Freescale AUTOSAR OS/MPC56xxAM v.3.0

Technical Reference

Revised: 16 March 2010

Legal Notices

Important Notice to Users

Contents

1

Introduction

This Technical Reference describes Freescale AUTOSAR OS / MPC56xxAM, the implementation of the AUTOSAR[1] Operating System (AUTOSAR OS) for Power Architecture providing high speed performance and low RAM usage. The AUTOSAR OS specification based on the OSEK OS specification with addition of more protection facilities and ability to integrate into overall AUTOSAR framework. The Freescale AUTOSAR OS / MPC56xxAM is intended to be used inside AUTOSAR framework, however it can be used without it to build the applications in the traditional OSEK manner.

"Operating System Architecture" chapter gives a high level description of the OS architecture and presents OS Conformance Classes.

"Task Management" chapter explains the task concept in AUTOSAR and all other questions related to tasks.

"Scheduler" chapter provides a description of scheduling policies in AUTOSAR OS.

"Interrupt Processing" chapter highlights the AUTOSAR approach to interrupt handling.

"Resource Management" chapter describes resource management and task coordination by resources.

"Counters and Alarms" chapter describes usage of these control mechanisms in AUTOSAR OS.

"Events" chapter is devoted to event management and task coordination by events.

1. The term AUTOSAR means 'Automotive Open System Achitecture".

"Communication" chapter describes a message concept in OSEK and its usage.

"Error Handling and Special Routines" chapter describes the support provided for the user to debug an application and handle errors.

"System Configuration" chapter describes possible AUTOSAR OS versions, configuration options and the configuration mechanism.

"System Objects Definition" chapter describes the objects controlled by the Operating System: tasks, resources, alarms, messages, counters, ISRs and even the OS itself are considered as system objects.

"Building of Application" chapter contains information on how to build a user's application using the AUTOSAR OS. It also describes memory requirements.

"Power Architecture Platform-Specific Features" chapter describes special AUTOSAR OS features for different MCU types and issues connected with porting applications to these MCUs.

"Application Troubleshooting" chapter contains useful information for debugging applications developed using the AUTOSAR OS.

"System Services" chapter provides a detailed description for all AUTOSAR Operating System run-time services, with appropriate examples.

"Debugging Application" chapter provides information about preparation of all data required for the OSEK aware debugger to display information about an application in AUTOSAR terms.

"Sample Application" appendix contains the text and listing of a sample customer application developed using the AUTOSAR OS.

"System Service Timing" appendix provides information about OS services execution time

"Memory Requirements" appendix provides information about the amount of ROM and RAM directly used by AUTOSAR OS.

"System Generation Error Messages" appendix explains AUTOSAR OS System Generator error messages.

The "Introduction" chapter consists of the following sections:

AUTOSAR OS Overview

The AUTOSAR Operating System is a real-time operating system which conforms to the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification.

The AUTOSAR OS meets the following requirements:

- The OS is fully configured and statically scaled;
- The OS is intended to be used as integrated part of AUTOSAR;
- The OS provides memory, timing and other protection facilities in accordance with Scalabilty Classes (SC1..SC4);
- The OS performance parameters are well known;
- The OS is written in correspondence with MISRA C Guidelines, all deviations from MISRA are documented.

A wide range of scalability, a set of system services, various scheduling mechanisms, and convenient configuration features make the AUTOSAR Operating System feasible for a broad spectrum of applications and hardware platforms.

The AUTOSAR OS provides a pool of different services and processing mechanisms for task management and synchronization, data exchange, resource management, and interrupt handling. The following features are provided for the user:

Task Management

- Activation and termination of tasks;
- Management of task states, task switch.

Scheduling Policies

- Full-, non-, and mixed-preemptive scheduling techniques.

Event Control

- Event Control for task synchronization.

Interrupt Management

- Services for disabling / enabling all interrupts;
- Services for disabling / enabling interrupts of category 2;

Resource Management

- Control of mutually exclusive access to jointly used resources or devices, or for control of a program flow.

Communication[1]

- Data exchange between tasks and / or ISRs.

Counter and Alarm Management

- The counter management provides services for execution of recurring events;
- The alarm management is based on the counter management. The alarm management allows the user to perform link task activation or event setting to a certain counter value. These alarms can be defined as either single (one-shoot) or cyclic alarms. Expiration of a preset relative counter value, or the fact that a preset absolute counter value is reached, results in activation of a task, or setting a task event;
- ScheduleTable enables periodic activations of tasks in accordance with a static defined schedule. Freescale AUTOSAR OS / MPC56xxAM supports Global Time Synchronization in all Scalability Classes.

