the list of Resources accessed by this Task.

5.2.2 OsTask - OsTaskAutostart

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.

Parameter Name	Value	
OsTaskAppModeRef	OsAppMode0	Fixed

1) OsTaskAppModeRef
This parameter references the application modes in which that Task is activated on startup of the OS.

5.2.3 OsTask - OsTaskTimingProtection

This container contains all parameters regarding timing protection of the Task.
To use Timing Protection, Os Scalability Class (OsScalabilityClass) should be configured as SC2 or SC4.

Parameter Name	Value	
OsTaskAllInterruptLockBudget	-	Fixed
OsTaskExecutionBudget	User Defined	Changeable
OsTaskOsInterruptLockBudget	-	Fixed
OsTaskTimeFrame	-	Fixed

1) OsTaskAllInterruptLockBudget
This parameter specifies the maximum time for which the task is allowed to lock all interrupts (via SuspendAllInterrupts () or DisableAllInterrupts ()) (in seconds).
2) OsTaskExecutionBudget
This parameter specifies the maximum allowed execution time of the task (in seconds).
3) OsTaskOsInterruptLockBudget
This parameter specifies the maximum time for which the task is allowed to lock all Category 2 interrupts (via SuspendOSInterrupts()) (in seconds).
4) OsTaskTimeFrame
This parameter specifies the minimum inter-arrival time between activations and/or releases of a task (in seconds).

5.2.4 OsTask - OsTaskTimingProtection - OsTaskResourceLock

This container contains the worst case time between getting and releasing a given resource (in seconds).

Parameter Name	Value	
OsTaskResourceLockBudget	-	Fixed

Parameter Name	Value	
OsTaskResourceLockResourceRef	-	Fixed

1) OsTaskResourceLockBudget
   This parameter specifies the maximum time the task is allowed to lock the resource (in seconds).
2) OsTaskResourceLockResourceRef
   This parameter references the resource used by the Task.

5.3 Interrupt Processing

5.3.1 OsIsr

This container contains the configuration parameter(s) for Os Isrs.

Parameter Name	Value	
OsIsrCategory	CATEGORY_2	Fixed
OsIsrName	User Defined	Changeable
OsIsrHardwarePriority	User Defined	Changeable
OsIsrResourceRef	User Defined	Changeable

1) OsIsrCategory
   This parameter specifies the category of this ISR.
   - CATEGORY_1: Interrupt is of category 1
   - CATEGORY_2: Interrupt is of category 2
2) OsIsrName
   Interrupt Names for the Interrupt.
3) OsIsrHardwarePriority
   This parameter designates the hardware priority of the ISR.
4) OsIsrResourceRef
   This parameter defines the resources accessed by this ISR.

5.3.2 OsIsr - OsIsrTimingProtection

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.
To use Timing Protection, Os Scalability Class (OsScalabilityClass) should be configured as SC2 or SC4.

Parameter Name	Value	
OsIsrAllInterruptLockBudget	-	Fixed
OsIsrExecutionBudget	User Defined	Changeable
OsIsrOsInterruptLockBudget	-	Fixed
OsIsrTimeFrame	-	Fixed

1) OsIsrAllInterruptLockBudget
   This parameter specifies the maximum time for which the ISR is allowed to lock all interrupts (via SuspendAllInterrupts() or DisableAllInterrupts()) (in seconds).
2) OsIsrExecutionBudget

This parameter specifies the maximum allowed execution time of the interrupt (in seconds).

3) OsIsrOsInterruptLockBudget
   This parameter specifies the maximum time for which the ISR is allowed to lock all Category 2 in terrupts (via SuspendOSInterrupts()) (in seconds).
4) OsIsrTimeFrame
   This parameter specifies the minimum inter-arrival time between successive interrupts (in second s).

5.3.3 OsIsr - OsIsrTimingProtection - OsIsrResourceLock

This container contains a list of times the interrupt uses resources.

Parameter Name	Value	
OsIsrResourceLockBudget	-	Fixed
OsIsrResourceLockResourceRef	-	Fixed

1) OsIsrResourceLockBudget
   This parameter specifies the maximum time the interrupt is allowed to hold the given resource (in seconds).
2) OsIsrResourceLockResourceRef
   This parameter defines the resource the locking time is depending on.

5.4 Resource Management

5.4.1 OsResource

This container 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.

Parameter Name	Value	
OsResourceProperty	User Defined	Changeable
OsResourceAccessingApplication	User Defined	Changeable
OsResourceLinkedResourceRef	-	Not Supported

1) OsResourceProperty
   This parameter specifies the type of the resource.
   - 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 an standard resource
2) OsResourceAccessingApplication
   This parameter references to the applications which have an access to this Resource.
3) OsResourceLinkedResourceRef
   This parameter references to the linked resource. Configuration of this parameter is valid only if the

Resource property is configured as LINKED.

5.5 Counter

5.5.1 OsCounter

This container contains the configuration parameters for Os Counters.

Parameter Name	Value	
OsCounterMaxAllowedValue	0xFFFFFFFF	Fixed
OsCounterMinCycle	1	Fixed
OsCounterTicksPerBase	1	Fixed
OsCounterType	HARDWARE	Fixed
OsDrivingInterrupt	First source from available timer	Fixed
OsSecondsPerTick	User Defined	Changeable
OsCounterHardwarePriority	-	Fixed
OsCounterAccessingApplication	User Defined	Changeable

1) OsCounterMaxAllowedValue
   This parameter specifies the maximum possible allowed value of the system counter in ticks.
2) OsCounterMinCycle
   This parameter specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.
3) OsCounterTicksPerBase
   This parameter specifies the number of ticks required to reach a counter specific unit.
4) OsCounterType
   This parameter specifies the natural type or unit of the Counter.
   - HARDWARE: This counter is driven by some hardware e.g. a hardware timer unit.
   - SOFTWARE: This counter is driven by some software which calls the Increment counter service.
5) OsDrivingInterrupt
   Port Specific interrupt which drives the HARDWARE counter. On occurance of this interrupt the counter is incremented by 1.
6) OsSecondsPerTick
   This parameter specifies the Time of one hardware tick in seconds.
   MAX value is depends on SYSTICK clock (CPU clock). SYSTICK clock should be set to OsSystemTimerClock.
   Ex) Max OsSecondsPerTick is 0.067s when OsSystemTimerClock is 160MHz
7) OsCounterHardwarePriority
   The hardware priority of the Counter ISR.
   If this parameter is not set, assign priority automatically.(CAT2 ISR max priority + 1)
8) OsCounterAccessingApplication
   This parameter specifies the reference to applications which have an access to this object.

5.5.2 OsCounter – OsDriver

This container contains the information that 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 has 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.

Parameter Name	Value	
OsGptChannelRef	-	Not Supported

1) OsGptChannelRef
   This parameter specifies the reference to Gpt channel.

5.5.3 OsCounter – OsTimeConstant

This container allows the user to define constants which can be e.g. used to compare time values with timer tick values.

Parameter Name	Value	
OsConstName	User Defined	Changeable
OsTimeValue	User Defined	Changeable

1) OsConstName
   This parameter specifies the name which is accessed by the application to get the OsTimeValue.
2) OsTimeValue
   This parameter specifies the value of the constant in seconds.

5.6 Alarm

5.6.1 OsAlarm

This container contains the configuration (parameters) of the Os Alarms. 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.

Parameter Name	Value	
OsAlarmAccessingApplication	User Defined	Changeable
OsAlarmCounterRef	User Defined	Changeable

1) OsAlarmAccessingApplication
   This parameter specifies the reference to applications which have an access to Alarm object.
2) OsAlarmCounterRef
   This parameter specifies the reference to the assigned counter for that alarm.

5.6.2 OsAlarm - OsAlarmAction

This container defines which type of notification is used when the alarm expires.

- OsAlarmActivateTask
- OsAlarmCallback
- OsAlarmIncrementCounter
- OsAlarmSetEvent

5.6.3 OsAlarm – OsAlarmAction - OsAlarmActivateTask

This container specifies the parameters to activate a task.

Parameter Name	Value	
OsAlarmActivateTaskRef	User Defined	Changeable

1) OsAlarmActivateTaskRef
   This container specifies the parameters to activate a task.

5.6.4 OsAlarm – OsAlarmAction - OsAlarmCallback

This container specifies the parameters to call a callback Os alarm action.

Parameter Name	Value	
OsAlarmCallbackName	User Defined	Changeable

1) OsAlarmCallbackName
   This parameter specifies the Name of the function that is called when this alarm callback is triggered.

5.6.5 OsAlarm – OsAlarmAction - OsAlarmIncrementCounter

This container specifies the parameters to increment a counter.

Parameter Name	Value	
OsAlarmIncrementCounterRef	User Defined	Changeable

1) OsAlarmIncrementCounterRef
   This parameter specifies the reference to the counter that will be incremented by the alarm action.

5.6.6 OsAlarm – OsAlarmAction - OsAlarmSetEvent

This container specifies the parameters to set an event.

Parameter Name	Value	
OsAlarmSetEventRef	User Defined	Changeable
OsAlarmSetEventTaskRef	User Defined	Changeable

1) OsAlarmSetEventRef
   This parameter specifies the reference to the event that will be set by the alarm action.
2) OsAlarmSetEventTaskRef
   This parameter specifies the reference to the task that will be activated by the event.

5.6.7 OsAlarm – OsAlarmAutostart

This container defines if an alarm is started automatically at system start-up depending on the application mode.

Parameter Name	Value	
OsAlarmAlarmTime	User Defined	Changeable
OsAlarmAutostartType	User Defined	Changeable
OsAlarmCycleTime	User Defined	Changeable
OsAlarmAppModeRef	User Defined	Changeable

1) OsAlarmAlarmTime
   This parameter specifies 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.
2) OsAlarmAutostartType
   This parameter specifies the type of auto start for the alarm.
   - ABSOLUTE: The alarm is started on startup via SetAbsAlarm().
   - RELATIVE: The alarm is started on startup via SetAbsAlarm.
3) OsAlarmCycleTime
   This parameter specifies the Cycle time of a cyclic alarm in ticks. If the value is 0 than the alarm is not cyclic.
4) OsAlarmAppModeRef
   This parameter specifies the reference to the application modes for which the AUTOSTART shall be performed.

5.7 Event Mechanism

5.7.1 OsEvent

This container contains the configuration parameter(s) for Os Events.

Parameter Name	Value	
OsEventMask	User Defined	Changeable

1) OsEventMask
   The event is represented by its mask. The event mask is the number which range is from 1 to 0xFFFFFFFF, preferably with only one bit set. If this parameter is left empty, OS generator makes Event mask automatically but auto generation is limited to 32 Events.

5.8 Schedule Table

AutoEver SWP does not support Schedule Table due to AutoEver policy.

5.9 Application Modes

5.9.1 OSAppMode

This container defines 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.

5.10 OsApplication

5.10.1 OsApplication

This container defines the collection of objects. 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.

Parameter Name	Value	
OsApplicationCoreAssignment	User Defined	Changeable
OsTrusted	User Defined	Changeable
OsApplicationStackSize	User Defined	Changeable
OsAppAlarmRef	User Defined	Changeable
OsAppCounterRef	User Defined	Changeable
OsAppEcucPartitionRef	User Defined	Changeable
OsAppIsrRef	User Defined	Changeable
OsAppResourceRef	User Defined	Changeable
OsAppScheduleTableRef	User Defined	Changeable
OsAppTaskRef	User Defined	Changeable
OsRestartTask	User Defined	Changeable

1) OsApplicationCoreAssignment
   This parameter specifies an ID of the core onto which the OsApplication is bound.
2) OsTrusted
   This parameter specifies that an OS-Application is trusted or not.
   - True: OS-Application is trusted.
   - False: OS-Application is not trusted.
3) OsApplicationStackSize
   This parameter specifies the stack size in an OS-Application. Basic Task and Category2 ISR in this OS Application share this stack area.

And if SC3/4 is used, the stack size of all os-applications and tasks should be the same. And the ld file should be modified so that the base address of the stack is aligned with the stack size.

4) OsAppAlarmRef
   This parameter specifies the reference to the Os Alarms that belong to the OsApplication.
5) OsAppCounterRef
   This parameter specifies the reference to the Os Counters that belong to the OsApplication.
6) OsAppEcucPartitionRef
   Denotes which "EcucPartition" is implemented by this "OSApplication".
7) OsAppIsrRef
   This parameter specifies the reference to the Os Isrs belong to the OsApplication.
8) OsAppResourceRef
   This parameter specifies the reference to the Os Resources that belong to the OsApplication.
9) OsAppScheduleTableRef
   This parameter specifies the reference to the Os Schedule Tables that belong to the OsApplication.
10) OsAppTaskRef
    This parameter specifies the reference to the Os Tasks that belong to the OsApplication.
11) OsRestartTask
    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.

5.10.2 OsApplication - OsApplicationHooks

This container defines the Os Application specific hooks.

Parameter Name	Value	
OsAppErrorHook	False	Fixed
OsAppShutdownHook	False	Fixed
OsAppStartupHook	False	Fixed

1) OsAppErrorHook
   This parameter selects the OS-Application error hook.
   - True: Hook is called.
   - False: Hook is not called.
2) OsAppShutdownHook
   This parameter selects the OS-Application specific shutdown hook.
   - True: Hook is called.
   - False: Hook is not called.
3) OsAppStartupHook
   This parameter selects the OS-Application specific startup.
   - True: Hook is called.
   - False: Hook is not called.

5.10.3 OsApplication - OsApplicationTrustedFunction

AutoEver SWP does not support OsApplicationTrustedFunction due to AutoEver policy.

5.10.4 OsApplication - OsMemoryBlock

Container with details about memory area accessible to the Application

To use Memory Protection, Os Scalability Class (OsScalabilityClass) should be configured as SC3 or SC4.

Parameter Name	Value	
OsMemoryStartAddress	User Defined	Changeable
OsMemoryEndAddress	User Defined	Changeable
OsMemoryAccesType	User Defined	Changeable

1) OsMemoryStartAddress
   Parameter indicating the start label in the linker file.
2) OsMemoryEndAddress
   Parameter indicating the End label in the linker file.
3) OsMemoryAccesType
   Parameter indicating the type of the access the Application has READ access is granted by default with WRITE access.

5.11 Timing Protection

If SWP configured SC2 or SC4 in OsScalabilityClass(see 5.1.1) is delivered, only Execution Budget can configure and work. Because usage of all timing protection functions causes considerable performance loss. To use other timing protection functions (i.e. Lock Budget and Time Frame), user should request delivery of SWP that necessary features are enabled.

5.11.1 OsTaskTimingProtection

Please refer Section 5.2.3

5.11.2 OsTaskResourceLock

Please refer Section 5.2.4

5.11.3 OsIsrTimingProtection

Please refer Section 5.3.2

5.11.4 OsIsrResourceLock

Please refer Section 5.3.3

5.12 Stack Monitoring

5.12.1 OsStackMonitoring

Please Refer ‘OsStackMonitoring’ in Section 4.1.1

5.13 Spinlock

5.13.1 OsSpinlock

An OsSpinlock object is used to co-ordinate concurrent access by TASKs/ISR2s on different cores to a shared resource.

Parameter Name	Value	
OsSpinlockAccessingApplication	User Defined	Changeable
OsSpinlockSuccessor	User Defined	Changeable

1) OsSpinlockAccessingApplication
   Reference to OsApplications that have an access to this object.
2) OsSpinlockSuccessor
   Reference to OsApplications that have an access to this object. To check whether a spinlock can be occupied (in a nested way) without any danger of deadlock, a linked list of spinlocks can be defined. A spinlock can only be occupied in the order of the linked list. It is allowed to skip a spinlock. If no linked list is specified, spinlocks cannot be nested.

5.14 IOC

5.14.1 OsIoc

This container contains the Configuration of the IOC (Inter OS Application Communicator).

Parameter Name	Value	
OsIocCbkStackSize	User Defined	Changeable

1) OsIocCbkStackSize
   This parameter defines the stack size of the IOC callback functions used by the Os.

5.14.2 OsIoc - OsIocCommunication

Representation of a 1:1 or N:1 communication between software parts located in different OS-Applications that are bound to the same or to different cores. The name shall begin with the name of the sending software service and be followed by a unique identifier delivered by the sending software service. In the case of RTE as user attention shall be paid on the fact that uniqueness for identifier names has to be reached over ports, data elements, object instances and maybe additional identification properties (E.g. Case 1:N mapping to 1:1). Example: - <NameSpace>_UniqueID

Parameter Name	Value	
OsIocBufferLength	User Defined	Changeable

1) OsIocBufferLength
   This attribute defines the size of the IOC internal queue to be allocated for a queued communication. This configuration information shall allow the optimization of the needed memory for communications requiring buffers within the RTE and within the IOC.

5.14.3 OsIoc – OsIocCommunication – OsIocDataProperties

Data properties of the data to be transferred on the IOC communication channel.

Parameter Name	Value	
OsIocDataPropertyIndex	User Defined	Changeable
OsIocInitValue	User Defined	Changeable
OsIocDataTypeRef	User Defined	Changeable

1) OsIocDataPropertyIndex
   This parameter is used to define in which order the data is send, e.g. whether IocSendGroup(A,B) or IocSendGroup(B,A) shall be used.
2) OsIocInitValue
   Initial Value for the data to be transferred on the IOC communication channel.
3) OsIocDataTypeRef
   This is the type of the data to be transferred on the IOC communication channel. This attribute is necessary to generate the parameter type of the Ioc functions. Additionally this information should be used to compute the data size for necessary data copy operations within the Ioc module. If more than one attribute is defined, the IOC generator should generate an IocXxxGroup function (Xxx= CHOICE [Send, Receive, Write, Read]). N:1 communication (Multiplicity of OsIocSenderProperties > 1) is only allowed for multiplicity of OsIocDataRef = 1

5.14.4 OsIoc – OsIocCommunication – OsIocReceiverProperties

Representation of receiver properties for one communication. For each OsIocCommunication (1:1 or N:1) one receiver has to be defined. This container should be instanciated within an OsIocCommunication.

Parameter Name	Value	
OsIocFunctionImplementationKind	User Defined	Changeable
OsIocReceiverPullCB	User Defined	Changeable
OsIocReceivingOsApplicationRef	User Defined	Changeable

1) OsIocFunctionImplementationKind
   This parameter is used to select whether this communication is implemented as a macro or as a function.
   - DO_NOT_CARE: It is not defined whether a macro or a function is used.
   - FUNCTION: Communication is implemented as a function.
   - MACRO: Communication is implemented as a macro.
2) OsIocReceiverPullCB
   This attribute defines the name of a callback function that the IOC shall call on the receiving core for each data reception. In case of non existence of this attribute no ReceiverPullCB notification shall be applied by the IOC. The name of the function shall begin with the name of the receiving module, followed with a callback name and followed by the IocId. Example: void RTE_ReceiverPullCB_RTE25 (void). If this attribute does not exist, it means that no ReceiverPullCB shall be called (No notification from IOC is required). If this attribute exists the IOC shall call the callback function on the receiving core.
3) OsIocReceivingOsApplicationRef
   This attribute is a reference to the receiving OsApplication instance defined in the configuration file of the OS. This information allows for the generator to get additional information necessary for the code generation like: * The protection properties of the communicating OsApplications to find out which protections have to be crossed * The core identifiers to find out if an intra or an inter core

communication has to be realized * Interrupt details in case of cross core notification to realize over IRQs

5.14.5 OsIoc - OsIocCommunication - OsIocSenderProperties

Representation of sender properties for one communication. For each OsIocCommunication one (1:1) or many senders (N:1) have to be defined. Multiplicity > 1 (N:1 communication) is only allowed for Multiplicity of OsIocDataTypeRef = 1.

Parameter Name	Value	
OsIocFunctionImplementationKind	User Defined	Changeable
OsIocSenderId	User Defined	Changeable
OsIocSendingOsApplicationRef	User Defined	Changeable

1) OsIocFunctionImplementationKind
   This parameter is used to select whether this communication is implemented as a macro or as a function.
2) OsIocSenderId
   Representation of a sender in a N:1 communication to distinguish between senders. This parameter does not exist in 1:1 communication.
3) OsIocSendingOsApplicationRef
   This attribute is a reference to the sending OS-Application instance definedin the configuration file of the OS. This information shall allows the generator to get additional information necessary for the code generation like: * The protection properties of the communicating OS-Applications to find out which protection boundaries have to be crossed. * The core identifiers to find out if an intra or an inter core communication has to be realized * Interrupt details in case of cross core notification to realize over IRQs

5.15 Deviation

1) OsTask – OsTaskActivation
   AutoEver SWP does not support multiple activation due to AutoEver policy, so this value is restricted to 1.

2) OsCounter - OsDriver
   AutoEver OS doesn't support GPT as counter source. Instead of GPT, AutoEver OS uses SYSTICK (System Timer) to handle OS counter.

3) OsResource
   AutoEver OS doesn't support "Linked Resource".

4) OsOS - OsUseResScheduler
   In AutoEver OS, "RES_SCHEDULER" can't be used with "Resource lock budget" which is a part of Timing Protection. So "OsUseResScheduler" should be set as "False" when "OsTaskResourceLock"(see 5.2.4) or "OsIsrResourceLock"(see 5.3.3) is configured.

5) OsScheduleTable
   AutoEver SWP does not support Schedule Table due to AutoEver policy.

6) OsApplication - OsApplicationTrustedFunction
   AutoEver SWP does not support OsApplicationTrustedFunction due to AutoEver policy.

6. Application Programming Interface (API)

6.1 Task Management

6.1.1 General

Complex control software can conveniently be subdivided in parts executed according to their real-time requirements. These parts can be implemented by the means of tasks. A task provides the framework for the execution of functions. The operating system provides concurrent and asynchronous execution of tasks. The scheduler organizes the sequence of task execution.
The AUTOSAR operating system provides a task switching mechanism including a mechanism which is active when no other system or application functionality is active. This mechanism is called idle-mechanism. Two different task concepts are provided by the AUTOSAR operating system:

- Basic tasks
- Extended tasks

The primary difference between a basic task and an extended task is whether the task can go into a waiting state (in which it is waiting for an event to occur). Only extended tasks can wait for an event. Basic tasks must run to completion unless pre-empted. Pre-emptive tasks can be pre-empted by a higher-priority task becoming ready to run or by an interrupt. Non-pre-emptive tasks can only be pre-empted by an interrupt.

6.1.1.1 Basic Tasks

Basic tasks only release the processor, if

- They terminate
- The AUTOSAR operating system switches to a higher-priority task, or
- An interrupt occurs which causes the processor to switch to an interrupt service routine (ISR)

The Basic task state model is shown below in figure 6.

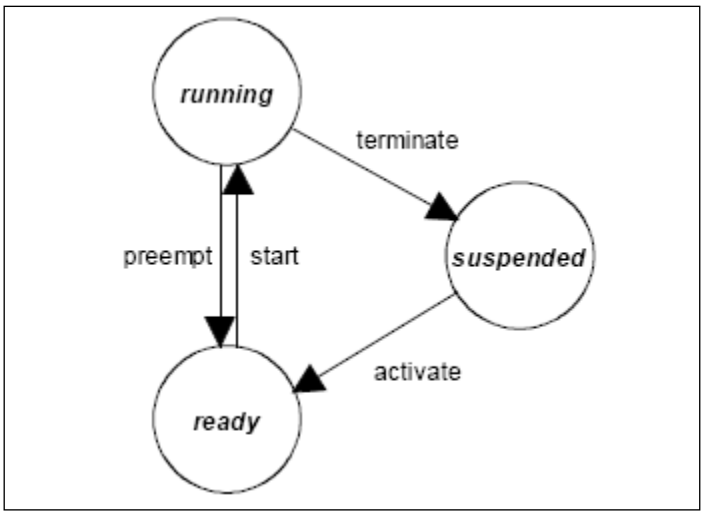


Figure 5: Basic task state model

The explanation of the various states in basic task state model is as follows:
Running: In the running state, the CPU is assigned to the task, so that its instructions can be executed. Only one task can be in this state at any point in time, while all the other states can be adopted simultaneously by several tasks.
Ready: All functional prerequisites for a transition into the running state exist, and the task only waits for allocation of the processor. The scheduler decides which ready task is executed next.
Suspended: In the suspended state the task is passive and can be activated.

6.1.1.2 Extended Tasks

Extended tasks are distinguished from basic tasks by being allowed to use the operating system call Wait Event, which may result in a waiting state. The waiting state allows the processor to be released and to be reassigned to a lower-priority task without the need to terminate the running extended task.
The extended task state model is as shown in figure 7.

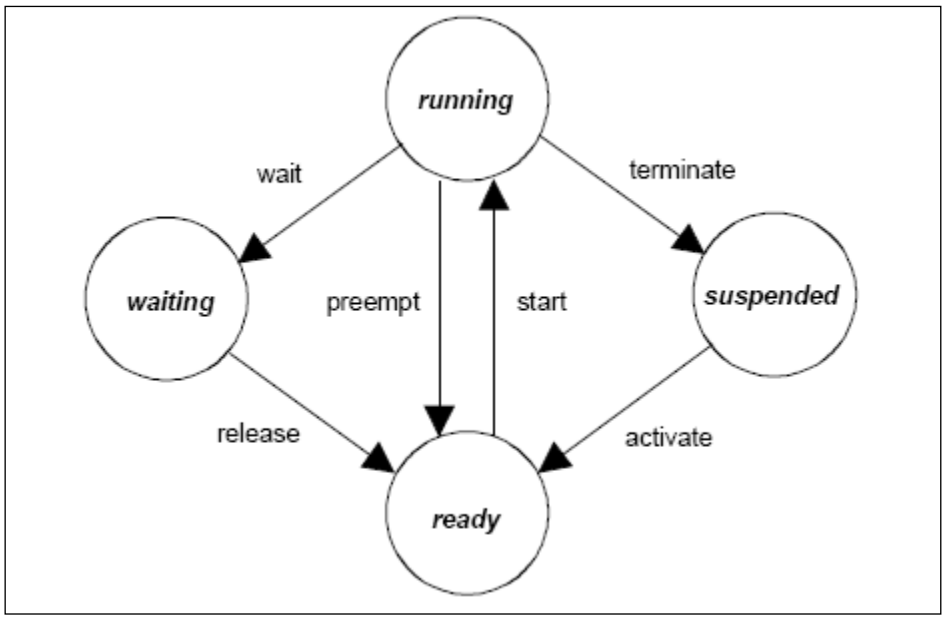


Figure 6: Extended task state model

The explanation of various states in extended task is provided in the given Table 5.

Transition	Former state	New state	Description
activate	suspended	ready	A new task is entered into the ready list by a system service.
start	ready	running	A ready task selected by the scheduler is executed.
wait	running	waiting	To be able to continue operation, the running task requires an event. It causes its transition into the waiting state by using a system service.
release	waiting	ready	Events have occurred which a task has waited on.
preempt	running	ready	The scheduler decides to start another task. The running task is put into the ready state.
terminate	running	suspended	The running task causes its transition into the suspended state by a system service.

Table 4: States and status transitions in the case of Extended Tasks

6.1.2 Data Types

The OSEK Operating System establishes the following data types for the task management:

TaskType	
Type:	Scalar
Range:	0-255
Description:	This data type identifies a task.
TaskRefType	
Type:	Pointer
Range:	NA
Description:	This data type points to a variable of TaskType.
TaskStateType	
Type:	Scalar
Range:	NA
Description:	This data type identifies the state of a task.
TaskStateRefType	
Type:	Pointer
Range:	NA
Description:	This data type points to a variable of the data type TaskStateType.

6.1.3 Run Time Services

This section describes the APIs that includes functionalities of TASK in OS Module.

6.1.3.1 ActivateTask

ActivateTask		
Prototype:	StatusType ActivateTask (TaskType TaskID)	
Service ID:	OSServiceId_ ActivateTask	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskType	TaskID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_LIMIT
Description:	This service transfers the task <TaskID> from the suspended state into the ready state. The operating system ensures that the task code is being executed from the first statement.	
Configuration Dependency:	Task should be configured in OsTask container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ActivateTask () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{
   StatusType      LenStatus;
   TaskType         Task1;

   /* Service activates the Task */
   LenStatus = ActivateTask (Task1);

}
```

6.1.3.2 TerminateTask

TerminateTask		
Prototype:	StatusType TerminateTask (void)	
Service ID:	OSServiceId_TerminateTask	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OS_CALLEVEL E_OS_RESOURCE E_OS_SPINLOCK
Description:	This service causes the termination of the calling task. The calling task is transferred from the running state into the suspended state.	
Configuration Dependency:	Task should be configured in OsTask container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ActivateTask () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void Task1 (void)
{
   StatusType      LenStatus;
      /* Service terminates the task */
   LenStatus = TerminateTask ();
}
```

6.1.3.3 ChainTask

ChainTask		
Prototype:	StatusType ChainTask (TaskType TaskID)	
Service ID:	OSServiceId_ChainTask	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskType	TaskID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OS_ID E_OS_LIMIT E_OS_CALLEVEL E_OS_RESOURCE E_OS_SPINLOCK
Description:	This service causes the termination of the calling task. After termination of the calling task a succeeding task with <TaskID> is activated. Using this service, it ensures that the succeeding task starts to run at the earliest after the calling task has been terminated.	
Configuration Dependency:	Task should be configured in OsTask container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ChainTask () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType      LenStatus;
   TaskType         Task2;

   /* Service terminates Task1 and activates Task2 */
   LenStatus = ChainTask (Task2);
}
```

6.1.3.4 Schedule

Schedule		
Prototype:	StatusType Schedule (void)	
Service ID:	OSServiceId_Schedule	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_CALLEVEL E_OS_RESOURCE E_OS_SPINLOCK
Description:	If a higher-priority task is ready, this service causes release of the internal resource of the task, the current task is put into the ready state, its context is saved and the higher-priority task is executed. Otherwise the calling task is continued.	
Configuration Dependency:	None	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking Schedule () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
void Task1 (void)
{
   StatusType      LenStatus;

   /* Service releases internal resource and caused scheduling */
   LenStatus = Schedule ();

}
```

6.1.3.5 GetTaskID

GetTaskID		
Prototype:	StatusType GetTaskID (TaskRefType TaskID)	
Service ID:	OSServiceId_GetTaskID	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	TaskRefType	TaskID
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK
Description:	This service returns the information about the TaskID of the task which is currently running.	
Configuration Dependency:	Task should be configured in OsTask container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetTaskID () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType      LenStatus;
   TaskType         Task2;

   /* Service returns the TASKID of the running task */
   LenStatus = GetTaskID (&Task1);

}
```

6.1.3.6 GetTaskState

GetTaskState		
Prototype:	StatusType GetTaskState (TaskType TaskID, TaskStateType State)	
Service ID:	OSServiceId_GetTaskState	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskType	TaskID
Parameters (out)	TaskStateType	State
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID
Description:	This service returns the state of a task (running, ready, waiting, suspended) at the time of calling.	
Configuration Dependency:	Task should be configured in OsTask container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetTaskState () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void Task1 (void)
{
   StatusType      LenStatus;
   TaskType         Task1;
   StatusType      State;

   /* Service returns the state of Task1 */
   LenStatus = GetTaskState (Task1, &state1);

}
```

6.2 Scheduler

6.2.1 General

The algorithm deciding which task has to be started and triggering all necessary OSEK Operating System

internal activities is called scheduler. It performs all actions to switch CPU from one instruction thread to another. It is either switching from task to task or from ISR back to a task. The task execution sequence is controlled on the base of task priorities and the scheduling policy used.
The scheduling policy being used determines whether execution of a task may be interrupted by other tasks or not. In this context, a distinction is made between full-, non- and mixed-preemptive scheduling policies.

6.2.1.1 Non-Preemptive scheduling policy

The scheduling policy is considered as non-preemptive, if a task switch is only performed via one of a selection of explicitly defined system services (explicit point of rescheduling). Non-preemptive scheduling imposes particular constraints on the possible timing requirements of tasks. Specifically the non-preemptive section of a running task with lower priority delays the start of a task with higher priority up to the next point of rescheduling. The time diagram of the task execution sequence for this policy looks like the following:

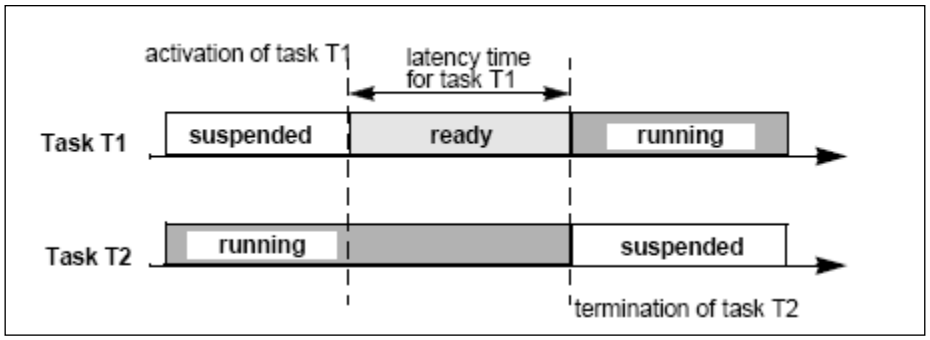


Figure 7: Non-Preemptive scheduling

Task T2 has the lower priority than task T1. Therefore, it delays task T1 up the point of rescheduling (in this case termination of task T2). The following points of rescheduling exist in the OSEK Operating System:

- Successful termination of a task (via the TerminateTask system service)
- Successful termination of a task with explicit activating of a successor task (via the ChainTask system service)
- Explicit call of the scheduler (via the Schedule system service)
- Explicit wait call, if a transition into the waiting state takes place (via the WaitEvent system service, Extended Tasks only)

In the non-preemptive system all tasks are non-preemptive and the task switching will take place exactly in the listed cases.

6.2.1.2 Full-Preemptive scheduling policy

Full-preemptive scheduling means that a task which is presently running may be rescheduled at any instruction by the occurrence of trigger conditions preset by the operating system. Full-preemptive scheduling will put the running task into the ready state, as soon as a higher-priority task has got ready. The task context is saved so that the preempted task can be continued at the location where it was interrupted.

With full-preemptive scheduling the latency time is independent of the run time of lower priority tasks. Certain restrictions are related to the increased RAM space required for saving the context, and the enhanced complexity of features necessary for synchronization between tasks. In full-preemptive system all tasks are preemptive.

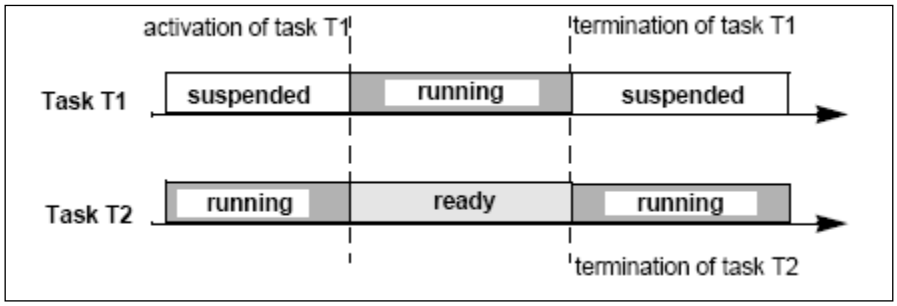


Figure 8: Full-Preemptive scheduling

6.2.1.3 Mixed-preemptive Scheduling

If full-preemptive and non-preemptive scheduling principles are to be used for execution of different tasks on the same system, the resulting policy is called "mixed-preemptive" scheduling. The distinction is made via the task property (preemptive/non-preemptive).
The definition of a non-preemptive task makes sense in a full-preemptive operating system in the following cases:

- if the execution time of the task is in the same magnitude of the time of a task switch
- if RAM is to be used economically to provide space for saving the task context
- if the task must not be preempted

Many applications comprise only few parallel tasks with a long execution time, for which a full-preemptive operating system would be convenient, and many short tasks with a defined execution time where non-preemptive scheduling would be more efficient. For this configuration the mixed-preemptive scheduling policy was developed as a compromise.
AUTOSAR OS has extended the configuration of scheduling policy per Task. Thus a configuration parameter suggesting the scheduling algorithm for a Task is available in OsTask container.

6.2.2 Data types

NA

6.2.3 Run Time Services

NA

6.3 Interrupt Processing

6.3.1 General

Interrupt processing is the important part of any real-time operating system. An Interrupt Service Routine (ISR) is a routine which is invoked from an interrupt source, such as a timer or an external hardware event. ISRs have higher priority than all tasks and the scheduler. Addresses of ISRs should be pointed in the vector table.

In OSEK OS all ISRs should use the separate stack (ISR stack) which is used only by ISRs during their execution. The size of the ISR stack is defined by the user. At the beginning of an Interrupt Service Routine the user should switch to this stack using the system service EnterISR. After the ISR completion the corresponded service LeaveISR should be performed to switch back to the previous stack.
ISRs can communicate with tasks by the following means:

- ISR can activate a task;
- ISR can send a state or an event message to a task;
- ISR can trigger a counter

Interrupts cannot use any OS services except those which are specially allowed to be used within ISRs.
In the OSEK Operating System two types of Interrupt Service Routines are considered:

Category 1: This ISR does not use an operating system service. After the ISR is finished, processing continues exactly at the instruction where the interrupt has occurred, i.e. the interrupt has no influence on task management. ISRs of this category have the least overhead. Category 1 interrupts are highest priority interrupts. OS services can not be called in CAT1 interrupts.

Category 2: This ISR is handled by the OS. CAT2 interrupts can use most of the OS calls unlike CAT1 interrupt. This ISR has higher latency than CAT1 interrupt.

Following figure shows the difference between the CAT1 and CAT2 Interrupts.

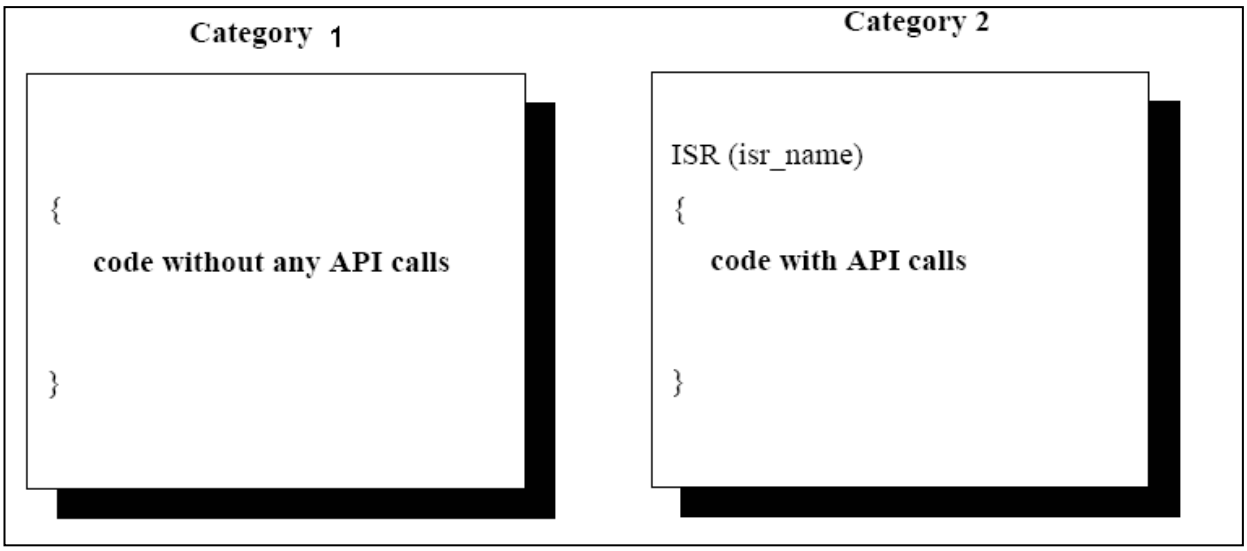


Figure 9: Difference between the CAT1 and CAT2 interrupts.

6.3.2 Data Types

6.3.2.1 ISRType

ISR Type	
Type:	Scalar
Range	0 to 255
Description:	This data type identifies an interrupt service routine. It is being used as “uint8”.
Constants of this Type:	INVALID_ISR

6.3.3 Run Time Services

This section describes the APIs provided by the Os Module to handle Interrupts.

6.3.3.1 GetISRID

GetISRID		
Prototype:	ISRType GetISRID (void)	
Service ID:	OSServiceId_GetISRID	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	ISRType	<Identifier of running OsIsr> INVALID_ISR
Description:	This service if called from category 2 OsIsr (or Hook routines called inside a category 2 OsIsr), returns the identifier of the currently executing OsIsr	
Configuration Dependency:	None	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*********************************************************************
 An example Application Software invoking GetISRID() API
 *********************************************************************/
#include “Os.h”    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void ISR_OsIsr1 (void)
{
   ISRType ISRId;
   /* Service returns running ISRID */
   ISRId = GetISRID();

}
```

6.3.3.2 EnableAllInterrupts

EnableAllInterrupts		
Prototype:	Void EnableAllInterrupts (void)	
Service ID:	OSServiceId_EnableAllInterrupts	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service restores the state saved by DisableAllInterrupts	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking EnableAllInterrupts () API
 *******************************************************************************/
#include “Os.h”    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service enables the interrupts */
  EnableAllInterrupts();

}
```

6.3.3.3 DisableAllInterrupts

DisableAllInterrupts		
Prototype:	void DisableAllInterrupts (void)	
Service ID:	OSServiceId_DisableAllInterrupts	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service disables all interrupts for which the hardware supports disabling. The state before is saved for the *EnableAllInterrupts* call.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking DisableAllInterrupts () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service disables the interrupts */
  DisableAllInterrupts();

}
```

6.3.3.4 ResumeAllInterrupts

ResumeAllInterrupts		
Prototype:	void ResumeAllInterrupt (void)	
Service ID:	OSServiceId_ResumeAllInterrupt	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service restores the recognition status of all interrupts saved by the SuspendAllInterrupts service.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ResumeAllInterrupt () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service resumes the interrupts */
  ResumeAllInterrupt ();

}
```

6.3.3.5 SuspendAllInterrupts

SuspendAllInterrupts		
Prototype:	void SuspendAllInterrupt (void)	
Service ID:	OSServiceId_SuspendAllInterrupt	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service saves the recognition status of all interrupts and disables all interrupts for which the hardware supports disabling.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking SuspendAllInterrupts () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service resumes the interrupts */
  SuspendAllInterrupts();

}
```

6.3.3.6 ResumeOSInterrupts

ResumeOSInterrupts		
Prototype:	Void ResumeOSInterrupts (void)	
Service ID:	OSServiceId_ResumeOSInterrupts	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service restores the recognition status of interrupts saved by the SuspendOSInterrupts service	
Configuration Dependency:	None	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ResumeOsInterrupts () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service resumes the interrupts */
  ResumeOsInterrupts();

}
```

6.3.3.7 SuspendOsInterrupts

SuspendOsInterrupts		
Prototype:	Void SuspendOsInterrupts (void)	
Service ID:	OSServiceId_SuspendOsInterrupts	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This service saves the recognition status of interrupts of category 2 and disables the recognition of these interrupts.	
Configuration Dependency:	None	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking SuspendOsInterrupts () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{

   /* Service resumes the interrupts */
  SuspendOsInterrupts ();

}
```

6.4 Resource Management

6.4.1 General

The resource management is used to coordinate concurrent accesses of several tasks to shared resources. Resource management ensures that:

- two tasks cannot “own” the same resource at the same time
- priority inversion cannot arise while resources are used
- deadlocks do not occur by use of these resources
- access to resources never results in a waiting state

The functionality of resource management is only required in the following cases:

- full- or mixed-preemptive scheduling
- non-preemptive scheduling, if resources are also to remain occupied beyond a scheduling point (except the scheduler resource)
- non-preemptive scheduling, if the user intends to have the application code executed under other scheduling policies too

The OSEK operating system ensures that tasks are only transferred from the ready state into the running state, if all resources which might be occupied by that task during its execution have been released. Consequently, no situation occurs in which a task tries to access an occupied resource. The special mechanism is used by the OSEK Operating System to provide such behavior, see Figure 11 Priority Ceiling Protocol for details. The waiting state is not admissible for Extended Tasks while a resource is occupied. It means that the task occupying a resource is not allowed to call the WaitEvent service.In case of multiple resource occupation, the task has to request and release resources following the LIFO principle (stack).
In the OSEK Operating System resources are ranked by priority. Each resource is assigned statically to a user defined priority which is called Ceiling Priority. It is possible to have resources with the same priorities, but the resource Ceiling Priority has to be identical or higher to the highest task priority with access to this resource. This resource feature supports the Priority Ceiling Protocol.

6.4.1.1 Priority Ceiling Protocol

The Priority Ceiling Protocol is implemented in the OSEK Operating System as a resource management discipline. When a task occupies a resource the system temporary changes its priority. It is automatically set to the Ceiling Priority by the resource management. Any other task which might occupy the same resource does not enter the running state due to its lower or equal priority. If the resource occupied by the task is released, the task returns to its former priority level. Other tasks which might occupy this resource can now enter the running state. The example shown in Figure 11 illustrates the mechanism of the Priority Ceiling Protocol.

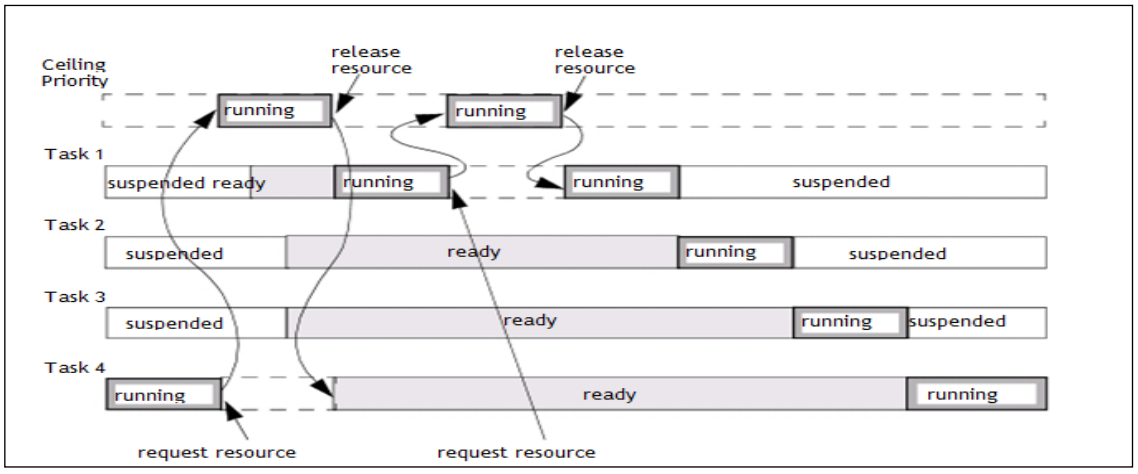


Figure 10: Priority Ceiling Protocol

In the figure above Task 1 has the highest priority; Task 4 has the lowest Priority. The resource has the priority greater than or equal to the Task 1 priority. When Task 4 occupies the resource it gets the priority not less then

Task 1, therefore it cannot be preempted by ready Task 1 until it release the resource. Just after the resource is released, Task 4 is returned to its low priority and becomes ready, and Task 1 becomes the running task. When Task 1, in its turn, occupies the resource, its priority is also changed to the Ceiling Priority.

6.4.2 Data Types

ResourceType	
Type:	Scalar
Range	0-255
Description:	This data type identifies a Resource.It is being used as “uint8”

6.4.3 Run Time services

This section describes the APIs provided by the Os Module for Resource functionalities:

6.4.3.1 GetResource

GetResource		
Prototype:	StatusType GetResource (ResourceType <ResID>)	
Service ID:	OSServiceId_GetResource	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	ResourceType	ResID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_ACCESS
Description:	This call serves to enter critical sections in the code that are assigned to the resource referenced by <ResID>. A critical section shall always be left using ReleaseResource.	
Configuration Dependency:	Resource should be configured in OsResource container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetResource () API
 *******************************************************************************/
```

```
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType              LenStatus;
   ResourceType         Resource1;

   /* Service assigns resource1 to task1 */
   LenStatus = GetResource (Resource1);

}
```