Error Treatment

- Mechanisms supporting the user in case of various errors, such as calling OS services with wrong parmeters and in the wrong context.

[1] The Communication part of this implementaition of AUTOSAR Operating System conforms to the *OSEK/VDX Communication, v.3.0.3, 20 July 2004* specification.

Memory Protection

- Protects the OS and OS-Applications Tasks and ISRs data and code from inadvertent writes by non-Trusted OS-Application code.

Timing Protection

- Protects the OS from the Tasks and ISRs deadline violations and from locking Resources and Interrupts for too long time.

ORTI Subsystem

- The ORTI provides an interface to Operating System run-time data for “OSEK aware” debuggers.

The AUTOSAR Operating System is scaled in two ways: either by changing the set of system services or through the so-called Conformance and Scalability Classes. They are available to meet different requirements concerning the OS functionality and capability. The Conformance Classes differ in the number of services they provide and OS capabilities. The Scalability Classes differ in the protection mechanisms provided by the OS (see “Scalability Classes”). The classes are based on one another in upwardly compatible fashion. (see “Conformance Classes”)

The AUTOSAR OS is built according to the user’s configuration instructions while the system is generated. Both system and application parameters are configured statically. Therefore, the special tool called the System Generator is used for this purpose. Special statements are designed to tune any parameter. The user should only edit the definition file, run the System Generator and then assemble the resulting files and the application ones. Thus, the user can adapt the Operating System for the desired task and the target hardware. The OS cannot be modified later at the run time.

Typographical Conventions

This Technical Reference employs the following typographical conventions:

Boldface type

Bold is used for important terms, notes and warnings.

Italics

Italics are used for all AUTOSAR names of directives, macros, constants, routines and variables.

`Courier` font

The courier typeface is used for code examples in the text.

References

[1] AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)

[2] Specification of ECU Configuration Parameters (AUTOSAR_EcucParamDef.arxml) V2.0.2 R.3.0 rev.0003

[3] OSEK/VDX System Generation OIL: OSEK Implementation Language, v.2.5, 01 July 2004

[4] OSEK/VDX Communication, v.3.0.3, 20 July 2004

[5] OSEK/VDX Operating System, v.2.2.2, 5 July 2004

[6] ORTI: OSEK Run Time Interface, v.2.0 (Draft c), 23 June 1999

[7] OSEK/VDX OSEK Run Time Interface (ORTI), Part A: Language Specification, v. 2.1.1, 4 March 2002

[8] OSEK/VDX OSEK Run Time Interface (ORTI), Part B: OSEK Objects and Attributes, v. 2.1, 17 April 2002

[9] AUTOSAR Specification of Memory Mapping v.1.1.0, 24 January 2007

Definitions, Acronyms and Abbreviations

The following acronyms and abbreviation are used in this Technical Reference.

API	Application Program Interface (a set of data types and functions)
AUTOSAR	Automotive Open System Architecture
BCC	Basic Conformance Class, a defined set of functionality in OSEK, for which waiting state of tasks is not permitted
BT	Basic task (the task which has no waiting state)
CCCB	OSEK Conformance Communication Class B

CPU	Central Processor Unit
ECC	Extended Conformance Class, a defined set of functionality in OSEK, for which waiting state of tasks is permitted
ET	Extended Task (the task which may have waiting state)
HW	Hardware
ID	Identifier, an abstract identifier of a system object
ISR	Interrupt Service Routine
MCU	Microcontroller Unit
MPU	Memory Protection Unit
N/A	Not applicable
OIL	OSEK Implementation Language
ORTI	OSEK Run Time Interface
OS	Operating System
OSEK	Open systems and the corresponding interfaces for automotive electronics (in German)
RAM	Random Access Memory
ROM	Read Only Memory
SC	Scalability Class, defines OS protection level
SG, SysGen	System Generator Utility
SW	Software

Technical Support Information

How to Contact Us:

Corporate Headquarters

```
Freescale Semiconductor, Inc.
7700 West Parmer Lane
Austin, TX 78729
U.S.A.
```

Technical Support: http://www.freescale.com/support

2

Operating System Architecture

This chapter gives a high level description of the OS architecture and presents the OS Conformance Classes.

This chapter consists of the following sections:

Processing Levels

The AUTOSAR Operating System provides a pool of different services and processing mechanisms. It serves as a basis for application programs which are independent of each other, and provides their environment on a processor. The AUTOSAR OS enables controlled real-time execution of several processes which virtually run in parallel.