6.4.3.2 ReleaseResource

ReleaseResource		
Prototype:	StatusType ReleaseResource (ResourceType <ResID>)	
Service ID:	OSServiceId_GetResource	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	ResourceType	ResID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_ACCESS E_OS_NOFUNC
Description:	This service is the counterpart of GetResource and serves to leave critical sections in the code that are assigned to the resource referenced by <ResID>.	
Configuration Dependency:	Resource should be configured in OsResource container.	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ReleseResource () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
```

```
}

void Task1 (void)
{
   StatusType              LenStatus;
   ResourceType         Resource1;

   /* Service releases resource1 */
   LenStatus = ReleseResource (Resource1);
}
```

6.5 Counters

6.5.1 General

Any event in the system can be linked with a counter. It means, when the event is occurred, the counter value is changed. A counter is identified in the system via its symbolic name which is assigned to the counter statically at the configuration stage.They are defined by the user. At least one counter always exists in the system. This counter is used as a system timer (the internal system clock). The system timer is a standard counter with the following additions:

- the user must always define the system timer in an application
- special constants are defined to describe counter parameters and to decrease access time
- the user defines the source of hardware interrupts for the system counter

6.5.2 Data Types

6.5.2.1 CounterType

CounterType	
Type:	Scalar
Range	0 – 4294967296
Description:	This data type identifies a counter. It is being used as “uint8” “uint16” or “uint32” as per the number Of counters configured.

6.5.2.2 PhysicalTimeType

PhysicalTimeType	
Type:	Scalar
Range	0-255
Description:	This data type is used for values returned by the conversion macro OS_TICKS2<Unit>_<Counter> (). It is being used as “uint8”.

6.5.3 Run Time services

This section describes the APIs provided by the Os Module for Counter functionalities.

6.5.3.1 IncrementCounter

IncrementCounter		
Prototype:	StatusType IncrementCounter (CounterType CounterID)	
Service ID:	OSServiceId_IncrementCounter	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	CounterType	CounterID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID
Description:	This service increments the counter <CounterID> by one (if any alarm connected to this counter expires, the given action, e.g. task activation, is done) and shall returns E_OK.	
Configuration Dependency:	Counter should be configured in OsCounter container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking IncrementCounter () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{
   StatusType LenStatus;
   CounterType Counter1;

   /* Service increments the counter by 1 */
   LenStatus = IncrementCounter (Counter1);

}
```

6.5.3.2 GetCounterValue

GetCounterValue		
Prototype:	StatusType GetCounterValue (CounterType CounterID, TickRefType Value)	
Service ID:	OSServiceId_ GetCounterValue	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	CounterType	CounterID
Parameters (out)	TickRefType	Value
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID
Description:	This service returns the current tick value of the counter via <Value> and return E_OK.	
Configuration Dependency:	Counter should be configured in OsCounter container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetCounterValue () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{
   StatusType      LenStatus;
   CounterType    Counter1;
   TickType         Value1;

   /* Service returns the counter value */
   LenStatus = GetCounterValue (Counter1, &Value1);
}
```

6.5.3.3 GetElapsedCounterValue

GetElapsedCounterValue		
Prototype:	StatusType GetElapsedCounterValue (CounterType CounterID, TickRefType Value, TickRefType ElapsedValue)	
Service ID:	OSServiceId_GetElapsedCounterValue	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	CounterType	CounterID
Parameters (Out)	TickRefType	ElapsedValue
Parameters (InOut)	TickRefType	Value
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_VALUE
Description:	This service returns the number of elapsed ticks since the given <Value> value via <ElapsedValue>.In the <Value> parameter the current tick value of the counter is returned.	
Configuration Dependency:	Counter should be configured in OsCounter container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking GetElapsedCounterValue () API
 *******************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void application_main (void)
{
   StatusType      LenStatus;
   CounterType    Counter1;
   TickType         Value1;
   TickType         ElapsedValue;

   /* Service returns the elapsed counter value */
   LenStatus = GetElapsedCounterValue (Counter1, &Value1, &ElapsedValue);

}
```

6.6 Alarm

6.6.1 General

The AUTOSAR operating system provides services for processing recurring events. Such events may be for example timers that provide an interrupt at regular intervals, or encoders at axles that generate an interrupt in case of a constant change of a (camshaft or crankshaft) angle, or other regular application specific triggers.
The AUTOSAR operating system provides a two-stage concept to process such events. The recurring events (sources) are registered by implementation specific counters. Based on counters, the AUTOSAR operating system software offers alarm mechanisms to the application software.
Layered model of alarm management is shown in figure 13.

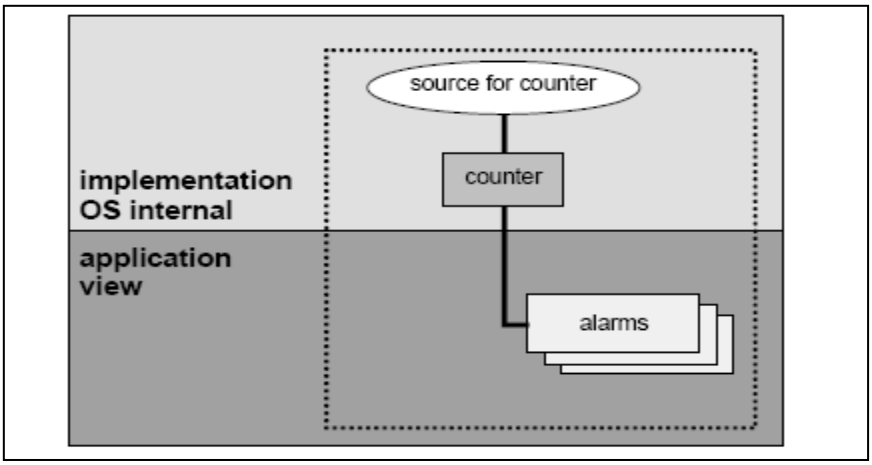


Figure 11: Layered model of alarm management

Each alarm belongs to a counter which is often derived from a hardware timer. A counter counts hardware ticks (it can be prescaled) and updates a current date. When an alarm becomes ACTIVE (as a result of SetRelAlarm or SetAbsAlarm system calls) OS computes its expiration date and inserts it in its counter alarm queue. When the date of the counter equals the next alarm expiration date, the alarm expires and is removed from the queue. At expiration time, an alarm may have three behaviors: 1) a task activation, 2) a callback routine execution, 3) an event setting. A cyclic alarm is put back in the queue after date computation. Since counters have a maximum value, date computation is done modulus this maximum value and a date may be lower than the current date although it is in the future. The counter keeps a pointer to the next alarm. When the pointer reaches the end of the queue, it is reset to the beginning. Figure 14 shows the data structures involved in alarm management.

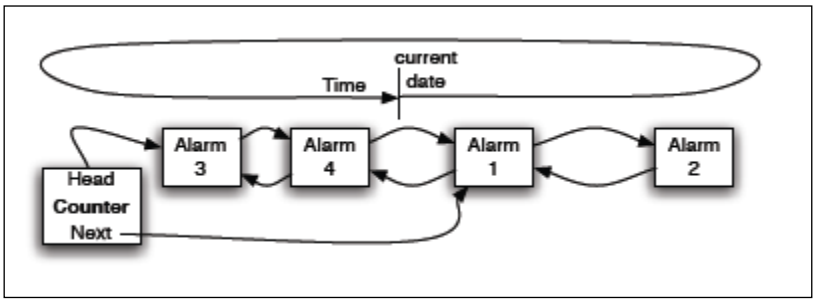


Figure 12: Alarm management

Each counter manages an alarm list. Head points to the first alarm of the list and Next to the nearest alarm. When the current date reaches the date of the alarm pointed by Next, the alarm expires, is removed from the queue and the code corresponding to its behavior is executed. In both cases Next and Head are updated if necessary.

6.6.2 Data Types

TickType	
Type:	Scalar
Range	0 – 4294967296
Description:	This data type represents count values in ticks. It is being used as “uint32”.
TickRefType	
Type:	Pointer
Range	-
Description:	This data type points to the data type TickType. It is being used as pointer to variable.
AlarmBaseType	
Type:	Structure of scalars
Range	-
Description:	This data type represents a structure for storage of counter characteristics. The individual elements of the structure are: **Maxallowedvalue** : Maximum possible allowed count value in ticks **Ticksperbase:** Number of ticks required to reach a counter-specific (significant) unit. **Mincycle:** Smallest allowed value for the cycle-parameter of SetRelAlarm/SetAbsAlarm) (only for systems with extended status).
AlarmBaseRefType	
Type:	Pointer
Range	-
Description:	This data type points to the data type AlarmBaseType. It is being used as pointer to variable.
AlarmType	
Type:	Scalar
Range	0 – 4294967296
Description:	This data type represents an alarm object. It is being used as “uint8” “uint16” or “uint32” as per the no. Of Alarms configured.

6.6.3 Run Time services

This section describes the APIs provided by the Os Module for Alarm functionalities.

6.6.3.1 GetAlarmBase

GetAlarmBase		
Prototype:	StatusType GetAlarmBase (AlarmType <AlarmID>, AlarmBaseRefType <Info>)	
Service ID:	OSServiceId_GetAlarmBase	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AlarmType	AlarmID
Parameters (out)	AlarmBaseRefType	Info
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID
Description:	The system service GetAlarmBase reads the alarm base characteristics. The return value <Info> is a structure in which the information of data type AlarmBaseType is stored.	
Configuration Dependency:	Alarm should be configured in OsAlarm container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetAlarmBase () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
  AalrmType          Alarm1;
  AlarmBaseType    Info;

   /* Service returns alarm information   */
   LenStatus = GetAlarmBase(Alarm1,&Info);

}
```

6.6.3.2 GetAlarm

GetAlarm		
Prototype:	StatusType GetAlarm (AlarmType <AlarmID>, TickRefType <Tick>)	
Service ID:	OSServiceId_GetAlarm	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AlarmType	AlarmID
Parameters (out)	TickRefType	Tick
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_NOFUNC
Description:	The system service GetAlarm returns the relative value in ticks before the alarm <AlarmID> expires.	
Configuration Dependency:	Alarm should be configured in OsAlarm container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetAlarm () API
 *******************************************************************************/
#include “Os.h”    /* Os Module Header file*/
Void main (void)
{

  StartOS (Mode1);

}
void Task1 (void)
{
  AaIrmType          Alarm1;
  TickType            Tick1;

   /* Service returns Alarm tick */
   LenStatus = GetAlarm(Alarm1, &Tick);

}
```

6.6.3.3 SetRelAlarm

SetRelAlarm		
Prototype:	StatusType SetRelAlarm (AlarmType <AlarmID>, TickType <Increment>, TickType <cycle>)	
Service ID:	OSServiceId_SetRelAlarm	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AlarmType	AlarmID
	TickType	increment
	TickType	Cycle
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_STATE E_OS_VALUE
Description:	The system service occupies the alarm <AlarmID> element. After <increment> ticks have elapsed, the task assigned to the alarm <AlarmID> is activated or the assigned event (only for extended tasks) is set or the alarm-callback routine is called.	
Configuration Dependency:	Alarm should be configured in OsAlarm container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking SetRelAlarm () API
 *******************************************************************************/
#include “Os.h”    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
  AaIrmType          Alarm1;
  TickType            Tick1;
  TickType            Tick2;
   /*   Service starts the alarm at relative   value */
   LenStatus = SetRelAlarm (Alarm1, Tick1, Tick2);
}
```

6.6.3.4 SetAbsAlarm

SetAbsAlarm		
Prototype:	StatusType SetAbsAlarm (AlarmType <AlarmID>, TickType <Start>, TickType <cycle>)	
Service ID:	OSServiceId_SetRelAlarm	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AlarmType	AlarmID
	TickType	Start
	TickType	Cycle
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_STATE E_OS_VALUE
Description:	The system service occupies the alarm <AlarmID> element. When <start> ticks are reached, the task assigned to the alarm <AlarmID> is activated or the assigned event (only for extended tasks) is set or the alarm-callback routine is called.	
Configuration Dependency:	Alarm should be configured in OsAlarm container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking SetAbsAlarm () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
  AaIrmType          Alarm1;
  TickType            Tick1;
  TickType            Tick2;
   /* Service starts the alarm at absolute value */
   LenStatus = SetAbsAlarm (Alarm1, Tick1, Tick2);
  }
```

6.6.3.5 CancelAlarm

CancelAlarm		
Prototype:	StatusType CancelAlarm (AlarmType <AlarmID>,)	
Service ID:	OSServiceId_CancelAlarm	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AlarmType	AlarmID
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_NOFUNC
Description:	The system service cancels the alarm <AlarmID>.	
Configuration Dependency:	Alarm should be configured in OsAlarm container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking CancelAlarm () API
 *******************************************************************************/
#include “Os.h”    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
  AaIrmType          Alarm1;

   /* Service Cancels the Alarm */
   LenStatus = CancelAlarm (Alarm1);
}
```

6.7 Event Mechanism

6.7.1 General

Event mechanism

- is a means of synchronization
- is only provided for extended tasks
- is used to initiate state transitions of tasks to and from the waiting state

Events are objects managed by the operating system. They are not independent objects, but assigned to extended tasks. Each extended task has a definite number of events. This task is called the owner of these events. An individual event is identified by its owner and its name. When activating an extended task, these events are cleared by the operating system. Events can be used to communicate binary information to the extended task to which they are assigned. The meaning of events is defined by the application, e.g. signalling of an expiring timer, the availability of a resource, the reception of a message, etc.

Events are the criteria for the transition of extended tasks from the waiting state into the ready state. The operating system provides services for setting, clearing and interrogation of events and for waiting for events to occur.

Any task or ISR of category 2 can set an event for a not suspended extended task, and thus inform the extended task about any status change via this event.

The receiver of an event is an extended task in any case. Consequently, it is not possible for an interrupt service routine or a basic task to wait for an event. An event can only be cleared by the task which is the owner of the event. Extended tasks may only clear events they own, whereas basic tasks are not allowed to use the operating system service for clearing events.

An extended task in the waiting state is released to the ready state if at least one event for which the task is waiting for has occurred. If a running extended task tries to wait for an event and this event has already occurred, the task remains in the running state.

6.7.1.1 Event and scheduling

An event is an exclusive signal which is assigned to an Extended Task. For the scheduler, events are the criteria for the transition of Extended Tasks from the waiting state into the ready state. The operating system provides services for setting, clearing and interrogation of events, and for waiting for events to occur.

Extended Tasks are in the waiting state, if an event for which the task is waiting has not occurred. If an Extended Tasks tries to wait for an event and this event has already occurred, the task remains in the running state.

Figure 14 illustrates the procedures which are affected by setting an event: Extended Task 1 (with higher priority) waits for an event. Extended Task 2 sets this event for Extended Task 1. The scheduler is activated. Subsequently, Task 1 is transferred from the waiting state into the ready state. Due to the higher priority of Tasks 1 this results in a task switch, Task 2 being preempted by Task 1. Task 1 resets the event. Thereafter Task 1 waits for this event again and the scheduler continues execution of Task 2.

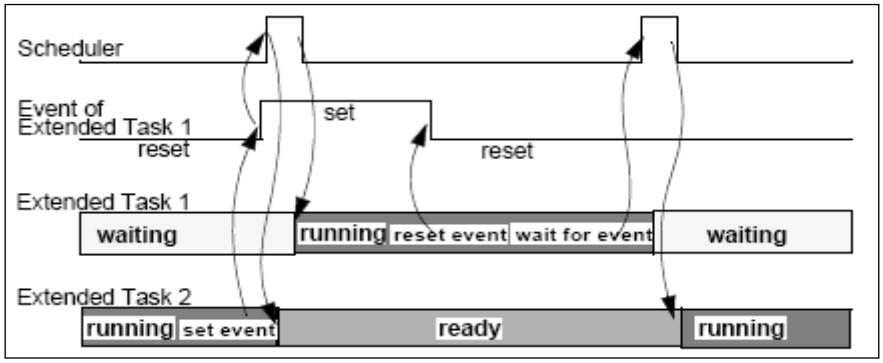


Figure 13: Synchronization of Extended Tasks by setting events in case of full-preemptive scheduling.

If non-preemptive scheduling is supposed, rescheduling does not take place immediately after the event has

been set as it is shown in Figure 15.

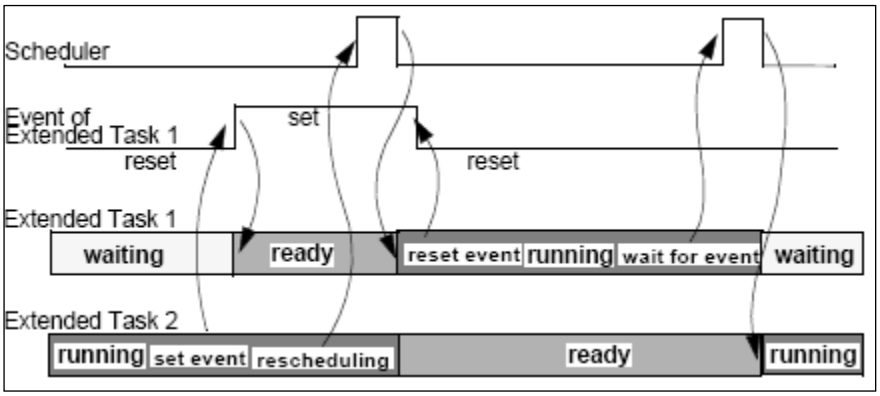


Figure 14: Synchronization of Extended Tasks by setting events in case of non-preemptive scheduling.

6.7.2 Data Types

EventMaskType	
Type:	Scalar
Range	0-255
Description:	This data type identifies an Event. It is being used as “uint8”.
EventMaskRefType	
Type:	Pointer
Range	-
Description:	This data type points to an EventMask. It is being used as pointer to variable.

6.7.3 Run Time services

This section describes the APIs provided by the Os Module for Event functionalities.

6.7.3.1 GetEvent

GetEvent		
Prototype:	StatusType GetEvent (TaskType <TaskID>, EventMaskRefType <Event>)	
Service ID:	OSServiceId_GetEvent	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskType	TaskID
Parameters (out)	EventMaskRefType	Event
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_ACCESS E_OS_STATE
Description:	This service returns the current state of all event bits of the task <TaskID>.	
Configuration Dependency:	Event should be configured in OsEvent container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking GetEvent () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType              LenStatus;
   EventMaskType         Event0;

   /* Service returns state of the event */
   LenStatus = GetEvent(Task2,&Event0);
}
```

6.7.3.2 WaitEvent

WaitEvent		
Prototype:	StatusType WaitEvent (EventMaskRefType <Event>)	
Service ID:	OSServiceId_WaitEvent	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	EventMaskRefType	Event
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_RESOURCE E_OS_ACCESS E_OS_CALLEVEL
Description:	This service The state of the calling task is set to waiting, unless at least one of the events specified in <Mask> has already been set.	
Configuration Dependency:	Event should be configured in OsEvent container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking WaiEvent () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType              LenStatus;
   EventMaskType         Event0;

   /* Service causes task to wait for the event   */
   LenStatus = WaitEvent(&Event0);

}
```

6.7.3.3 SetEvent

SetEvent		
Prototype:	StatusType SetEvent (TaskType <TaskID>, EventMaskType <Mask>)	
Service ID:	OSServiceId_SetEvent	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskType	TaskID
	EventMaskType	Mask
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ID E_OS_ACCESS E_OS_STATE
Description:	The events of task <TaskID> are set according to the event mask <Mask>. Calling SetEvent causes the task <TaskID> to be transferred to the ready state, if it was waiting for at least one of the events specified in <Mask>.	
Configuration Dependency:	Event should be configured in OsEvent container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking SetEvent () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType              LenStatus;
   EventMaskType         Mask;

   /* Service sets the event   */
   LenStatus = SetEvent(Task2,Mask);

}
```

6.7.3.4 ClearEvent

ClearEvent		
Prototype:	StatusType ClearEvent (EventMaskType <Mask>)	
Service ID:	OSServiceId_ClearEvent	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	EventMaskType	Mask
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK E_OS_ACCESS E_OS_CALLEVEL
Description:	The events of the extended task calling ClearEvent are cleared according to the event mask <Mask>.	
Configuration Dependency:	Event should be configured in OsEvent container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************
 An example Application Software invoking ClearEvent () API
 *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}

void Task1 (void)
{
   StatusType              LenStatus;
   EventMaskType        Mask;

   /* Service clears the Event */
   LenStatus = ClearEvent (Mask);

}
```

6.8 Application Modes

6.8.1 General

Application modes are designed to allow an AUTOSAR operating system to come up under different modes of operation. The minimum number of supported application modes is one. It is intended only for modes of operation that are totally mutually exclusive. An example of two exclusive modes of operation would be end-of-line programming and normal operation. Once the operating system has been started, it shall not be allowed to change the application mode.

6.8.2 Data Types

NA

6.8.3 Run Time services

6.8.3.1 GetActiveApplicationMode

GetActiveApplicationMode		
Prototype:	AppModeType GetActiveApplicationMode (void)	
Service ID:	OSServiceId_GetActiveApplicationMode	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	AppModeType	AppModeID
Description:	This service returns the current application mode. It may be used to write mode dependent code.	
Configuration Dependency:	Application should be configured in OsApplication container	
Pre-conditions:	StartOS () API should be called	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
- An example Application Software invoking GetActiveApplicationMode () API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
```

```
}

void Task1 (void)
{
  ApplicationModeType   Application;

   /* Service returns Application id */
   Application = GetActiveApplicationMode (void);

}
```

6.9 OS-Application

6.9.1 General

An AUTOSAR OS must be capable of supporting a collection of Operating System objects (Tasks, ISRs, Alarms, Schedule tables, Counters, Resources) that form a cohesive functional unit. This collection of objects is termed an OS-Application. The Operating System module is responsible for scheduling the available processing resource between the OS-Applications that share the processor. If OS-Application(s) are used, all Tasks, ISRs, Resources, Counters, Alarms and Schedule tables must belong to an OS-Application. 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.
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 Operating System module's 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 Operating System module's 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 Operating System module itself is trusted.

OS-Applications have a state which defines the scope of accessability of its Operating System objects from other OS-Applications. Each OS-Application is always in one of the following states:

- **APPLICATION_ACCESSIBLE:** Operating System objects may be accessed from other OS-Applications. This is the default state at startup which mean active and accessible.
- **APPLICATION_RESTART:** Operating System objects can not be accessed from other OS-Applications. State is valid until the OS-Application calls AllowAccess().
- **APPLICATION_TERMINATED:** Operating System objects can not be accessed from other OS-Applications. State will not change.

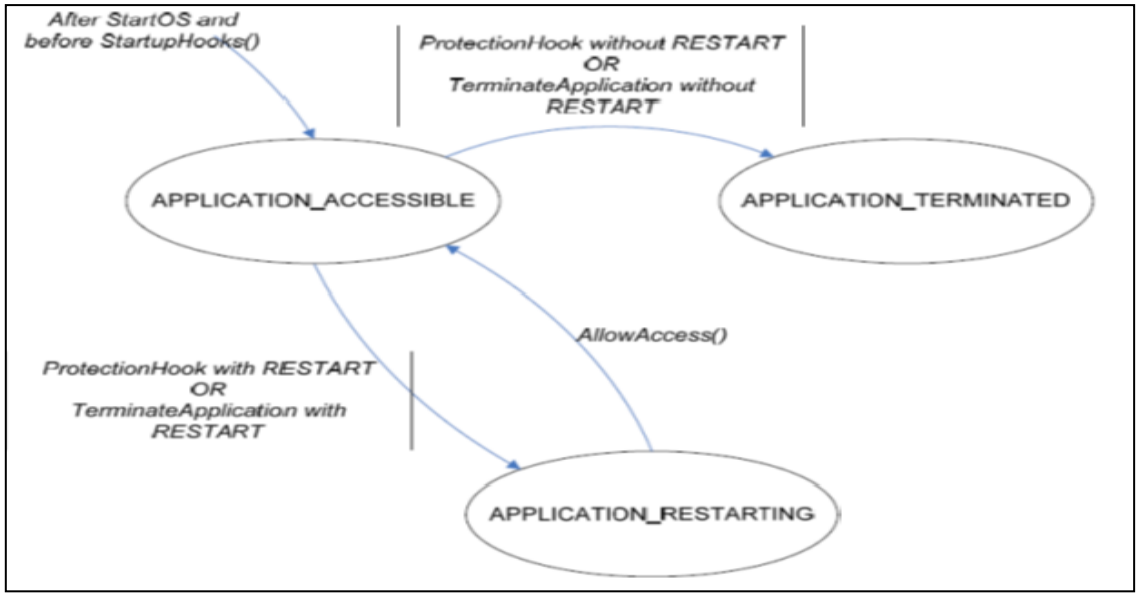


Figure 15: OS-Application States

6.9.2 Data Types

6.9.2.1 ApplicationType

Application Type	
Type:	Scalar
Range	0 to 255
Description:	This data type identifies the OS-Application. It is being used as “uint8”.
Constants of this Type:	INVALID_OSAPPLICATION

6.9.2.2 ApplicationStateType

ApplicationStateType	
Type:	Scalar
Range:	0 to 255
Description:	This data type identifies the state of an OS-Application.
Constants of this Type:	APPLICATION_ACCESSIBLE APPLICATION_RESTARTING APPLICATION_TERMINATED

6.9.2.3 ApplicationStateRefType

ApplicationStateRefType	
Type:	Pointer
Range:	-
Description:	This data type points to location where a ApplicationStateType can be stored. It is being used as pointer to variable.
Constants of this Type:	-

6.9.2.4 RestartType

RestartType	
Type:	Scalar
Range	0 to 1
Description:	This data type defines the use of a Restart Task after terminating an OS-Application. It is being used as “uint8”.
Constants of this Type:	RESTART NO_RESTART

6.9.2.5 AccessType

AccessType	
Type:	Integral
Range:	0-255
Description:	This type holds information how a specific memory region can be accessed.
Constants of this Type:	-

6.9.2.6 ObjectAccessType

ObjectAccessType	
Type:	Scalar
Range:	0 - 255
Description:	This data type identifies if an OS-Application has access to an object.
Constants of this Type:	ACCESS NO_ACCESS

6.9.2.7 ObjectTypeType

ObjectTypeType	
Type:	Scalar
Range:	0 – 255
Description:	This data type identifies an object.
Constants of this Type:	OBJECT_TASK OBJECT_ISR OBJECT_ALARM OBJECT_RESOURCE OBJECT_COUNTER OBJECT_SCHEDULETABLE

6.9.2.8 MemorySizeType

MemoryStartAddressType	
Type:	Scalar
Range:	0 – 4294967296
Description:	This data type holds the size (in bytes) of a memory region.
Constants of this Type:	-

6.9.2.9 MemoryStartAddressType

MemoryStartAddressType	
Type:	Pointer
Range:	-
Description:	This data type is a pointer which is able to point to any location in MCU address space.
Constants of this Type:	It is used as pointer to variable.

6.9.3 Run Time services

6.9.3.1 GetApplicationID

GetApplicationID		
Prototype:	ApplicationType GetApplicationID(void)	
Service ID:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	TType	Possible Return Values
	ApplicationType	<identifier of running OS-Application> or INVALID_OSAPPLICATION
Description:	This service determines the currently running OS-Application (a unique identifier has to be allocated to all application.	
Configuration Dependency:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking GetApplicationID() API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main(void)
{
startOS(Mode1);
}
void application_main()
{
                ApplicationType      LenStatus;
                /* Set Application ID with ID of application */

                /* Return the Application ID */
                return(LddAppId);
}
```

6.9.3.2 CheckISRMemoryAccess

CheckISRMemoryAccess		
Prototype:	AccessType CheckISRMemoryAccess(ISRType ISRID, MemoryStartAddressType Address, MemorySizeType Size)	
Service ID:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	ISRType, MemoryStartAddressType, MemorySizeType	ISRID Address Size
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	AccessType	Value which contains the access rights to the memory area.
Description:	This service checks if a memory region is write/read/execute accessible and also returns information if the memory region is part of the stack space.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking CheckISRMemoryAccess () API
************* *******************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main(void)
{
startOS(Mode1);
}
void application_main()
{
  StatusType LenStatusReturn;
  /* Set LenStatusReturn to ACCESS */
  LenStatusReturn = ACCESS;
  /* Check if LenStatusReturn is ACCESS */
  if(LenStatusReturn != NO_ACCESS)
  {
    /* Call CheckMemoryAccess */
    LenStatusReturn = CheckMemoryAccess(ISRID, Address, Size);
  }
  /* Return the Status */
  return(LenStatusReturn);
}
```

6.9.3.3 CheckTaskMemoryAccess

CheckTaskMemoryAccess		
Prototype:	AccessType CheckTaskMemoryAccess(TaskType TaskID, MemoryStartAddressType Address, MemorySizeType Size)	
Service ID:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	TaskID, Address, Size	Task reference , Start of memory area, Size of memory area
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	AccessType	Value which contains the access rights to the memory area.
Description:	This service checks if a memory region is write/read/execute accessible and also returns information if the memory region is part of the stack space.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking CheckTaskMemoryAccess () API
************* ********************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main(void)
{
startOS(Mode1);
}
void application_main()
{
  StatusType LenStatusReturn;
  /* Set LenStatusReturn to ACCESS */
  LenStatusReturn = ACCESS;
  /* Check if LenStatusReturn is ACCESS */
  if(LenStatusReturn != NO_ACCESS)
  {
    /* Call CheckMemoryAccess */
    LenStatusReturn = CheckMemoryAccess(TaskID, Address, Size);
  }
  /* Return the Status */
  return(LenStatusReturn);
}
```

6.9.3.4 CheckObjectOwnership

CheckObjectOwnership		
Prototype:	ApplicationType CheckObjectOwnership(ObjectTypeType ObjectType, void ...)	
Service ID:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	ObjectType	The type of following parameter, The object to be examined
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	ApplicationType	<OS-Application>: the OS-Application to which the object ObjectType belongs or INVALID_OSAPPLICATION if object doesnot exists.
Description:	This service determines to which OS-Application a given Task, ISR, Resource, Counter, Alarm or schedule Table belongs.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking CheckObjectOwnership () API
************* ********************************************************************************************/

#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void application_main()
{
  StatusType LenStatusReturn;
  ApplicationType ApplicationID;
  /* Set Return status to ACCESS */
  LenStatusReturn = ACCESS;
  /* Check if LenStatusReturn is ACCESS or not */
  if (LenStatusReturn == ACCESS)
  {
      switch(ObjectType)
      {
        case OBJECT_TASK:
                        /* Get the application Id from the structure */
```

```
                        break;

        case OBJECT_ISR:
                        /* Get the application Id from the structure */
                        break;

                    case OBJECT_ALARM:
                        /* Get the application Id from the structure */
                        break;

        case OBJECT_RESOURCE:
                        /* Get the application Id from the structure */
                        break;
        case OBJECT_COUNTER:
                        /* Get the application Id from the structure */
                        break;
        case OBJECT_SCHEDULETABLE:
                        /* Get the application Id from the structure */
                        break;
        default:      break;
      }
  }
  return(ApplicationID);
}
```

6.9.3.5 TerminateApplication

TerminateApplication		
Prototype:	StatusType TerminateApplication(ApplicationType Application, RestartType RestartOption)	
Service ID:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	RestartOption Application	Either RESTART for doing a restart of the OS-Application or NO_RESTART if OS-Application shall not be restarted The identifier of the OS-Application to be terminated. If the caller belongs to <Application> the call results in a self termination.
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK: No errors E_OS_ID: <Application> was not valid E_OS_VALUE: <RestartOption> was neither RESTART nor NO_RESTART E_OS_ACCESS: The caller does not have the right to terminate <Application> E_OS_STATE: The state of <Application> does not allow terminating <Application>
Description:	This service terminates the OS-Application to which the calling Task/Category 2 ISR/application specific error hook belongs	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking TerminateApplication() API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void application_main()
{
  StatusType LenStatusReturn;
```

```
  /* Set Return status to E_OK */
  LenStatusReturn = E_OK;
  if (LenStatusReturn == E_OK)
  {
    /* Store the Running Task in local variable */
    /* Terminate all the objects of Application */
  }
  /* Return the value */
  return(LenStatusReturn);
}
```

6.9.3.6 AllowAccess

AllowAccess		
Prototype:	StatusType AllowAccess (void)	
Service ID:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK: No errors E_OS_STATE: The OS-Application of the caller is in the wrong state
Description:	This service sets the own state of an OS-Application from APPLICATION_RESTARTING to APPLICATION_ACCESSIBLE.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*****************************************************************************************
 An example Application Software invoking AllowAccess() API
************* *****************************************************************************/

#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void application_main()
{
  StatusType LenStatusReturn;
  ApplicationType AppId;
```

```
  /* Set the Status to E_OS_STATE */
  LenStatusReturn = E_OS_STATE;
  /* Store Application ID in Local variable */
  /* Check if state of Application is Restarting */
  if (...)
  {
    /* Update state with Accessible */
  }
  /* Return the Status */
  return(LenStatusReturn);
}
```

6.9.3.7 GetApplicationState

GetApplicationState		
Prototype:	StatusType GetApplicationState(ApplicationType Application, ApplicationStateRefType Value)	
Service ID:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	Application	The OS-Application from which the state is requested,
Parameters (out)	Value	The current state of the application
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK: No errors E_OS_ID: <Application> was not valid
Description:	This service returns the current state of an OS-Application	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*****************************************************************************************
 An example Application Software invoking GetApplicationState() API
************* ****************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
void application_main()
```

```
{
  StatusType LenStatusReturn;
  /* Set Return status to E_OK */
  LenStatusReturn = E_OK;
  /* Check whether status return is E_OK */
  if (LenStatusReturn == E_OK)
  {
    /* Update the state of the Application */
  }
  /* Return the value */
  return (LenStatusReturn);
}
```

6.10 Timing Protection

6.10.1 General

In a real- time system, when a task or interrupt misses its deadlines at runtime, a timing fault occurs. AUTOSAR Os does not offer deadline monitoring for timing protection. When a deadline is violated this may be due to a timing fault introduced by an unrelated Task or interrupt that interferes or block for too long. 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.
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 inter-arrival rate of Task/OsIsrs in the system

To counter the above factors following budgets are considered in AUTOSAR OS:
Execution Budget:
AUTOSAR OS prevents timing errors from the execution time of Task or Interrupt by using execution time protection to guarantee a statically configured upper bound, called the Execution Budget, on the execution time of Task and Interrupts.
Lock Budget:
AUTOSAR OS prevents timing errors from the blocking time that task or interrupt suffers from lower priority tasks or interrupts, 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

TimeFrame:
AUTOSAR OS prevents timing errors from the inter-arrival rate of task or interrupt, by using inter-arrival time protection to guarantee statically configured lower bounds, called the Time Frame, on the time between:

1. A task being permitted to transition into the READY state due to: Activation and Release (transition from wait state to ready state)
2. A Category 2 OsIsr arriving.
   An arrival occurs when the Category 2 OsIsr is recognized by the OS

6.10.2 Data Types

NA

6.10.3 Run Time services

NA

6.11 Service Protection

6.11.1 General

As OS-Applications can interact with the Operating System module through services, it is essential that the service calls will not corrupt the Operating System module itself. Service Protection guards against such corruption at runtime.
Cases which are considered with Service Protection:
An OS-Application makes an API call

- With an invalid handle or out of range value.
- In the wrong context, e.g. calling ActivateTask() in the StartupHook().
- Or fails to make an API call that result in the OSEK OS being left in an undefined state.
- that impacts on the behaviour of every other OS-Application in the system,
- For e.g, ShutdownOS()
- To manipulate Operating System 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.

6.11.2 Data Types

6.11.2.1 AccessType

Please refer Section 5.11.2.5

6.11.2.2 ObjectAccessType

Please refer Section 5.11.2.6

6.11.2.3 ObjectTypeType

Please refer Section 5.11.2.7

6.11.3 Run Time services

6.11.3.1 CheckObjectAccess

CheckObjectAccess		
Prototype:	ObjectAccessType CheckObjectAccess (ApplicationType AppIID, ObjectTypeType ObjectType, void ...)	
Service ID:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	AppIID, ObjectType,	OS-Application identifier, The type of following parameter, The object to be examined
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	ObjectAccessType	ACCESS if the AppIID has access to the object NO_ACCESS otherwise
Description:	This service determines if the OS-Applications, given by AppIID, is allowed to use the IDs of a Task, ISR, Resource, Counter, Alarm or Schedule Table in API calls.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking CheckObjectAccess () API
************* ***************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main (void)
{
  StartOS (Mode1);
}
{
  StatusType LenStatusReturn;
  ObjectTypeIndex LddObjectId;
  ObjectAccessType LenStatusReturn;
  /* Set Return status to ACCESS */
  LenStatusReturn = ACCESS;
  /* Check if LenStatusReturn is ACCESS or not */
  if (LenStatusReturn == ACCESS)
  {
     LenStatusReturn = NO_ACCESS;
     switch(ObjectType)
     {
       case OBJECT_TASK:
                          break;
       case OBJECT_ISR :
```

```
                    break;
        case OBJECT_ALARM:
                    break;
        case OBJECT_RESOURCE:
                    break;
        case OBJECT_COUNTER:
                    break;
        case OBJECT_SCHEDULETABLE:
                    break;
        default:
                    break;

    }
      /* Grant the access */
      LenStatusReturn = ACCESS;
    }
  }
  /* Return the value */
  return(LenStatusReturn);
}
```

6.12 Memory Protection

6.12.1 General

Memory protection will only be possible on processors that provide hardware support for memory protection. The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in a device. Using region descriptors which define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.
The memory protection scheme is based on the (data, code and stack) sections of the executable program.

1. **Data:** OS-Applications can have private data sections and Tasks/ISRs can have private data sections. OS-Application's private data sections are shared by all Tasks/ISRs belonging to that OS-Application.
2. **Stack:** An OS-Application comprises a number of Tasks and ISRs. The stack for these objects, by definition, belongs only to the owner object and there is therefore no need to share stack data between objects, even if those objects belong to the same OS-Application.

Memory protection for the stacks of Tasks and ISRs is useful mainly for two reasons:

- Provide a more immediate detection of stack overflow and underflow for the Task or ISR than can be achieved with stack monitoring
- Provide protection between constituent parts of and OS-Application, for example to satisfy some safety constraints.

3. **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.

6.12.2 Data Types

6.12.2.1 MemorySizeType

Please Refer Section 6.9.2.8

6.12.2.2 MemoryStartAddressType

Please Refer Section 6.9.2.9

6.12.3 Run Time servicesNA

6.13 Stack Monitoring

6.13.1 General

The processors which 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 ISR 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/ISR that was executing that used more than stack space than required – it could be a lower priority object that was pre-empted.

6.13.2 Data TypesNA

6.13.3 Run Time services

NA

6.14 Hook Routine

6.14.1 General

The AUTOSAR operating system provides system specific hook routines to allow user-defined actions within the OS internal processing.
Those hook routines are

- called by the operating system, in a special context depending on the implementation of the operating system
- Higher priority than all tasks
- Not interrupted by category 2 interrupt routines
- Part of the operating system implemented by the user with user defined functionality
- standardised interface, but not standardised in functionality (environment and behaviour of the hook routine itself), therefore usually hook routines are not portable
- are only allowed to use a subset of API functions
- In the AUTOSAR operating system hook routines are used for:
- System start-up. The corresponding hook routine (StartupHook) is called after the operating system start-up and before the scheduler is running
- System shutdown. The corresponding hook routine (ShutdownHook) is called when a system shutdown is requested by the application or by the operating system in case of a severe error
- Tracing or application dependent debugging purposes as well as user defined extensions of the context switch
- Error handling

6.14.1.1 Error hook routine:

The error hook routine (ErrorHook) is called if a system service returns a StatusType value not equal to E_OK. The hook routine ErrorHook is not called if a system service is called from the ErrorHook itself (i.e., a recursive call of error hook never occurs). Any possibly occurring error by calling system services from the ErrorHook can only be detected by evaluating the return value. ErrorHook also is called if an error is detected during task activation or event setting, for example upon alarm expiration or message arrival.

6.14.1.2 Debugging

Two hook routines (PreTaskHook and PostTaskHook) are called on task context switches. These two hook routines may be used for debugging or time measurement (including context switch time). Therefore PostTaskHook is called each time directly before the old task leaves the RUNNING state; PreTaskHook is called each time directly after a new task enters the RUNNING state. Because the task is still/already in the RUNNING state, GetTaskId does not return INVALID_TASK.

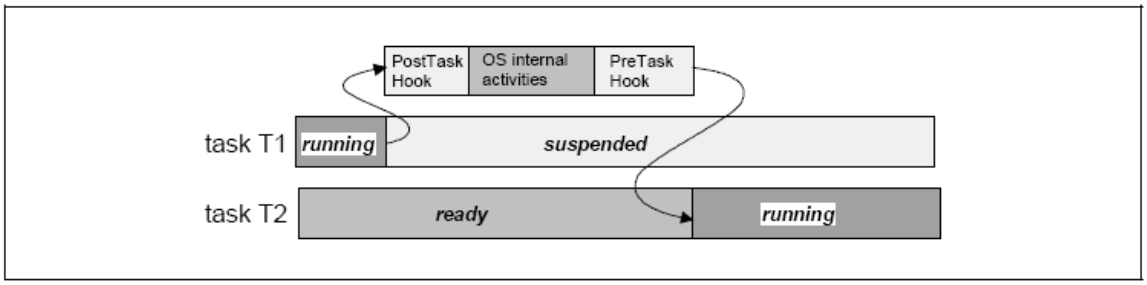


Figure 16: PreTaskHook and PostTaskHook

When ShutdownOS is called while a task is running ShutdownOS may or may not call PostTaskHook. If PostTaskHook is called it is undefined if it is called before or after ShutdownHook.

6.14.2 Data Types

6.14.2.1 StatusType

StatusType	
Type:	Scalar
Range:	0-103
Description:	This data type is used for all status information the API services offer. Naming convention: all errors for API services start with E_. Those reserved for the operating system will begin with E_OS_. To show the difference in use, the names internal errors shall start with E_OS_SYS_ It is being used as “uint8”.

Constants of this Type:

Error Code	Value	Description
E_OK	0	No error, successful completion
E_OS_ACCESS	1	Access to the service/object denied
E_OS_CALLEVEL	2	Access to the service from the ISR is not permitted
E_OS_ID	3	The object ID is invalid
E_OS_LIMIT	4	The limit of services/objects exceeded
E_OS_NOFUNC	5	The object is not used, the service is rejected
E_OS_RESOURCE	6	The task still occupies the resource
E_OS_STATE	7	The state of the object is not correct for the required service
E_OS_VALUE	8	A value outside of the admissible limit
E_OS_STACKFAULT	9	Stack fault detected via stack monitoring by the OS
E_OS_PROTECTION_ARRIVAL	10	Task/Category 2 ISR arrives before its timeframe has expired
E_OS_PROTECTION_TIME	11	Task/Category 2 ISR exceeds its execution time budget
E_OS_PROTECTION_LOCKED	12	Task/Category 2 ISR exceeds Resource or ISRs lock time
E_OS_DISABLEDINT	13	OS service is called inside an interrupt disable/enable pair
E_OS_PROTECTION_EXCEPTION	14	Trap occurred
E_OS_CORE	15	All functions that are not allowed to operate cross core shall return E_OS_CORE in extended status if called with parameters that require a cross core operation
E_OS_INTERFERENCE_DEADLOCK	16	The function GetSpinlock shall return this error if the spinlock referred by the parameter SpinlockID is already occupied by a TASK/ISR2 on the same core.
E_OS_NESTING_DEADLOCK	17	A TASK tries to occupy the spinlock while holding a different spinlock in a way that may cause a deadlock.
E_OS_SPINLOCK	18	This error means de-scheduling with occupied spinlock
E_OS_SERVICEID	19	Service can not be called
OS_E_PARAM_POINTER	20	A pointer argument to an API is null
E_OS_PROTECTION_MEMORY	21	Memory access violation occurred
E_OS_ILLEGAL_ADDRESS	22	An invalid address is given as a parameter to a service
E_OS_SYS_ALARM_INUSE	23	Counter interrupt is nested
E_OS_SYS_RAMECC	24	An ECC error has occurred on the RAM
E_OS_SYS_DFLASHECC	25	An ECC error has occurred on the Data Flash
E_OS_SYS_PFLASHECC	26	An ECC error has occurred on the Program Flash
E_OS_MISSINGEND	35	Tasks terminates without a TerminateTask() or ChainTask() call
E_OS_SYS_CORE_IS_DOWN	100	This error code means that the core is shutting down state.
E_OS_SYS_PANIC	101	This error code means that Inter-core message handling is fault.
E_OS_SYS_NMI	102	This error code means that NMI handling is fault.
E_OS_SYS_INTERCOREMSG	103	A problem occurred during the inter-core API request process.