The AUTOSAR Operating System provides a defined set of interfaces for the user. These interfaces are used by entities competing for the CPU. There are two types of entities:

- Interrupts (service routines managed by the Operating System);
- Tasks (basic tasks and extended tasks).

The highest processing priority is assigned to the interrupt level, where interrupt service routines (ISR) are executed. The interrupt services may call a number of operating system services. The processing level of the operating system has the priority immediately below the former one. This is the level on which the

operating system works: task management procedures, scheduler and system services. Immediately below there is the task level on which the application software is executed. Tasks are executed according to their user assigned priority. A distinction is made between the management of tasks with and without *waiting* state (*Extended* and *Basic Tasks*, see "Task Concept").

The following set of priority rules has been established:

- interrupts have precedence over tasks;
- the interrupt priority is defined by specific hardware conditions;
- for the items handled by the OS, bigger numbers refer to higher priorities;
- the task's priority is statically assigned by the user.

The Operating System provides services and ensures compliance with the set of priority rules mentioned above.

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 4 scalability classes SC1 .. SC4.

Table 2.1 AUTOSAR OS Scalability Classes

Feature	Scalabylity Class			
	SC1	SC2	SC3	SC4
OSEK OS (all conformance classes)	•	•	•	•
Counter Interface	•	•	•	•
Schedule Tables	•	•	•	•
Stack Monitoring	•	•	•	•
Timing Protection		•		•
Global Time and Synchronization Support		•		•
Memory Protection			•	•

Table 2.1 AUTOSAR OS Scalability Classes

Feature	Scalabylity Class			
	SC1	SC2	SC3	SC4
OS-Applications			•	•
Service Protection			•	•
CallTrustedFunction			•	•

The Freescale AUTOSAR OS / MPC56xxAM v.3.0 supports all 4 classes with additon of Service Protection in *Extended Status* for SC1, SC2 and Global Time Synchronization for ScheduleTable in all Scalability Classes.

Conformance Classes

Various requirements of the application software for the system, and various capabilities of a specific system (e.g. processor type, amount of memory) require different features of the operating system. These operating system features are described as *Conformance Classes* (CC). They differ in the number of services provided, their capabilities and different types of tasks.

The Conformance classes were created to support the following objectives:

- providing convenient groups of operating system features for easier understanding and discussion of the AUTOSAR operating system.
- allowing partial implementations along pre-defined lines. These partial implementations may be certified as AUTOSAR compliant.
- creating an upgrade path from the classes of less functionality to the classes of higher functionality with no changes to the application using AUTOSAR related features.

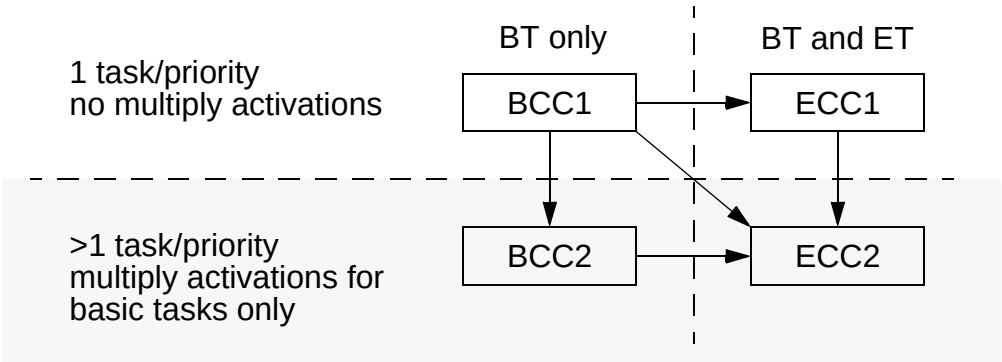
The required Conformance Class is selected by the user at the system generation time and cannot be changed during execution.

Definition of the functionalities provided by each Conformance Class depends on the properties of the tasks and the scheduling behavior. As the task properties (Basic or Extended, see "Task Concept") have a distinct influence on CC, they also assume part of their names. There are Basic-CC and Extended-CC, and each group can have various "derivatives".

The Conformance classes are determined by the following attributes:

- Multiply requesting of task activation - not supported by Freescale AUTOSAR OS;
- Task types (see "Task Concept");
- Number of tasks per priority.

Figure 2.1 Restricted Upward Compatibility for Conformance Classes



The OSEK OS specification defines the following Conformance Classes: BCC1, BCC2, ECC1, ECC2.The Freescale AUTOSAR OS does not support multiply activation and therefore it doesn't have BCC2 and ECC2 classes.