6.14.2.2 ProtectionReturnType

ProtectionReturnType	
Type:	Scalar
Range:	0-4
Description:	This data type identifies a value which controls further actions of the OS on return from the protection hook. It is being used as “uint8”.
Constants of this Type:	PRO_IGNORE PRO_TERMINATETASKISR PRO_TERMINATEAPPL PRO_TERMINATEAPPL_RESTART PRO_SHUTDOWN

6.14.3 Run Time services

6.14.3.1 ApplicationSpecificStartupHook

StartupHook<App>		
Prototype:	void StartupHook_<App>(void)	
Service ID:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	The application specific startup hook is called during the start of the OS (after the user has started the OS via StartOS().	
Configuration Dependency:	None	
Pre-conditions:	None	

6.14.3.2 ApplicationSpecificErrorHook

GetActiveApplicationMode		
Prototype:	void ErrorHook_<App>(StatusType Error)	
Service ID:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	Error	The error which caused the call to the error hook
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	The application specific error hook is called whenever a Task or Category 2 ISR which belongs to the OS-Application causes an error.	
Configuration Dependency:	None	
Pre-conditions:	None	

6.14.3.3 ApplicationSpecificShutdownHook

ShutDownHook_<App>		
Prototype:	void ShutdownHook_<App>(StatusType Fatalerror)	
Service ID:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	Fatalerror	The error which caused the action to shut down the operating system.
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	The application specific shutdown hook is called whenever the system starts the Shut down of itself.	
Configuration Dependency:	None	
Pre-conditions:	None	

6.14.3.4 ProtectionHook

ProtectionHook		
Prototype:	ProtectionReturnType ProtectionHook(StatusType Fatalerror)	
Service ID:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	Fatalerror	The error which caused the call to the protection hook
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	ProtectionReturnType	PRO_IGNORE PRO_TERMINATETASKISR PRO_TERMINATEAPPL PRO_TERMINATEAPPL_RESTART PRO_SHUTDOWN The return value defines the action the OS shall take after the protection hook.
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.	
Configuration Dependency:	None	
Pre-conditions:	None	

6.15 Platform specific extensions

None

6.16 Spinlock

6.16.1 General

A spinlock is a busy waiting mechanism that polls a (lock) variable until it becomes available. Typically, this requires an atomic “test and set” functionality. Once a lock variable is occupied by a TASK/ISR2, other TASKs/ISR2s on other cores shall be unable to occupy the lock variable. The spinlock mechanism will not de-schedule these other TASKs while they poll the lock variable. However it might happen that a TASK/ISR with a higher priority becomes ready while the lock variable is being polled. In such cases the spinning TASK will be interfered.

6.16.2 Data Types

6.16.2.1 SpinlockIdType

SpinlockIdType	
Type:	Scalar
Range:	1-65535
Description:	SpinlockIdType identifies a spinlock instance and is used by the API functions: GetSpinlock, ReleaseSpinlock and TryToGetSpinlock.

6.16.2.2 TryToGetSpinlockType

TryToGetSpinlockType		
Type:	Enumeration	
Range:	TRYTOGETSPINLOCK_SUCCESS	Spinlock successfully occupied
	ETSPINLOCK_NOSUCCESS	Unable to occupy the spinlock
Description:	The TryToGetSpinlockType indicates if the spinlock has been occupied or not.	

6.16.3 Run Time services

6.16.3.1 GetSpinlock

GetSpinlock		
Prototype:	StatusType GetSpinlock(SpinlockIdType SpinlockId)	
Service ID:	OS_ServiceID_GetSpinlock	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	SpinlockId	The value refers to the spinlock instance that shall be locked.
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK - In standard and extended status : No Error E_OS_ID - In extended status: The SpinlockId is invalid E_OS_INTERFERENCE_DEADLOCK - In extended status: A TASK tries to occupy the spinlock while the lock is already occupied by a TASK on the same core. This would cause a deadlock. E_OS_NESTING_DEADLOCK - In extended status: A TASK tries to occupy the spinlock while holding a different spinlock in a way that may cause a deadlock. E_OS_ACCESS - In extended status: The spinlock cannot be accessed.
Description:	GetSpinlock tries to occupy a spin-lock variable. If the function returns, either the lock is successfully taken or an error has occurred. The spinlock mechanism is an active polling mechanism. The function does not cause a de-scheduling.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    GetSpinlock() API
*******************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main ()
{
StatusType GetSpinlock(SpinlockIdType SpinlockId)
   {
     if (LenStatusReturn == E_OK)
     {
       Os_GetSpinlock(SpinlockId);
     }
  }
Return LenStatusReturn
}
```

6.16.3.2 ReleaseSpinlock

ReleaseSpinlock		
Prototype:	StatusType ReleaseSpinlock(SpinlockIdType SpinlockId)	
Service ID:	OS_ServiceID_ReleaseSpinlock	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	SpinlockId	The value refers to the spinlock instance that shall be locked.
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK - In standard and extended status: No Error E_OS_ID - In extended status: The SpinlockId is invalid. E_OS_STATE - In extended status: The Spinlock is not occupied by the TASK E_OS_ACCESS - In extended status: The Spinlock cannot be accessed. E_OS_NOFUNC - In extended status: Attempt to release a spinlock while another spinlock has to be released before.
Description:	ReleaseSpinlock releases a spinlock variable that was occupied before. Before terminating a TASK all spinlock variables that have been occupied with GetSpinlock() shall be released. Before calling WaitEVENT all Spinlocks shall be released.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking   ReleaseSpinlock() API
************* *****************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
Void main ()
{
   StatusType ReleaseSpinlock(SpinlockIdType SpinlockId)
     {
      /* Check whether status return is E_OK */
      if (LenStatusReturn == E_OK)
      {
       /* Release the Spinlock */
       Os_ReleaseSpinlock(SpinlockId);
      }
 }
return (LenStatusReturn);
}
```

6.16.3.3 TryToGetSpinlock

TryToGetSpinlock		
Prototype:	StatusType TryToGetSpinlock(SpinlockIdType SpinlockId, TryToGetSpinlockType* Success)	
Service ID:	OS_ServiceID_TryToGetSpinlock	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	SpinlockId	The value refers to the spinlock instance that shall be locked.
Parameters (out)	Success	Returns if the lock has been occupied or not
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	StatusType	E_OK - In standard and extended status: No Error E_OS_ID - In extended status: The SpinlockId is invalid. E_OS_INTERFERENCE_DEADLOCK - In extended status: A TASK tries to occupy the spinlock while the lock is already occupied by a TASK on the same core. This would cause a deadlock. E_OS_NESTING_DEADLOCK - In extended status: A TASK tries to occupy a spinlock while holding a different spinlock in a way that may cause a deadlock. E_OS_ACCESS - In extended status: The spinlock cannot be accessed.
Description:	TryToGetSpinlock has the same functionality as GetSpinlock with the difference that if the spinlock is already occupied by a TASK on a different core the function sets the OUT parameter "Success" and returns with E_OK.	
Configuration Dependency:	None	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking   TryToGetSpinlock () API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
void Main (void)
{
  /* Check whether status return is E_OK */
  if (LenStatusReturn == E_OK)
  {
    GenStatusReturn1 = TryToGetSpinlock (SpinlockIdType SpinlockId, TryToGetSpinlockType* Success )
  }
return (LenStatusReturn);
}
```

6.17 Multicore

6.17.1 General

The Multi-Core OS in AUTOSAR is not a virtual ECU concept, instead it shall be understood as an OS that shares the same configuration and most of the code, but operates on different data structures for each core. The hardware only starts one core, referred as the master core, while the other cores (slaves) remain in halt state until they are activated by the software. In contrast to such a master-slave system other boot concepts with cores that start independently from each other are conceivable. However it is possible to emulate master-slave behavior on such systems by software. The master core is defined to be the core that requires no software activation, whereas a slave core requires activation by software.

In Multi-Core configurations, each slave core must be activated before StartOS is entered on the core. Depending on the hardware, it may be possible to only activate a subset of the available cores from the master. The slave cores might activate additional cores before calling StartOS. All cores that belong to the AUTOSAR system have to be activated by the designated AUTOSAR API function. Additionally, the StartOS function has to be called on all these cores.

6.17.2 Data Types

6.17.2.1 CoreIdType

CoreIdType		
Type:	Scalar	
Range:	OS_CORE_ID_MASTER	Refers to the master core, may be an alias for OS_CORE_ID_<x>
	OS_CORE_ID_0..OS_CORE_ID_65533	Refers to logical core 0, core 1 etc.
Description:	CoreIDType is a scalar that allows identifying a single core. The CoreIDType shall represent the logical CoreID.	

6.17.3 Run Time services

6.17.3.1 GetNumberOfActivatedCores

GetNumberOfActivatedCores		
Prototype:	uint32 GetNumberOfActivatedCores(void)	
Service ID:	OS_ServiceID_GetNumberOfActivatedCores	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Uint32	Number of cores activated by the StartCore function.
Description:	The function returns the number of cores activated by the StartCore function. This function might be a macro.	
Configuration Dependency:	None	
Pre-conditions:	None	

6.17.3.2 GetCoreID

GetCoreID		
Prototype:	CoreIdType GetCoreID(void)	
Service ID:	OS_ServiceID_GetCoreID	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	CoreIdType	The return value is the unique ID of the core.
Description:	The function returns a unique core identifier.	
Configuration Dependency:	None	
Pre-conditions:	None	

6.17.3.3 StartCore

StartCore		
Prototype:	void StartCore(CoreIdType CoreID, StatusType* Status)	
Service ID:	OS_ServiceID_StartCore	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant	
Parameters (In):	Type	Parameter
	CoreIDType	CoreID
Parameters (out)	StatusType	Return value of the function in extended status: E_OK: No Error E_OS_ID: Core ID is invalid. E_OS_ACCESS: The function was called after starting the OS. E_OS_STATE: The Core is already activated. Return value of the function in standard status E_OK: No Error
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	It is not supported to call this function after StartOS(). The function starts the core specified by the parameter CoreID. The OUT parameter allows the caller to check whether the operation was successful or not. If a core is started by means of this function StartOS shall be called on the core	
Configuration Dependency:	At least 2 cores should be present	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking   StartCore () API
************* *****************************************************************************************/
#include "Os.h"    /* Os Module Header file*/

void StartCore( CoreIdType CoreID, StatusType* Status )
{
  StatusType LenStatusReturn;
  LenStatusReturn = E_OK;
  if(LenStatusReturn == E_OK)
     {
       Os_StartCore(CoreID);
     }

  *Status = LenStatusReturn;
}
```

6.17.3.4 StartNonAutosarCore

StartNonAutosarCore		
Prototype:	void StartNonAutosarCore(CoreIdType CoreID, StatusType* Status)	
Service ID:	OS_ ServiceID_StartNonAutosarCore	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (In):	Type	Parameter
	CoreID	Core Identifier
Parameters (out)	Status	Return value of the function in standard status: E_OK: No Error E_OS_ID: Core ID is invalid. E_OS_STATE: The Core is already activated. Return value of the function in extended status E_OK: No Error
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	The function starts the core specified by the parameter CoreID. It is allowed to call this function after StartOS(). The OUT parameter allows the caller to check whether the operation was successful or not. It is not allowed to call StartOS on cores activated by StartNonAutosarCore. Otherwise the behaviour is unspecified.	
Configuration Dependency:	Non Autosar core should be present	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking  StartNonAutosarCore () API
************* *******************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
void StartCore( CoreIdType CoreID, StatusType* Status )
{
  StatusType LenStatusReturn;
  LenStatusReturn = E_OK;
  if(LenStatusReturn == E_OK)
  {
     Os_StartCore(CoreID);
  }
  *Status = LenStatusReturn;
}
```

6.17.3.5 ShutdownAllCores

ShutdownAllcores		
Prototype:	void ShutdownAllCores(StatusType Error)	
Service ID:	OS_ ServiceID_ShutdownAllCores	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (In):	Type	Parameter
	statusType	Error
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	After this service the OS on all AUTOSAR cores is shut down. Allowed at TASK level and ISR level and also internally by the OS. The function will never return. The function will force other cores into a shutdown	
Configuration Dependency:	At least 2 cores should be present	
Pre-conditions:	None	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    ShutdownAllCores () API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/

void ShutdownAllCores( StatusType Error)
{

}
```

6.18 IOC

6.18.1 General

The IOC described in this document provides communication between OS-Applications. The IOC generation is based on configuration information which is generated by the RTE generator. On the other hand the RTE uses functions generated by the IOC to transmit data.

6.18.2 Data Types

NA

6.18.3 Run Time services

This section describes the APIs provided by the Os Module and IOC module if configured for the communication between cores.

6.18.3.1 IocSend

IocSend		
Prototype:	Std_ReturnType IocSend_<IocId>[_<SenderId>](<Data> IN)	
Service ID:	IOCServiceId_IOC_Send	
Sync/Async:	Asynchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK IOC_E_LIMIT IOC_E_LOST_DATA
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************************
 An example Application Software invoking    IocSend() API
*******************************************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                    LucReturnVal = IocSend_<IOCID>(Data);
                   if(LucReturnVal == IOC_E_OK)
                   {
                   }
                   else if (LucReturnVal == IOC_E_LIMIT)
                   {
                     break;
                   }
```

```
}
```

6.18.3.2 IocWrite

IocWrite		
Prototype:	Std_ReturnType IocWrite_<IocId>[_<SenderId>](<Data> IN)	
Service ID:	IOCServiceId_IOC_Write	
Sync/Async:	Asynchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statutype	IOC_E_OK
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking IocWrite() API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
        /* Write data to IOC */
        LucReturnVal = IocWrite_<IOCID>(Data);
        if(LucReturnVal == IOC_E_OK)
        {

        }
}
```

6.18.3.3 IocSendGroup

IocSendGroup		
Prototype:	Std_ReturnType IocSendGroup_<IocId>(<Data1> IN1, <Data2> IN2, ...)	
Service ID:	IOCServiceId_IOC_SendGroup	
Sync/Async:	Asynchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK IOC_E_LIMIT IOC_E_LOST_DATA
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocSendGroup () API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                LucReturnVal = IocSendGroup_<IocId>(<Data1> IN1, <Data2> IN2, ... )
                if((LucReturnVal == IOC_E_OK) || (LucReturnVal == IOC_E_LIMIT))
                {
                  if (LucReturnVal == IOC_E_LIMIT)
                  {
                    break;
                  }
                }
 }
```

6.18.3.4 IocWriteGroup

IocWriteGroup		
Prototype:	Std_ReturnType IocWriteGroup_<IocId>(<Data1> IN1, <Data2> IN2, ...)	
Service ID:	IOCServiceId_IOC_WriteGroup	
Sync/Async:	Asynchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	. IOC_E_OK
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocWriteGroup() API
************ ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                /* Write data to IOC*/
                LucReturnVal = IocWriteGroup_<IOCID> (Data);
                if(LucReturnVal == IOC_E_OK)
                {

                }
  }
```

6.18.3.5 IocReceive

IocReceive		
Prototype:	Std_ReturnType IocReceive_<IocId>(<Data> OUT)	
Service ID:	IOCServiceId_IOC_Receive	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK IOC_E_LIMIT IOC_E_LOST_DATA
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocReceive() API
************* *******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                LucReturnVal = IocReceive_<IOCID> (Data);
                if((LucReturnVal == IOC_E_OK) &&(Data))
                {

                }
                else if ((LucReturnVal == IOC_E_LOST_DATA) ||
                                        (LucReturnVal == IOC_E_NO_DATA))
                { }
}
```

6.18.3.6 IocRead

IocRead		
Prototype:	Std_ReturnType IocRead_<IocId>()(<Data> OUT)	
Service ID:	IOCServiceId_IOC_Read	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked.	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocRead() API
************ ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                LucReturnVal =   IocRead_<IOCID> (Data);
                if(LucReturnVal == IOC_E_OK)
                {
                  if(Data1 == Data2)
                  {

                  }
                }
                break;
}
```

6.18.3.7 IocReceiveGroup

IocReceiveGroup		
Prototype:	Std_ReturnType IocReceiveGroup_<IocId>(<Data1> OUT1, <Data2> OUT2, ...)	
Service ID:	IOCServiceId_IOC_ReceiveGroup	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK IOC_E_NO_DATA IOC_E_LOST_DATA
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocReceiveGroup () API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
                LucReturnVal = IocReceiveGroup_<IOCID> (Data1, Data2...);

            if((LucReturnVal == IOC_E_OK) ||(LucReturnVal == IOC_E_LOST_DATA)
                                          || (LucReturnVal == IOC_E_NO_DATA))
               {
                 if((LucReturnVal == IOC_E_OK) ||(LucReturnVal == IOC_E_LOST_DATA))
                 {

              if((Data1) &&(Data2) &&...
                    {
                    }
                    if (LucReturnVal == IOC_E_LOST_DATA)
                    {
                    }
               }
```

```
            if(LucReturnVal == IOC_E_NO_DATA)
            {
              break;
            }
          }
}
```

6.18.3.8 IocReadGroup

IocReadGroup		
Prototype:	Std_ReturnType IocReadGroup_<IocId>(<Data1> OUT1,<Data2> OUT2, ...)	
Service ID:	IOCServiceId_IOC_ReadGroup	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK
Description:	This function is used to performs an "explicit" sender-receiver transmission of data elements with "event" semantic for a unidirectional 1:1 or N:1 communication between OS-Applications located on the same or on different cores.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocReadGroup () API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
            LucReturnVal = IocReadGroup_<IOCID> (&Data1, &Data2, &Data3...);
            if(LucReturnVal == IOC_E_OK)
            {
              if(Data1 == Data11)
              {

              }
              if(Data2 == Data21)
              {
```

```
        }
        if(Data3 == Data31)
        {

        }
    }
}
```

6.18.3.9 IocEmptyQueue

IocEmptyQueue		
Prototype:	Std_ReturnType IocEmptyQueue_<IocId>(void)	
Service ID:	IOCServiceId_IOC_EmptyQueue	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	Statustype	IOC_E_OK
Description:	In this Function the content of the IOC internal communication queue shall be deleted.	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    IocEmptyQueue () API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/

void Main (void)
{
        /* Empty queue if buffer is full */
        LucReturnVal = IocEmptyQueue_<IOCID>(void);
        if (LucReturnVal == IOC_E_OK)
        {

        }

}
```

6.18.3.10 ReceiverPullCB

ReceiverPullCB		
Prototype:	void <ReceiverPullCB>(void)	
Service ID:	IOCServiceId_IOC_ReceiverPullCB	
Sync/Async:	Synchronous	
Reentrancy:	Non Re-entrant	
Parameters (In):	Type	Parameter
	None	None
Parameters (out)	None	None
Parameters (InOut)	None	None
Return Value	Type	Possible Return Values
	None	None
Description:	This callback function can be configured for the receiver of a communication. If configured, IOC calls this callback on the receiving core for each data reception. <ReceiverPullCB> is the callback function name configured by the receiver in the OslocReceiverPullCB attribute to be called on data reception."	
Configuration Dependency:	This API is available only if Osloc container is configured.	
Pre-conditions:	StartOs () API should be invoked	

This example below is not applicable to the project because it is a simple reference.

```
/*******************************************************************************************
 An example Application Software invoking    ReceiverPullCB () API
************* ******************************************************************************/
#include "Os.h"    /* Os Module Header file*/
#include "Ioc.h"    /* Ioc Module Header file*/


void <ReceiverPullCB>
{

}
```

7. Generator

7.1 Getting started

This section provides the information regarding usage, input and output files of the Os Generation Tool.

Input File(s)
The Generation Tool accepts ECU Configuration Description File(s) and Module Description Template (MDT) as input. ECU Configuration Description File(s) contains information on AUTOSAR modules and MDT contains information on version information and common publish information. The Generation Tool will extract information pertaining to Os module.

ECU Configuration Description File(s) must be compliant to the AUTOSAR ECU Configuration Description File standards. The input file can be the description file generated from any ECU Configuration Editor. The MDT file is generated using the SysConf Tool.

Details of the ECU Configuration Description File(s) and MDT file are provided in Input Files. Details of the parameters in the ECU Configuration Description File(s) are provided in the Parameter Configuration section of each feature.

Output Files
Os Generation Tool generates Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, Os_Types.h, Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files. C Source and Header files are generated in the sub folders 'src' and 'inc' respectively. Detailed information of output files are provided in Output Files. Detailed information for parameter configuration is provided in parameter configuration section for each Os object. (Refer Respective Sections 5.x.4).

Options and Usage
This section provides the information regarding usage of the Os Generation Tool. It also provides the syntax of the command line arguments (input filename(s) and options).

Os Generation Tool executable is invoked as shown below:

{Os_Common.exe} 〈Options〉 〈ECU Configuration Description File〉 〈Module Description Template〉

{Os.exe} 〈Options〉 〈ECU Configuration Description File〉 〈Module Description Template〉

Where,

Os_Common.exe: Name of the Os Common Generation Tool Executable
Options: [-H/-Help -V/-Version -O/-OUTPUT -L/-Log -D/Dryrun -I/-Info -W/-Warn -prefix –RtelOCSpinlock -ORTI]
ECU Configuration Description File: {Input filename(s)}

Os: Name of the Os Generation Tool Executable
Options: [-O/-OUTPUT]
ECU Configuration Description File: {Input filename(s)}

Notations:

{data} represents compulsory data
〈data〉 represents the actual data that will be specified on command line during tool

usage.
[data] represents optional data.

Options:

Options	Description
-H/-Help	To display help regarding usage of the tool. Gets the highest priority when used with other options.
-V/-Version	To display the tool version. Gets the priority after –H/-Help option.
-O/-Output	By default, the tool generates output files in the 'Os_Output' folder in the path where executable is present. The user can use the -O option followed by the folder name, to generate the output files in an alternate folder. Either absolute path or relative path can be provided to specify the folder name. The C Source and Header files are generated in the sub folders 'src' and 'inc' respectively within the output folder.
-L/-Log	To log the output in Os.log file.
-D/-Dryrun	To execute tool in validation mode. The tool will not generate output files even though the input file provided is error free.
-I/-Info	To disable Information Messages. The Tool will not generate information messages on command line.
-W/-Warn	To disable Warning Messages. The Tool will not generate warning messages on command line.
-prefix	To generate TASK, RESOURCE, ISR prefix(OsConf_OsTask_, OsConf_OsResource_, OsConf_OsIsr_).
-RteIOCSpinlock	To generate RTE IOC Spinlock.
-ORTI	To generate orti file.

Table 5: Options and Description

	Os Generation Tool accepts any ECU Configuration Description File which complies with AUTOSAR ECU Configuration Description File standard Options are case insensitive. Options and filenames can appear on the command line in any order. Output directory should follow –O/-Output option. If command line options provided are correct and -L/-Log option is provided, then along with the output files, Os.log (log file) is generated either in the Default output directory 'Os_Output' (when –o option is not used) Output directory mentioned on the command line when –o option is used) Tool creates the log file (Os.log), which contains the list of error/warning/information messages in the output directory, if the command line arguments provided are correct. Otherwise, the log file is created in the directory where Os executable is present. "-" must be read as minus sign and not as hyphen. If command line options provided are correct and -L/-Log option is provided along with -H/-V option, Os.log (log file) is generated in the directory where Os.exe is present.

7.2 Sample Usage

Sample usage of the Os Generation Tool is shown below:

Os_Common
Os Generation Tool usage is displayed on the command line.

Os_Common -H
Display Os Generation Tool help information on the command line.

Os_Common -V
Os Generation Tool version and information is displayed on the command line.

Os_Common -V -H -O output Sample.arxml
Os Generation Tool help is displayed, since –H option has the highest priority.

Os_Common -V -O output Sample
Os Generation Tool version is displayed, since –V option has higher priority than –o option.

Os_Common –I –o output Sample.arxml
Os Generation Tool logs the output in the "Os.log" file. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in 'src' directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in 'inc' directory.

Os_Common –W Sample.arxml
To disable Warning Messages. The Tool will not generate warning messages on command line.

Os_Common –L –O output Os.arxml
Os Generation Tool accepts the input files from the current working directory, logs the output in the Os.log file. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in 'src' directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in 'inc' directory.

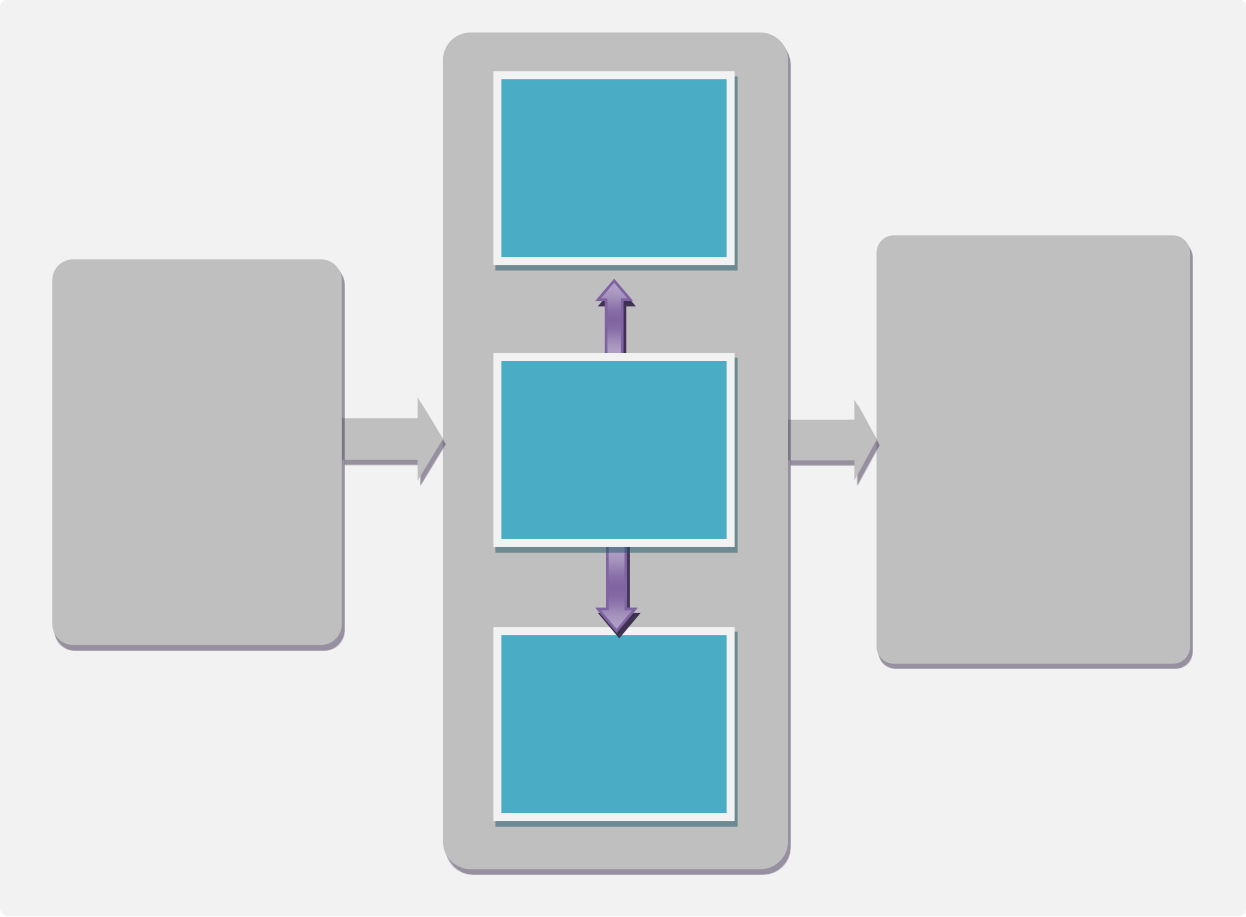
Os_Common -O output Os.arxml
Os Generation Tool accepts the input files from the current working directory and output files are generated in folder “output”. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in ‘src’ directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in ‘inc’ directory.

Os_Common Os.arxml
Os Generation Tool accepts the input files from the current working directory and output files are generated in the default folder “Os_Output”, since –O option is not provided on the command line. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in ‘src’ directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in ‘inc’ directory.

Os_Common C:₩path₩ Os.arxml
In the above command, Os Generation Tool accepts ECU Configuration Description Files from absolute path. Output files are generated in the default folder “Os_Output”, since –O option is not provided on the command line. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in ‘src’ directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in ‘inc’ directory.

Os_Common ₩path₩Os.arxml
In the above command, Os Generation Tool accepts ECU Configuration Description File from the relative path. Output files are generated in the default folder “Os_Output”, since –o option is not provided on the command line. Os_Cfg.c, Ioc.c, Os_IocCfg.c, Os_PCfg.c and Os_Vector.c files are generated in ‘src’ directory. Os_Cfg.h, Ioc.h, Os_IocCfg.h, Os_PCfg.h, Os_PortTypes.h, and Os_Types.h, files are generated in ‘inc’ directory.

7.3 AUTOSAR Os Generation Tool Overview



Os Generation Tool is a command line tool that provides scalability and configurability for Os Module. It accepts ECU Configuration Description File(s) and MDT as input and generates the C Source and Header files.

ECU Configuration Description File(s) can be created/edited using any ECU Configuration Editor and MDT is prepared by using SysConf Tool. The details of the configurable parameters present in the input file are provided in Parameter Configuration (Refer Respective Section 5.x.4).

The Os Generation Tool extracts analyses and validates the correctness of the configuration details provided in the input file(s). Tool displays appropriate context sensitive error messages for wrong input and exits. Tool creates the log file (Os.log), which contains the list of error/warning/information messages.

	Generation tool returns 1 when exits with errors and returns 0 when exits with no errors.

7.4 Tool installation requirements

The minimum hardware, software requirements for proper installation of Os Generation Tool is listed. This ensures optimal performance of the Tool.

7.4.1 Hardware requirements

Processor	Pentium/equivalent processor @ 500 Mega Hertz or greater
Memory	64MB RAM or greater
Hard Disk Drive	500MB or greater storage capacity

7.4.2 Software requirements

Operating System	Microsoft Windows Platform

7.4.3 Limitations

Command line characters are limited to 128 depending upon the operating system.

7.5 Tool installation

The installation procedure for Os Generation Tool is provided in the section below:

7.5.1 Pre Requisite

Os Generation Tool executable runs on Windows platforms only.

7.5.2 Installation Steps