The Freescale AUTOSAR OS OS supports the following Conformance Classes:

- BCC1 – only Basic tasks, limited to one activation request per task and one task per priority, and all tasks have different priorities;
- ECC1 – like BCC1, plus Extended tasks.

Table 2.2 indicates the minimum resources to which an application may resort, determined for each Conformance Class in the OSEK OS.

Table 2.2 OSEK OS Conformance Classes

	BCC1	BCC2	ECC1	ECC2
Multiple activation of tasks	no	yes	BT: no, ET: no	BT: yes, ET: no
Number of tasks which are not in *suspended* state	>=8		>= 16, any combination of BT/ET	
Number of tasks per priority	1	>1	1 (both BT/ ET)	>1 (both BT/ ET)
Number of events per task	-		BT: no ET: >= 8	
Number of task priorities	>=8		>=16	
Resources	only Scheduler	>= 8 resources (including Scheduler)		
Internal Resources	>=2			
Alarm	>= 1 single or cyclic alarm			
Messages	possible			

The system configuration option *CC* (specified by the user) defines the class of the overall system. In the Freescale AUTOSAR OS this option can have the values *BCC1* and *ECC1* or it can be set to *AUTO* (see "OS Definition").

Maximal numbers of the Freescale AUTOSAR OS / MPC56xxAM system objects are indicated in Table 2.3.

Table 2.3 Freescale AUTOSAR OS/MPC56xxAM Maximal System Resources

Number of OS-Applications	8
Number of task's priorities	64
Number of tasks which are not in suspended state	64
Number of events per task	32
Number of resources (including RES_SCHEDULER)	2047
Number of Apllication Modes	8
Number of all other OS objects	2047

Note that actual number of OS objects might be limited by amount of available memory.

AUTOSAR OS Overall Architecture

The AUTOSAR OS is a real-time operating system which is executed within a single electronic control unit. It provides local services for the user's tasks. The AUTOSAR OS consists of the following components:

- ***Scheduler*** controls the allocation of the CPU for different tasks;
- ***Task management*** provides operations with tasks;
- ***ISR management*** provides entry / exit frames for interrupt service routines and supports CPU interrupt level manipulation;
- ***Resource management*** supports a special kind of semaphore for mutually exclusive access to shared resources;
- ***Local communication*** provides message exchange between tasks;
- ***Counter management*** provides operations on objects like timers and incremental counters;
- ***Alarm management*** links tasks and counters;
- ***Memory, Timing and Service protection*** subsystems;
- ***Error handlers*** handle the user's application errors and internal errors, and provide recovery from the error conditions;
- ***Hook routines*** provide additional debugging features;
- ***System start-up*** initializes data and starts the execution of applications;
- ***System timers*** provides implementation-independent time management.

The Scalabilty Classes provide the various level of AUTOSAR OS protection, see Table 2.1

As it is shown in Table 2.2, the Conformance Classes, in general, differ in the degree of services provided for the task management and scheduling (the number of tasks per priority, multiple requesting, Basic / Extended Tasks). In higher CC an advanced functionality is added for the resource management and event

management only. But even in BCC1 the user is provided with almost all the AUTOSAR OS service mechanisms.

The AUTOSAR Operating System is not scaled through the Classes only, but it also has various extensions which can be used in any Scalability / Conformance Classes. These extensions affect memory requirements and overall system performance. The extensions can be turned on or off with the help of the corresponding system configuration options. They all are described in "System Objects Definition".

Since the AUTOSAR Operating System is fully statically configured, the configuration process is supported by the *System Generator* (SG). This is a command-line utility, which processes system generation statements defined by the user in a special file. These statements fully describe the required system features and application object's parameters. The SG produces a header file (osprop.h) which is used for system compilation and C-code files to be compiled together with the other user's source code. The produced code consists of C-language definitions and declarations of data as well as C-preprocessor directives. See "System Configuration" and "Building of Application" for details on system generation.

Application Program Interface

The AUTOSAR Operating System establishes the Application Program Interface (API) which must be used for all the user actions connected with system calls and system objects. This API defines the data types used by the system, the syntax of all run-time service calls, declarations and definitions of the system.

The AUTOSAR OS data types are described in the subsections dedicated to the corresponding mechanisms. The syntax of system calls and system configuration statements is described briefly in the corresponding subsections and in detail in "System Objects Definition" and "System Services".

NOTE The user's source code shall strictly correspond to the rules stated in this Technical Reference.