The procedure to be carried out for the installation of the tool is as follows:

- Copy the Generation Tool executable (Os.exe) and Os.template file on to the local hard disk.
- Copy the generation tool common executable (Os_Common.exe) and Os_Common.template on the same path as Os.exe or in the path Os₩Common₩ssc₩generator
- Run the executable (Os_Common.exe) with -H option to get help on usage of the tool.

  Os_Common.exe –H
- This command generates Os Generation Tool '-H' on the command line.

7.6 Tool uninstallation

There is no specific method for uninstalling the Os Generation Tool. Delete the Generation Tool executable from the existing folder.

7.7 Input Files

The Os Generation Tool accepts ECU Configuration Description File(s) and BSWMDT file as input. Os Generation Tool parses information on Os module. Parameters of Os Module are explained in the Parameter Configuration (Refer: Section 5.xx.4 and 8.7.1 note: xx can be 1 to 21).

Generation Tool either accepts single or multiple ECU Configuration Description File(s). ECU Configuration Description File(s) can be generated using any Configuration Editor. The ECU Configuration Description file must comply with AUTOSAR standard Description File format.

Notation in Document	Definition
	Template file 'Os.template' and 'Os_Common.template' provided along with Generation Tool Executable should not be edited.

7.8 Precautions

- The ECU Configuration Description File must comply with AUTOSAR standard Description File format.
- ECU Configuration Description files should not be edited manually.
- Template file (Os.template and Os_Common.template) should not be edited.
- Template file should be available in the folder where the executable is present.
- If the Output file(s) generated by the Tool are modified externally, then they may not produce the expected results.
- For the parameters, which are referenced using References, full path to the container must be provided.

  Example: /<package_name>/<module_shortname>/<container_shortname>.
- Short Name for a container must be unique within a name space.
- The input file must contain Os component.
- All the string values configured must follow C syntax for variables. It can only contain alphanumeric characters and "_". It must start with an alphabet.
- An error free ECU Configuration Description File(s) generated from configuration editor has to be provided as input to the Os Generation Tool. Otherwise Tool may not produce the expected results or may lead to errors/warnings/informations.

7.9 User Configuration Validation

This section provides help to analyze the errors or warnings displayed during the execution of Os Generation Tool. It ensures conformance of input file with syntax and semantics. It also performs validation on the input file for correctness of the data.

For more details on list of Error/Warning/Information messages that are displayed as a result of input file(s) validation, refer Section "12 CONSISTENCY CHECKS".

The Generation Tool displays error or warning or information when the user has configured incorrect inputs. The format of Error/Warning/Information message is as shown below.

- ERR/WRN/INF<mid><xxx>: < Error/Warning/Information Message>
  Where,
  <mid>: 035 – CanTp Module Id (035) for user configuration checks.
  000 – for command line checks.
  <xxx>: 051 – 999 – Message ID.
- File Name : Name of the file in which the error has occurred
- Path : Absolute path of the container in which the parameter is present

'File Name' and 'Path' are optional.

8. Bswmd

8.1 BSW MDT PARAMETER CONFIGURATION

This section explains about the elements and valid values

Element Name	BSW-IMPLEMENTATION
SW-VERSION	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific. Example: 1.0.0
VENDOR-ID	This parameter specifies vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list. Example: 76
AR-RELEASE-VERSION	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR. Example: 4.0.3
BEHAVIOR-REF	This parameter contains reference to a corresponding BSW-INTERNAL-BEHAVIOR. Example: /ArPackage_0/ModuleDescription_0/BswInternalBehavior_0
VENDOR-API-INFIX	In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: 〈ModuleName〉_〈vendorId〉_〈vendorApiInfix〉_〈API name from SWS〉. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity 〉 1. It shall not be used for modules with upper multiplicity =1.

Element Name	BSW-MODULE-DESCRIPTION
MODULE-ID	This parameter specifies Module ID of this Module from AUTOSAR Module List. Example: “1” for Os

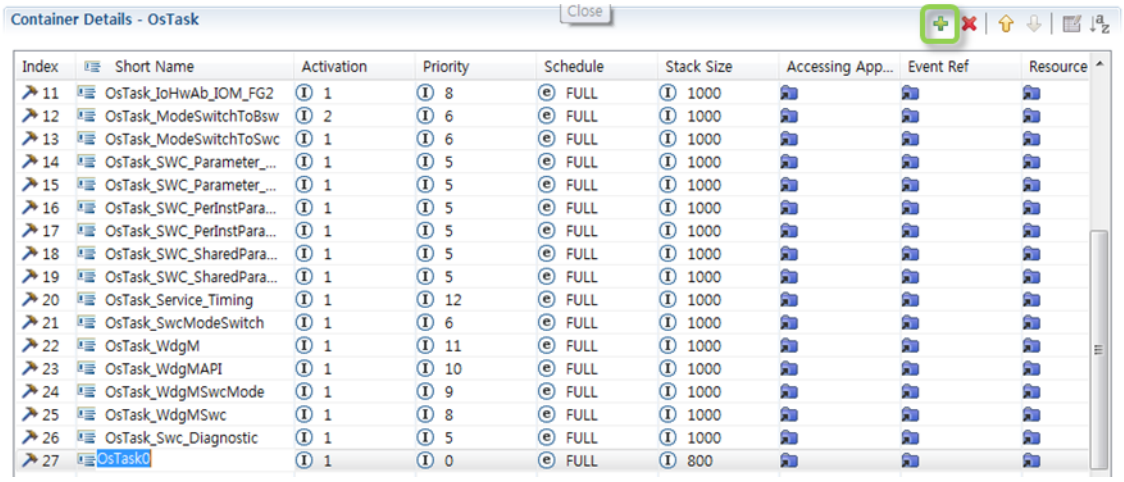
9. Exclusive Areas

N/A

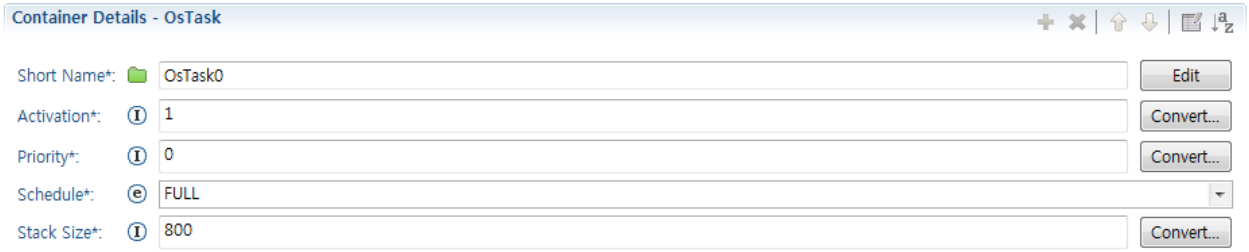
10. APPENDIX

10.1 Task configuration

1. Create a Task in the Task configuration window.



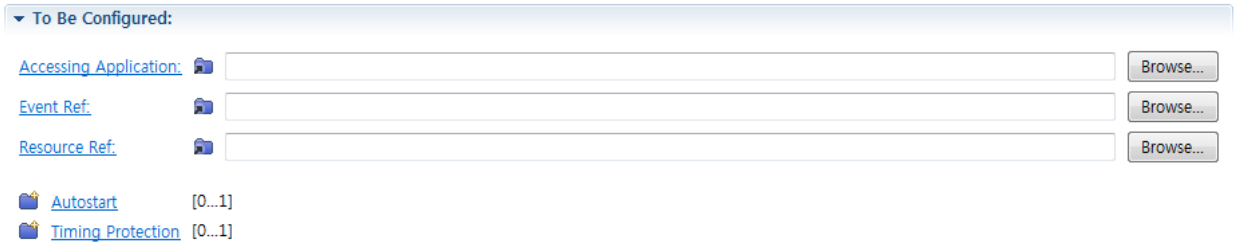
2. Configure Task's properties.



✓ 1) Bigger numbers of this parameter refer to higher priority.

Configuration Item	Description	M/O
Short Name	Task Name	M
Activation	This parameter specifies the maximum number of queued activation requests for the task	M
Prioirty [1)]	The priority of a task is defined by the value of this attribute	M
Schedule	This parameter defines the preemptability of the task FULL: Task is preemptable NON: Task is not preemptable	M
Stack Size [2)]	Stack size in extended task	M
		M: Mandatory O: Optional X: Not Supported

✓ 2) For basic task, this prarameter is not used.



✓ 1) If OS-Application is used and an other OS-Application (do not includes this Task) needs to access this Task object , this parameter should be configured.

Configuration Item	Description	M/O
Accessing Application [1)]	This parameter references the Application which has the access to this Task	O
Event Ref [2)]	This parameter defines the list of Events the Extended Task may react on	O
Resource Ref [3)]	This parameter defines the list of Resources accessed by this Task	O
Autostart [4)]	This container determines whether the Task is activated during the system start-up procedure or not for some specific application modes	O
Timing Protection [5)]	This container contains all parameters regarding timing protection of the Task	O
		M: Mandatory O: Optional X: Not Supported

✓ 2) If this Task waits for an event, this parameter should be configured.

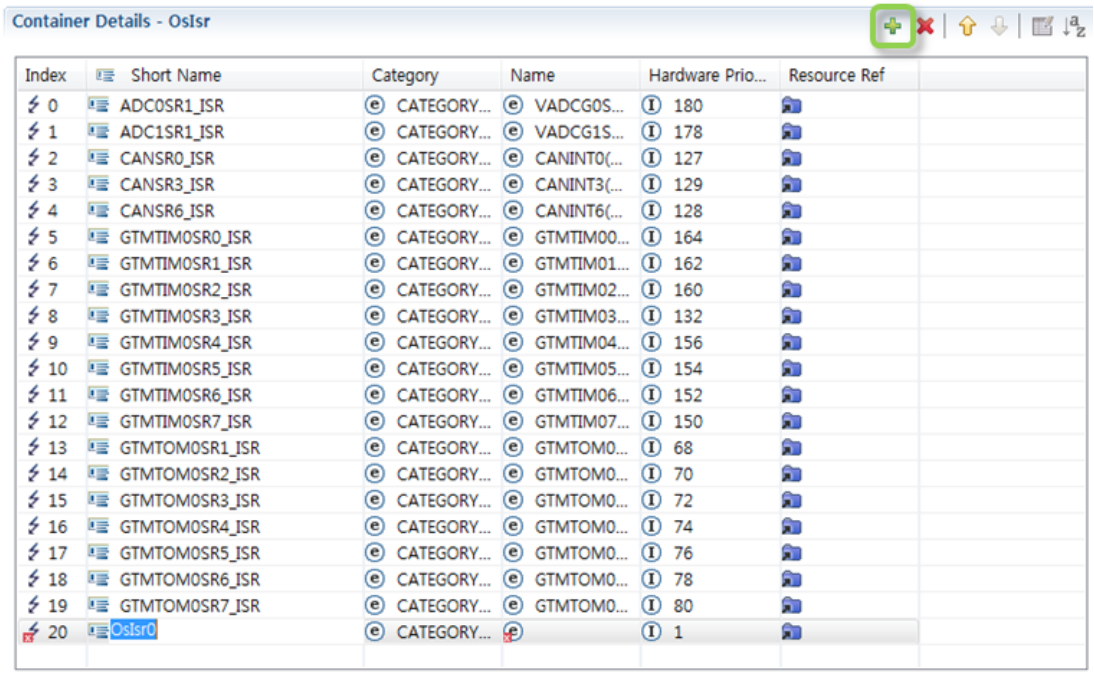
✓ 3) If this Task uses a resource, this parameter should be configured.

✓ 4) If this Task should be started automatically, this container should be configured.

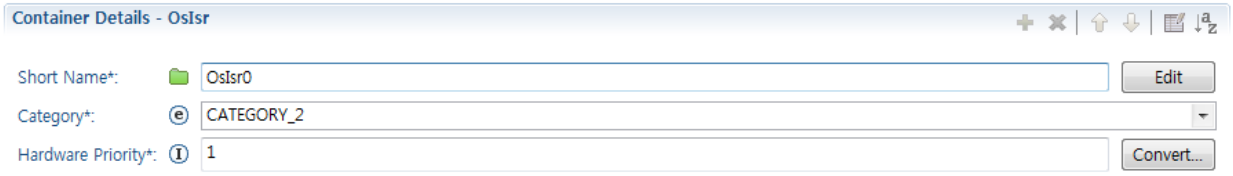
✓ 5) If this Task uses a Timing Protection, this container should be configured

10.2 ISR configuration

1. Create an ISR in the ISR configuration window.

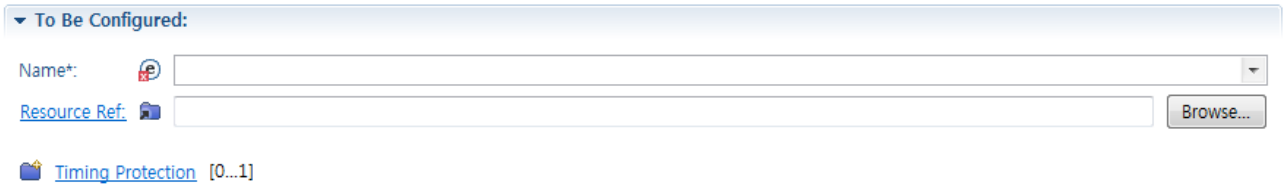


2. Configure ISR's properties.



Configuration Item	Description	M/O
Short Name	ISR Name	M
Category [1)]	This parameter specifies the category of this ISR CATEGORY_1: Interrupt is of category 1 CATEGORY_2: Interrupt is of category 2	M
Hardware Prioirty [2)]	ISR's hardware priority	M
		M: Mandatory O: Optional X: Not Supported

- ✓ 1) Category 1 Interrupts should have higher prioirty (lower number) than Category 2 Interrupts. If not, OS generator will print a generation error.
- ✓ 2) A range of Hardware priority relies on Micro Controller Unit. For example, In Infineon Aurix, this range is from 0 to 255.

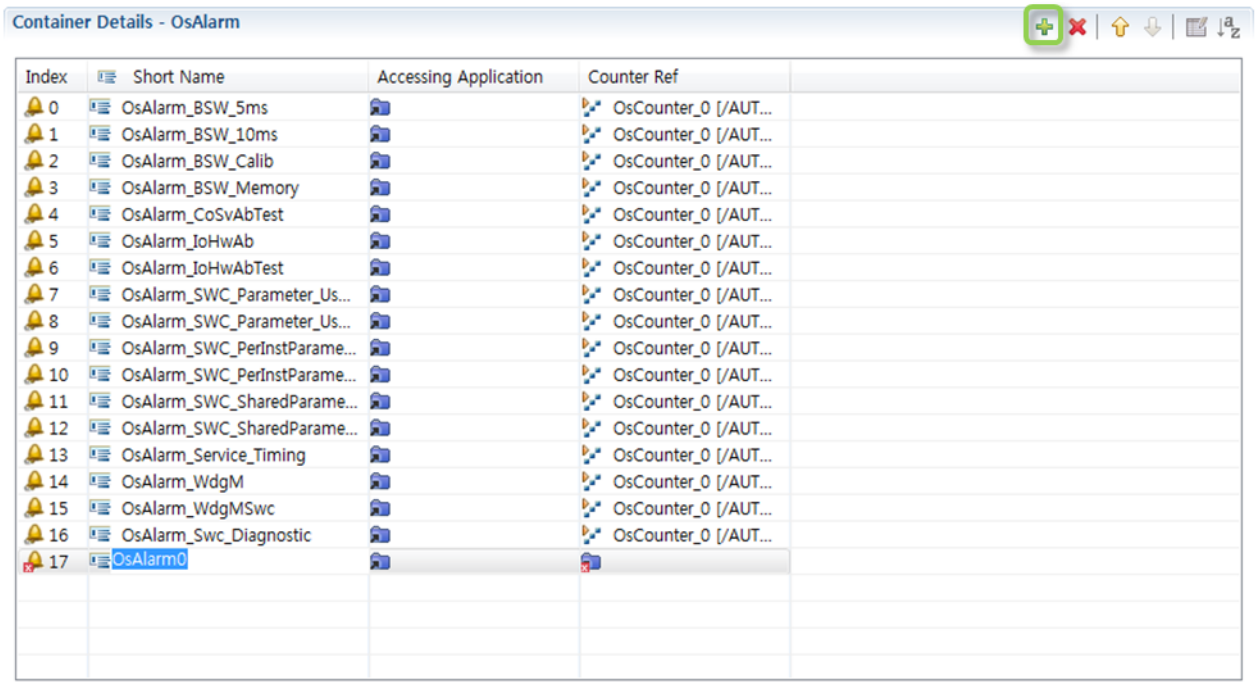


Configuration Item	Description	M/O
Name	Interrupt source name (ex. CAN1INT0 – Infineon Aurix)	M
Resource Ref [1)]	This parameter defines the list of Resources accessed by this Task	O
Timing Protection [2)]	This container contains all parameters regarding timing protection of the ISR	O
		M: Mandatory O: Optional X: Not Supported

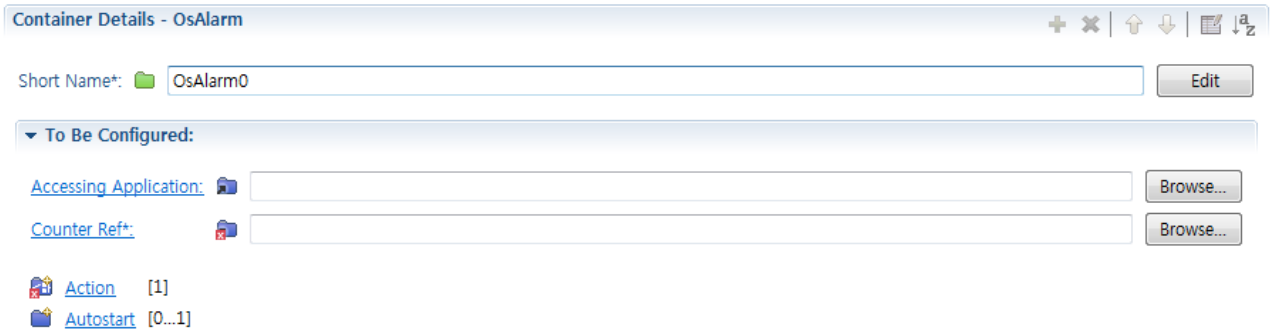
- ✓ 1) If this Task uses a resource, this parameter should be configured.
- ✓ 2) If this Task uses a Timing Protection, this container should be configured
- ✓

10.3 Alarm configuration

1. Create an Alarm in the Alarm configuration window.



2. Configure Alarm's properties.

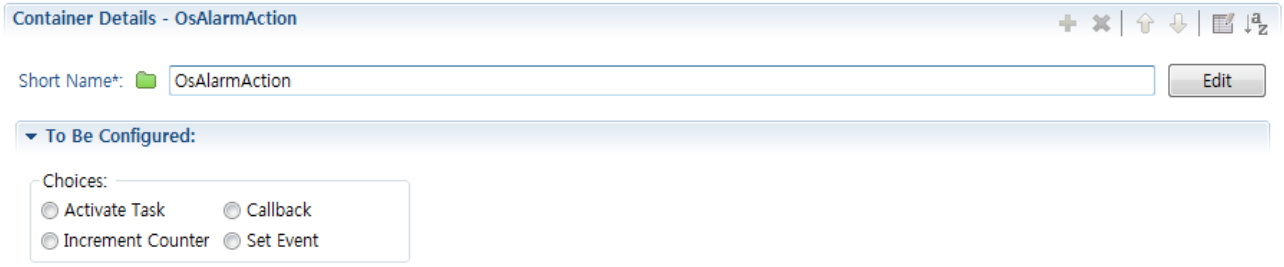


Configuration Item	Description	M/O
Short Name	Alarm Name	M
Accessing Application [1)]	This parameter specifies the reference to applications which have an access to Alarm object	O
Counter Ref	This parameter specifies the reference to the assigned counter for that alarm	M
Action	Alarm Action	M
Autostart	This container determines whether the Alarm is activated during the system start-up procedure or not for some specific application modes	O
		M: Mandatory O: Optional X: Not Supported

✓ 1) If OS-Application is used and an other OS-Application (do not includes this Task) needs to access this Task object , this parameter should be configured.

✓ 2) If this Alarm should be started automatically, this container should be configured.

3. Configure 'Action' container.

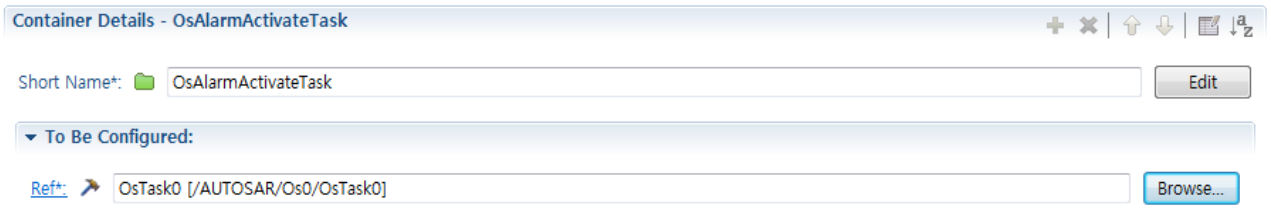


There are four types of alarm actions. Please refer to below table.

Alarm Action	Description
Activate Task	Activate a Task
Callback	Call a Alarm Callback function
Increment Counter	Increament Counter
Set Event	Set an Event

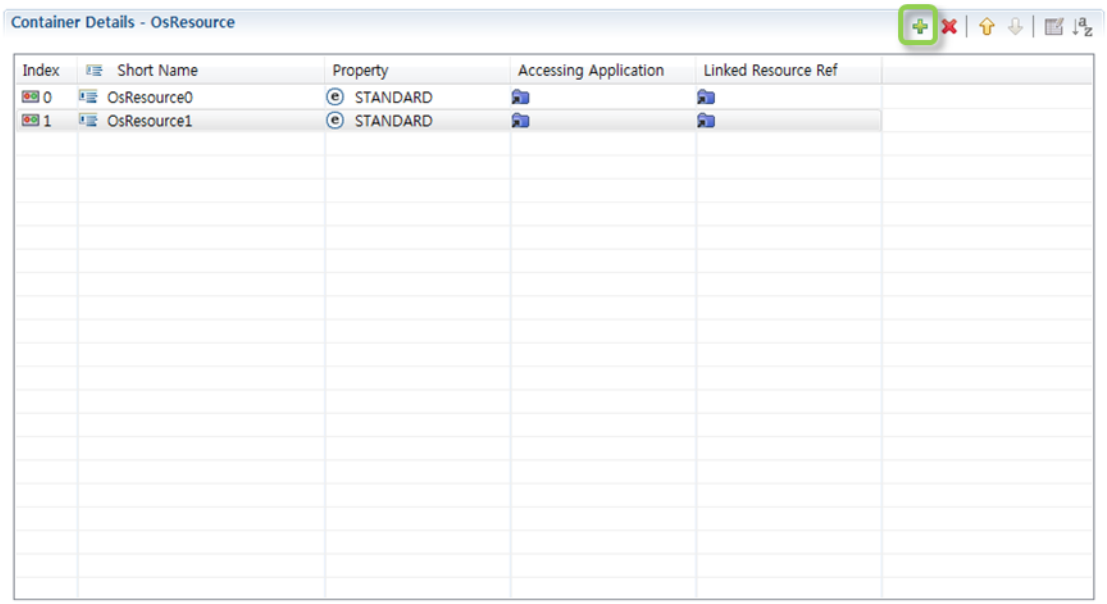
✓ Whenever Alarm is expired, this Alarm 'Action' is executed.

For example, If you select 'Activate Task' as an Alarm Action, a Task which is activated by alarm should be configured.

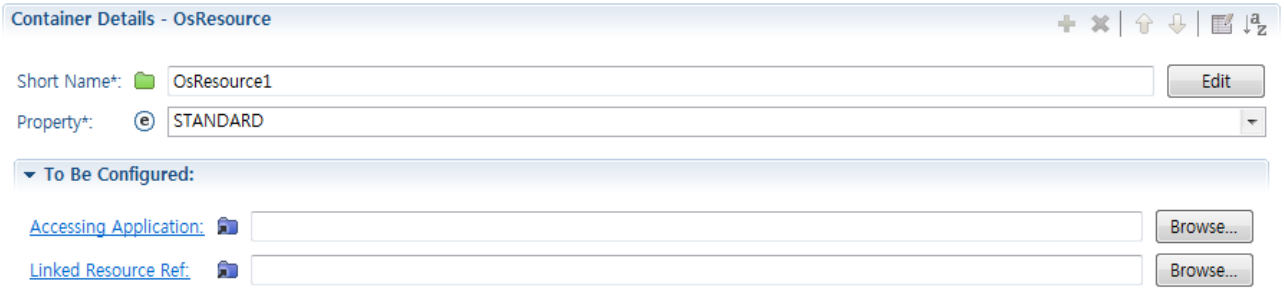


10.4 Resource configuration

1. Create a Resource in the Resource configuration window.



2. Configure Resource's properties.

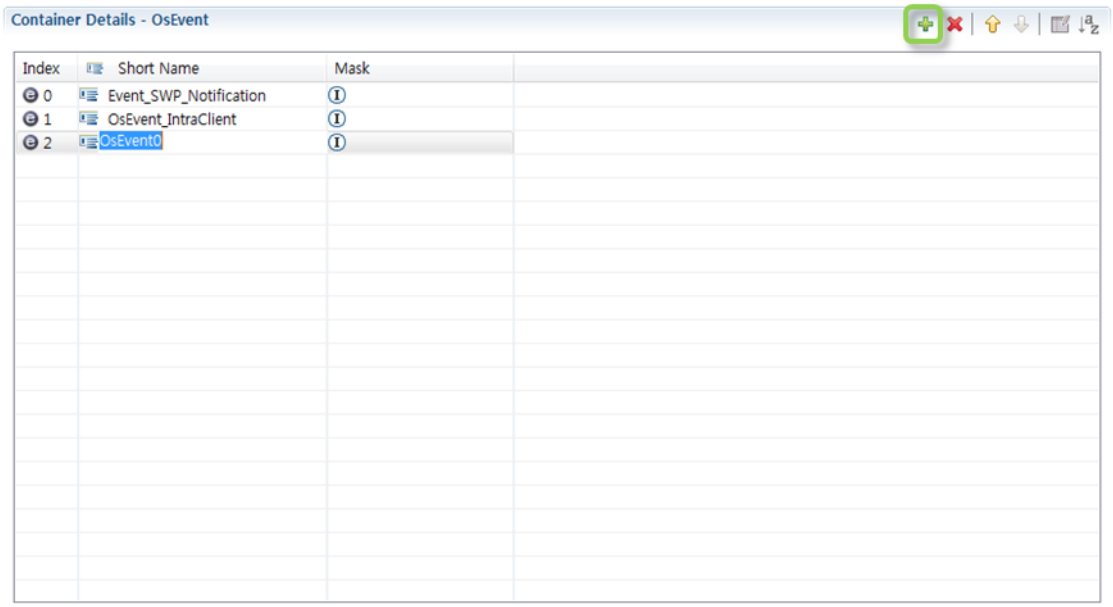


Configuration Item	Description	M/O
Short Name	Resource Name	M
Property	This parameter specifies the type of the resource. INTERNAL: The resource is an internal resource. LINKED: The resource is a linked resource (a second name for a existing resource). STANDARD: The resource is an standard resource.	M
Accessing Application [1)]	This parameter references to the applications which have an access to this Resource.	O
Linked Resource Ref [2)]	This parameter references to the linked resource. Configuration of this parameter is valid only if the Resource property is configured as LINKED.	X
		M: Mandatory O: Optional X: Not Supported

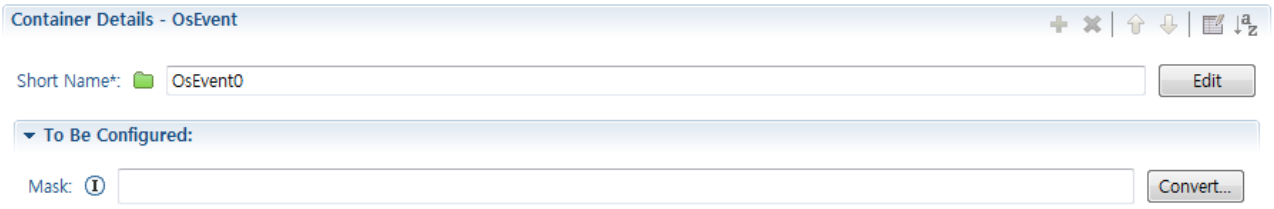
- ✓ 1) If OS-Application is used and an other OS-Application (do not includes this Task) needs to access this Task object , this parameter should be configured.
- ✓ 2) AutoEver OS doesn't support 'Linked Resource'.

10.5 Event configuration

1. Create a Event in the Event configuration window.



2. Configure Resource's properties.

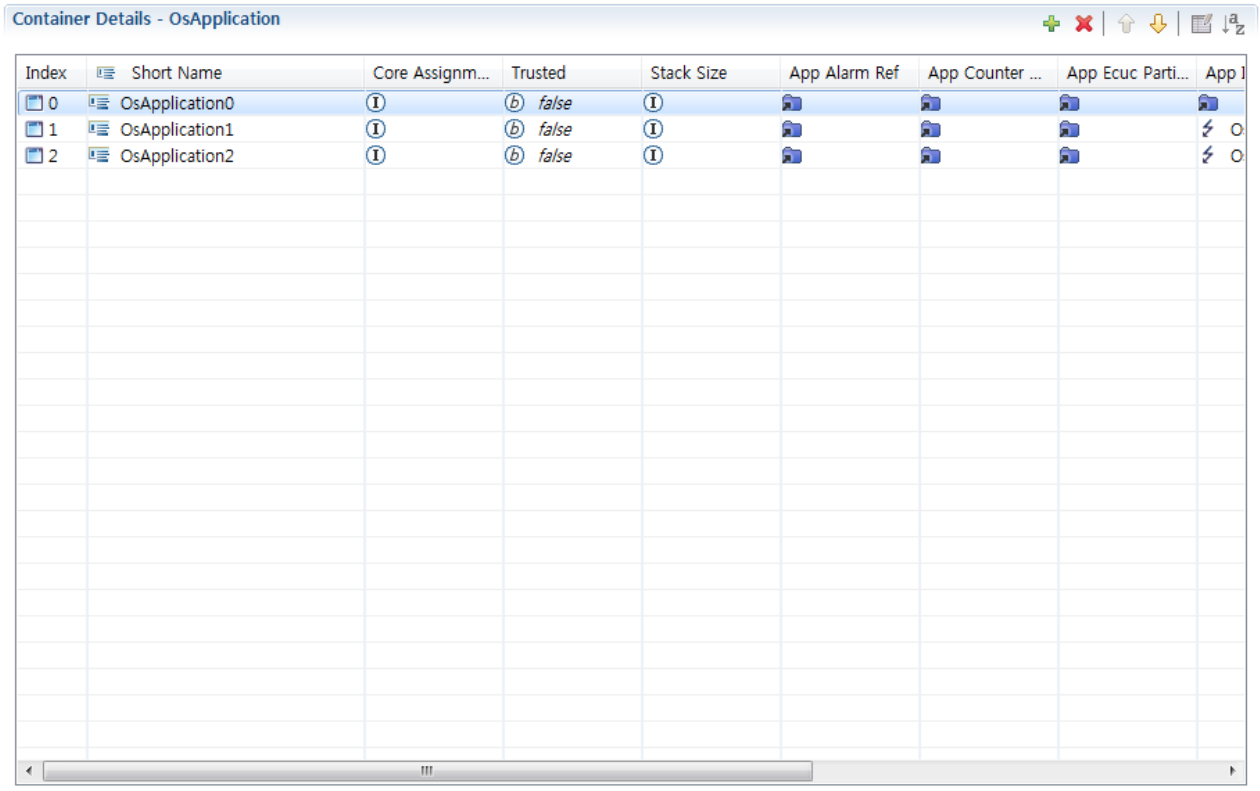


Configuration Item	Description	M/O
Short Name	Event Name	M
Mask [1)]	Integer value to be used as event mask	M
		M: Mandatory O: Optional X: Not Supported

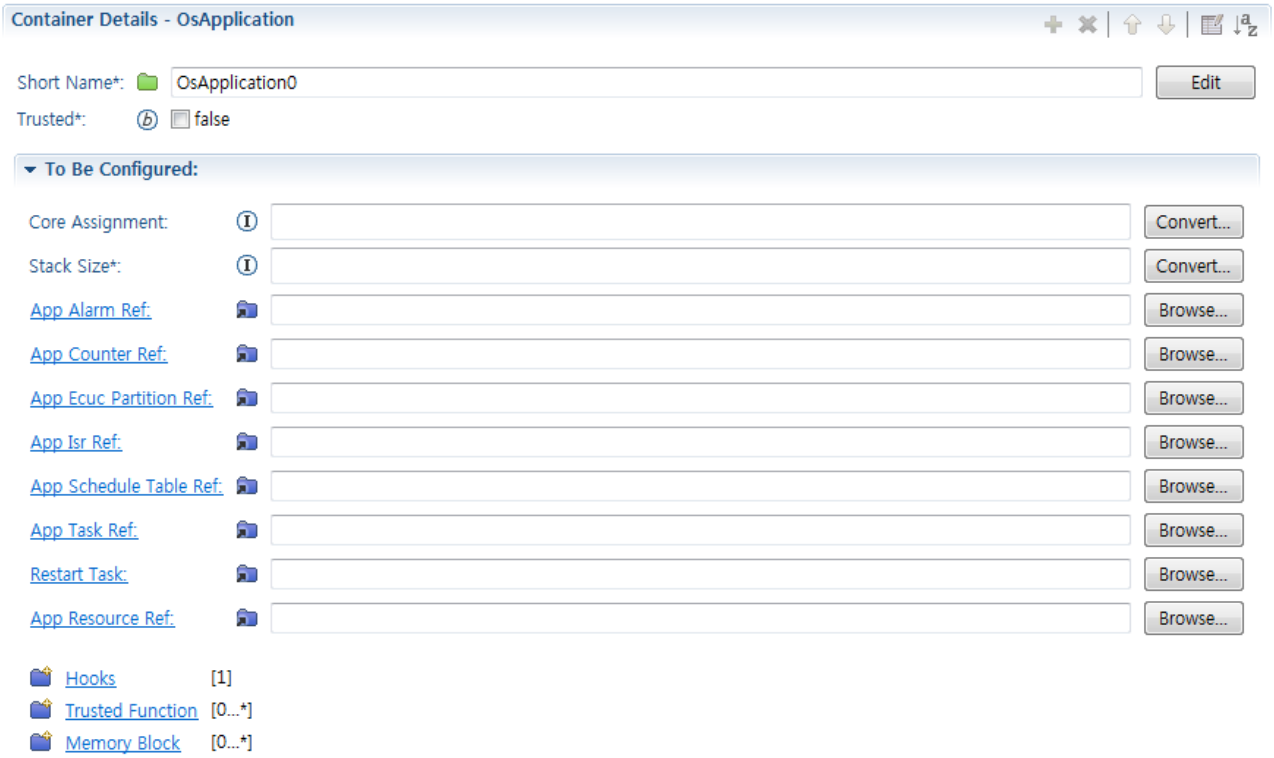
✓ 1) If Mask propertiy value remains empty, OS generator makes it automatically.

10.6 OS-Application configuration

1. Create an OS-Application in the OS-Application configuration window.



2. Configure OS-Application's properties.

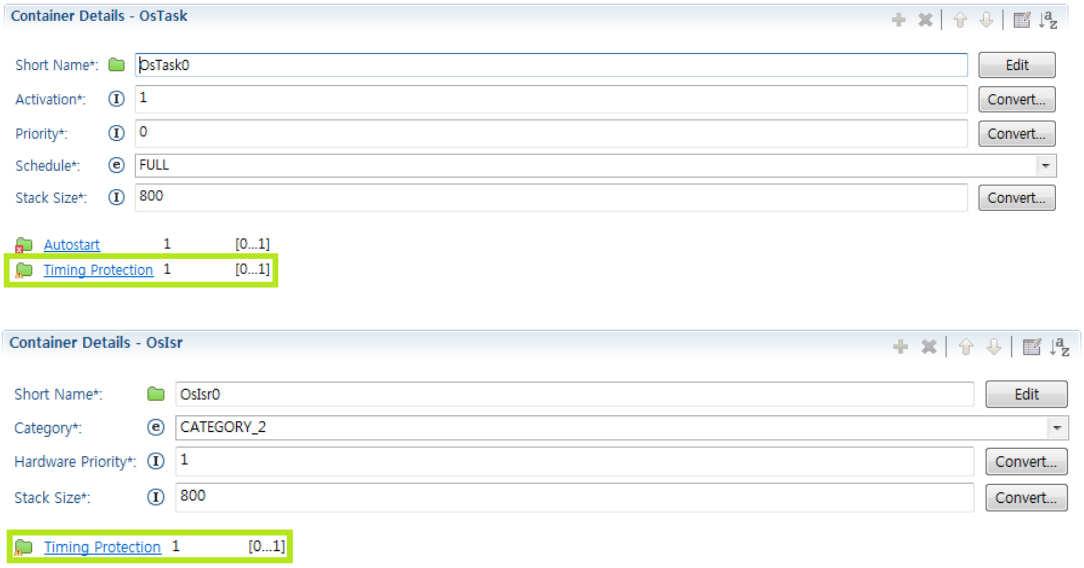


Configuration Item	Description	M/O
Short Name	OS-Application Name	M
Trusted	This parameter specifies that an OS-Application is trusted or not. True: OS-Application is trusted. False: OS-Application is not trusted.	M
Stack Size	This parameter specifies the stack size in an OS-Application. Basic Task and Category2 ISR in this OS Application share this stack area.	
Core Assignment [1)]	This parameter specifies an ID of the core onto which the OsApplication is bound.	O
App Alarm Ref [2)]	This parameter specifies the reference to the Os Alarms that belong to the OsApplication.	O
App Counter Ref [2)]	This parameter specifies the reference to the Os Counters that belong to the OsApplication.	O
App Ecuc Partition Ref	Denotes which "EcucPartition" is implemented by this "OSApplication".	O
App Isr Ref [2)]	This parameter specifies the reference to the Os Isrs belong to the OsApplication.	O
App Schedule Table Ref [2)]	This parameter specifies the reference to the Os Schedule Tables that belong to the OsApplication.	O
App Task Ref [2)]	This parameter specifies the reference to the Os Tasks that belong to the OsApplication.	O
Restart Task	Optionally one task of an OS-Application may be defined as Restart Task.	O
App Resource Ref [2)]	This parameter specifies the reference to the Os Resources that belong to the OsApplication.	O
Hooks	This container defines the Os Application specific hooks.	O
Trusted Function	This parameter specifies the Trusted function (as part of a trusted Os Application) available to other OS-Applications.	O
Memory Block	Container with details about memory area accessible to the Application	O
		M: Mandatory O: Optional X: Not Supported

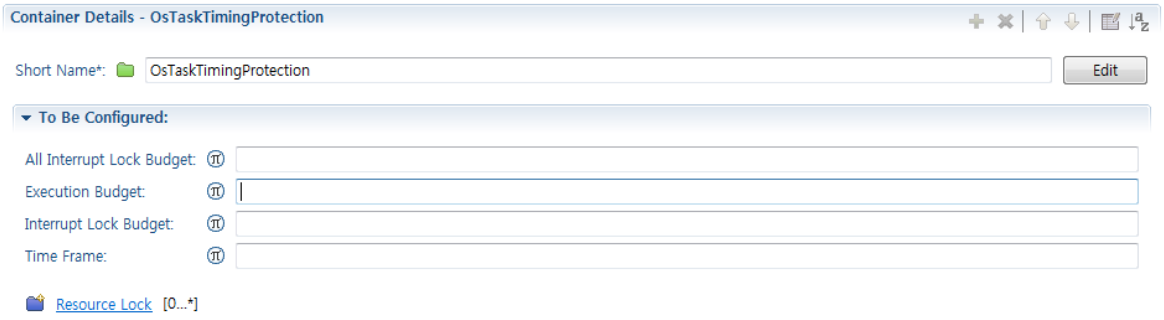
- ✓ 1) If Multi-Core is used, this parameter should be set.
- ✓ 2) If the OS-Application contains OS object - Task, Alarm, Counter, ScheduleTable, Resource .., this OS object should be referred in this OS-Application.

10.7 Timing-Protection configuration

1. Select 'Timing Protection' in a Task/ISR configuration window.



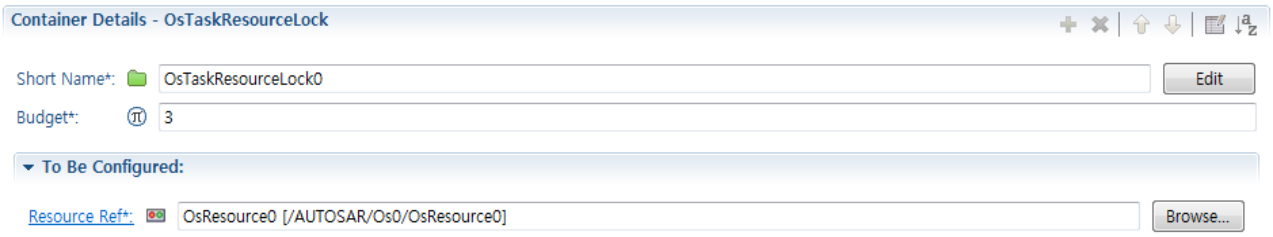
2. Select 'Timing Protection' in a Task/ISR configuration window.



Configuration Item	Description	M/O
Short Name	TimeProtection Name	M
All Interrupt Lock Budget	This parameter specifies the maximum time for which the task is allowed to lock all interrupts (via SuspendAllInterrupts () or DisableAllInterrupts ()) (in seconds).	O
Execution Budget	This parameter specifies the maximum allowed execution time of the task (in seconds).	O
Interrupt Lock Budget	This parameter specifies the maximum time for which the task is allowed to lock all Category 2 interrupts (via SuspendOSInterrupts()) (in seconds).	O
Time Frame	This parameter specifies the minimum inter-arrival time between activations and/or releases of a task (in seconds).	O
Resource Lock [1)]	This container contains the worst case time between getting and releasing a given resource (in seconds).	O
		M: Mandatory O: Optional X: Not Supported

✓ 1) For Resource Lock, more configuration is necessary like below.

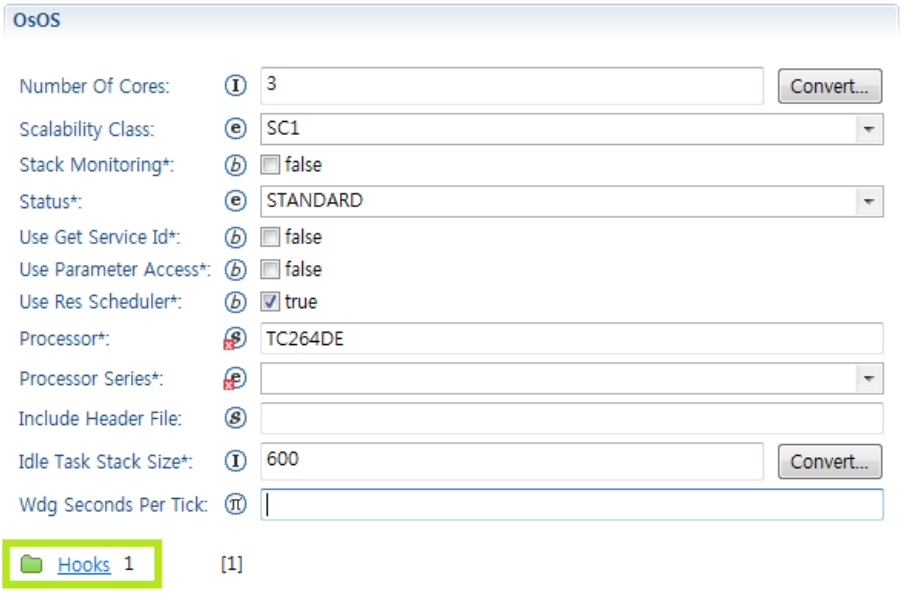
- Short Name : OsTaskResourceLock/OsIsrResourceLock Name
- Budget : This parameter specifies the maximum time the task/ISR is allowed to lock the resource (in seconds).
- Resource Ref : This parameter references the resource used by the Task/ISR.



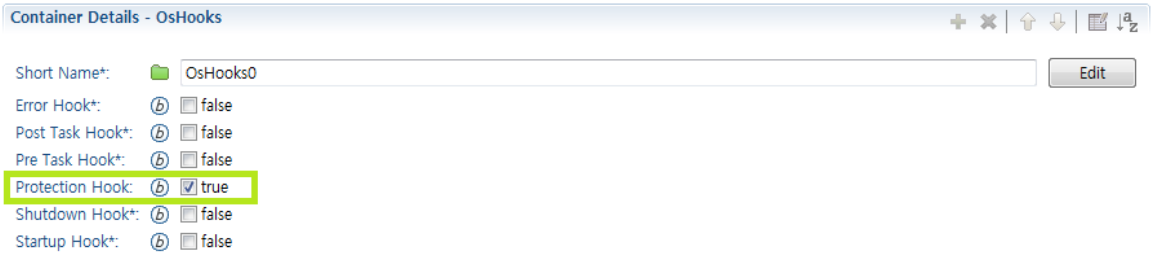
Note: If Resource Lock is configured, “OsOS-OsUseResScheduler” should set as “False”. For more detailed information, please refered to 3) in section 5.15.

10.8 ProtectionHook configuration

1. Select ‘Hooks’container in a OS configuration window.



2. Set 'Protection Hooks' attribute to true in a OsHooks configuration.



3. Write 'ProtectionHook' routine.

   - ProtectionReturnType ProtectionHook (StatusType FatalError)

   ProtectionHook has 5 types of return value and by reffering to this return value, OS decides next action.

Return value	Description
PRO_IGNORE	the system ignores the error and continuos operations as if no error happened at all.
PRO_TERMINATETASKISR	the system terminates a task or ISR related to the error.
PRO_TERMINATEAPPL	the system terminates an OS-Application related to the error.
PRO_TERMINATEAPPL_RESTART	the system restarts an OS-Application related to the error.
PRO_SHUTDOWN	The system executes ShutdownOS

10.9 NMI callback function

If NMI is enabled in the system, user can use below NMI callback function to handle additional processing during NMI exception.

- Os_CallBackNMInterrupt

Function Name	Os_CallBackNMInterrupt
Syntax	void Os_CallBackNMInterrupt(void)
Service ID	N/A
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None

10.10 Stack configuration

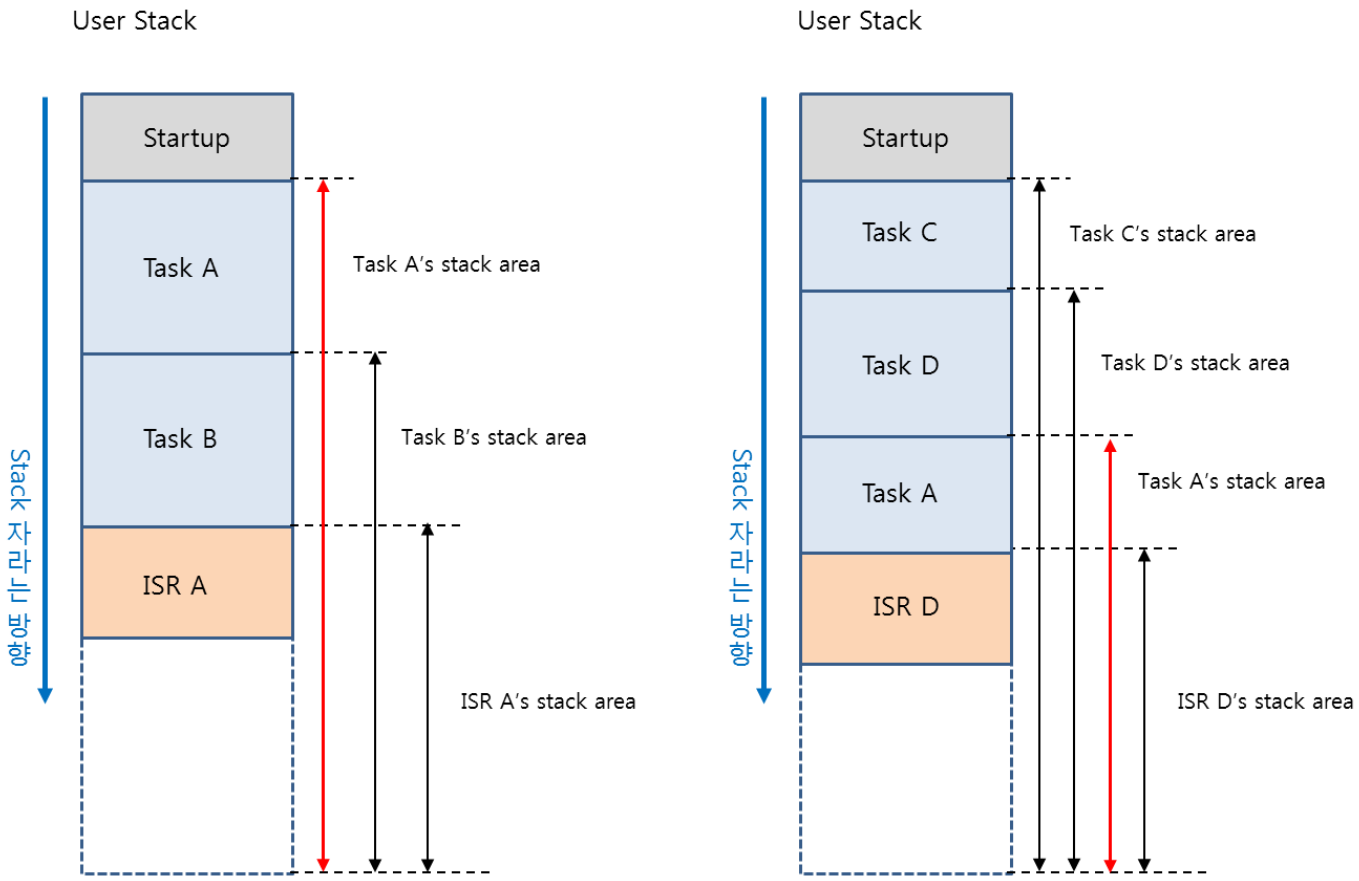
CYTxxx architecture uses two kinds of stack like below.

- User stack
- Kernel stack

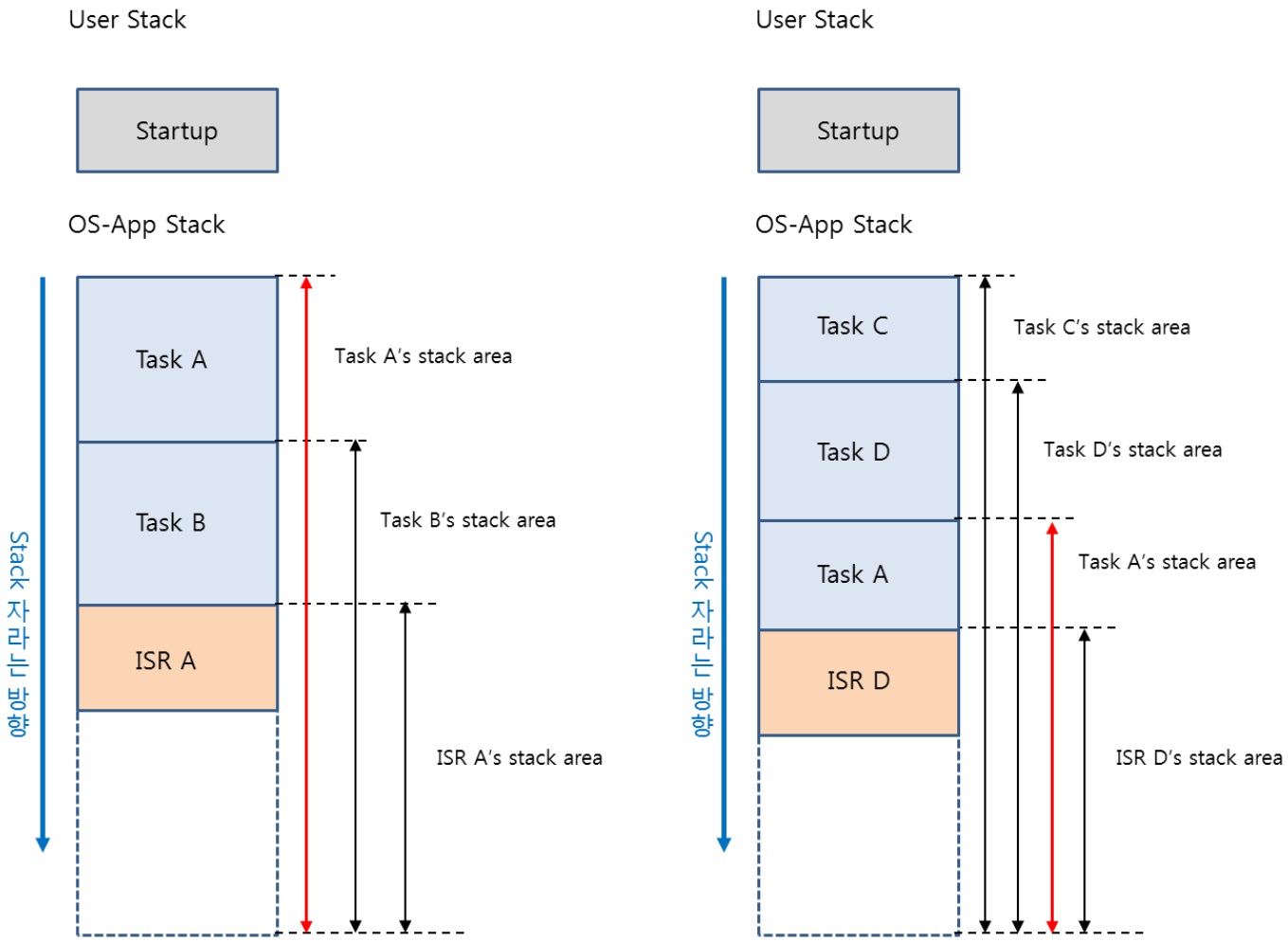
AutoEver AUTOSAR OS based on CYTxxx architecture uses two stacks additionally.

- Private task stack
- OS-Application stack

In AutoEver AUTOSAR OS, basic task and category2 ISR share User stack or OS-Application stack.



〈 Stack usage without OS-Application 〉



〈 Stack usage with OS-Application 〉

Below table shows stack usage according to OS configuration (Scalability, Multicore).

Scalability Class	Multicore	Basic Task	Extended Task	Cat2 ISR	Cat1 ISR (+ IOC ISR)
SC1	X	User stack [1]	Private task stack [2]	User stack	Kernel stack [3]
	O	OS-App stack [4]	Private task stack	OS-App stack	Kernel stack
SC2	X	User stack	Private task stack	User stack	Kernel stack
	O	OS-App stack	Private task stack	OS-App stack	Kernel stack
SC3	X	OS-App stack	Private task stack	OS-App stack	Kernel stack
	O	OS-App stack	Private task stack	OS-App stack	Kernel stack
SC4	X	OS-App stack	Private task stack	OS-App stack	Kernel stack
	O	OS-App stack	Private task stack	OS-App stack	Kernel stack

✓ 1) User stack is configured by Os configuration.

Please refer to OsUserStackSize in chapter 5.1.1 if user stack size is not sufficient.

✓ 2) Private task stack is configured by task configuration.

Please refer to OsStackSize in chapter 5.2.1 if private stack size is not sufficient.

✓ 3) Interrupt stack is configured by linker script file (.lsl).

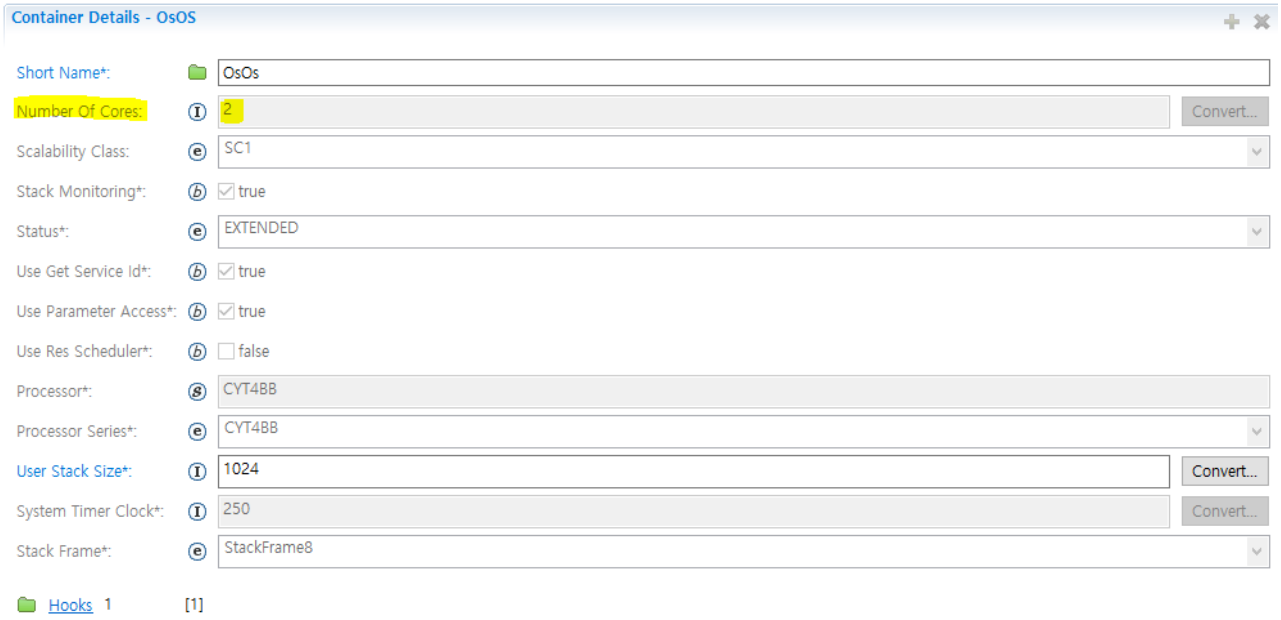
Please change ISTACK size in linker script file if interrupt stack size is not sufficient.

✓ 4) OS-Application stack is configured by OS-Application configuration.

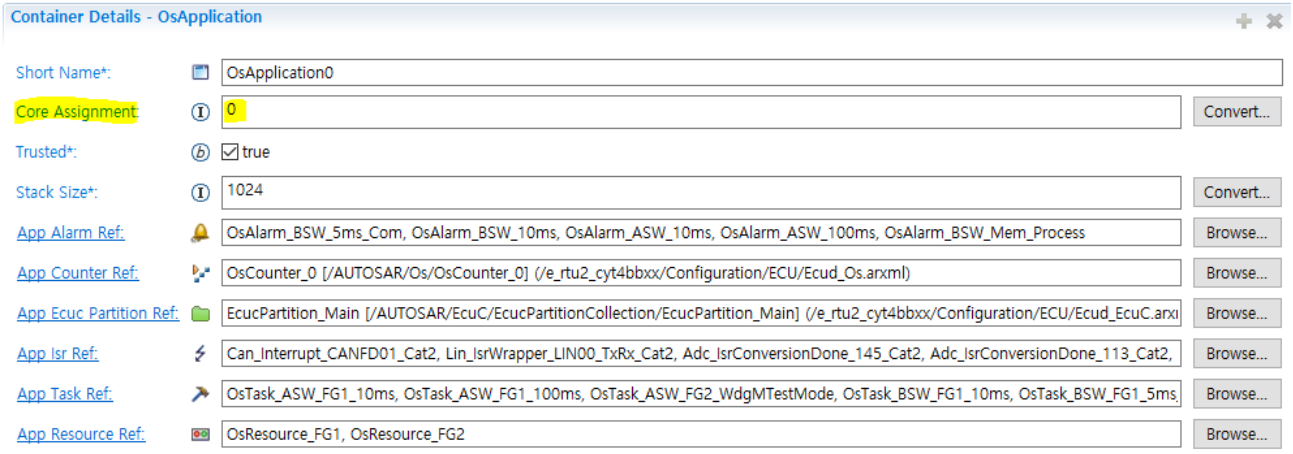
Please refer to OsApplicationStackSize in chapter 5.10.1 if os-application stack size is not sufficient

10.11 Multi-Core configuration

1. Configure number of cores.

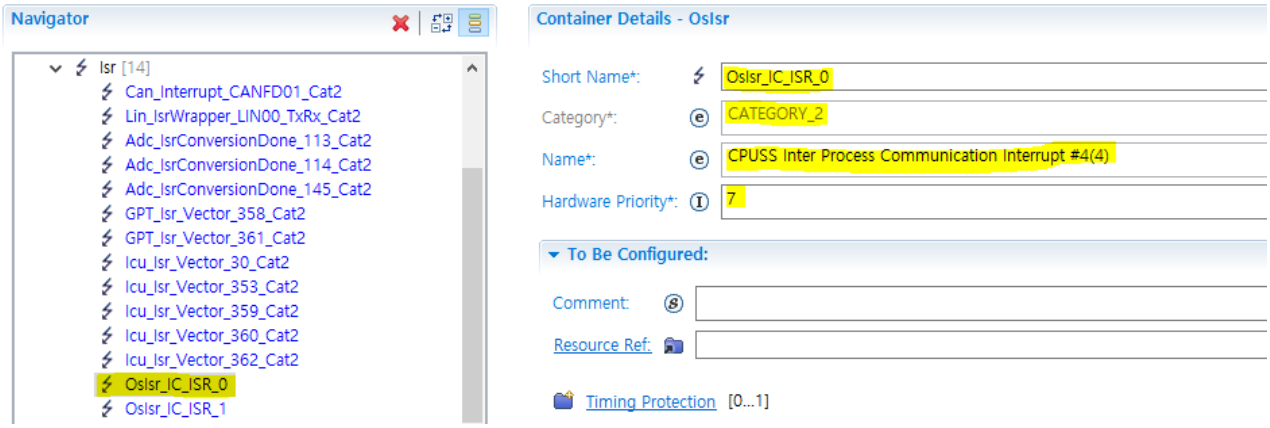


2. Assign an OS-Applicaion to a core.



3. Configure each Isr for inter core communication between cores

In order to communicate between two cores, two ISR (OsIsr_IC_ISR_0, OsIsr_IC_ISR_1) should be configured. Also, OsIsr_IC_ISR_0 should be refered in os application of core 0(M7_0) and OsIsr_IC_ISR_1 should be refed in os application of core 1(M7_1). Each ISR should be configured like below:

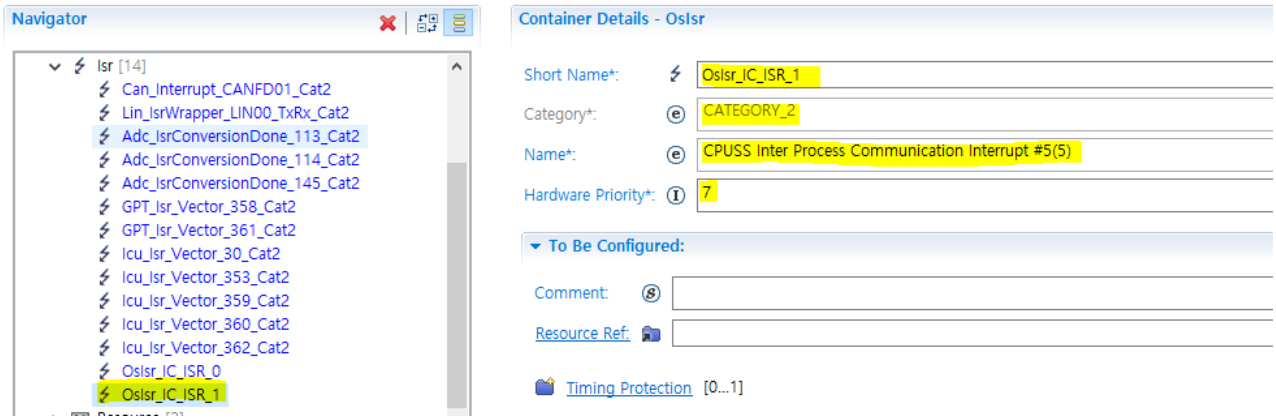


15) Short Name : OsIsr_IC_ISR_0

16) Category : CATEGORY_2

17) Name : CPUSS Inter Process Communication interrupt #4(4)

18) Hardware Priority : 7 (The lowest priority or user configuration)



19) Short Name : OsIsr_IC_ISR_1

20) Category : CATEGORY_2

21) Name : CPUSS Inter Process Communication interrupt #5(5)

22) Hardware Priority : 7 (The lowest priority or user configuration)