The AUTOSAR OS may be compiled in *Extended Status*. It means that an additional check is made within all OS activities and extended return codes are returned by all the OS services to indicate errors, if any. See "System Services" and "Error Handling" about Extended Status return values. In order to provide the Extended Status in the system, the configuration option *STATUS* must be set to *EXTENDED* at the configuration stage. For Scalability Classes SC3 and SC4 the *EXTENDED STATUS* is mandatory.

The Freescale AUTOSAR OS OS provides support for the "OSEK aware" debuggers by means of the OSEK Run Time Interface (ORTI). See "Debugging Application" and "Global System Attributes" for details.

AUTOSAR OS Protection features

Protection is only possible for OS managed objects. The ISRs of category 1 are not managed by the OS, thus no protection is provided during runtime of this ISRs. Therefore, if any protection is required, ISRs of category 1 are deprecated in classes SC2 .. SC4.. If Category 1 interrupts are used they shall belong to a trusted OS-Application in SC3 and SC4.

The OS calls the user's ProtectionHook in case of memory and timing violations or any exception. After return from the hook OS performs the action according to return code, please see "ProtectionHook" for details.

NOTE The OS does not clears the interrupt request flags for killed ISRs. It is deviation from AUTOSAR OS v.3.0 specification.

Memory Protection

The memory protection subsystem exists in SC3 and SC4 classes. It protects OS-Application data and code from modification by other non-Trusted OS-Application.

The memory protection operates with data, code and stack sections of the executable program.

The Operation System prevents write access to OS own data sections and OS own stack.

The Operation System prevents write access from non-trusted OS-Applications to:

- all private stacks of Task(s) / Category 2 ISR(s) except runnable own stack;
- private data sections of all other OS-Applications;
- memory mapping peripheral space except specially assigned by User address regions (via MemData<0 / 1 / 2>)

The Operation System permits:

- a Task(s) / Category 2 ISR(s) read and write access to that Task's / Category 2 ISR's own private stack;
- a non-trusted OS-Applcation read and write access to that OS-Application own private data sections;

The trusted OS-Applications have the same access rights as the OS with the exception that access to the stack area is limited.

In case of memory violation the OS calls the ProtectionHook(*E_OS_PROTECTION_MEMORY*) and performs an appropriate action defined by its return value. Three special variables are available for reading inside ProtectionHook to find what has caused the violation:

- *OsViolatorAppId* - contains the Application ID that caused violation or INVALID_OSAPPLICATION
- *OsViolatorTaskId* - contains the ID of the Task that caused violation or INVALID_TASK
- *OsViolatorISRId* - contains the ID of the ISR that caused violation or INVALID_ISR

This 3 variables are not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.

The Operating System provides the ability to read and execute all code sections for all OS-Applications.

Strict discipline of data placement is required to provide memory protection in AUTOSAR OS. The User shall:

- Set MemData<0,1,2> attribute of APPLICATION object in the configuration file (MemData0 is set by default).
- Place special START*/STOP* macros in User's source code. They control the assignment of variables and functions to specific sections.
- Create the memory mapping header file MemMap.h with these special macros (except types VAR_*, VAR_FAST_*, CONST_* and CONST_FAST_) and lists of sections.

  Macros VAR_*, VAR_FAST_*, CONST_* and CONST_FAST_ are used for assignment of OS-Applications variables and constants to specific sections. These macros are generated by SysGen in osmemmap.h file. This generated file shall be included in MemMap.h file.

  MemMap.h shall be included after each special macro in the User's source code.
- Create template Linker command file including special comments placed in dedicated memory regions where the User is going to allocate OS-Applications variables. SysGen uses it to generated an output Linker configuration file.

The OS allows placement of OS-Applications data into up to 3 memory regions that can be mapped onto different physical memory. Attribute ***MemData<0,1,2>*** of APPLICATION object defines whether the OS-Application uses the given (0,1,2) memory region for data placement. It allows to place data into different physical RAM. This mechanism may be used to access memory mapped peripherals from non-trusted OS-Applicaton.

Each OS-Application shall wrap declaration and definition of code, variables and constants using the following mechanism according to *AUTOSAR Specification of Memory Mapping v.1.1.0, 24 January 2007*:

1. Definition of start macro for OS-Application memory section:

- <APP_NAME>_START_SEC_VAR<MEM_REGION>_<TYPE>_<SIZE>
  <APP_NAME>_START_SEC_CONST_<TYPE>_<SIZE>

  Where:

  <APP_NAME> -

OS-Application name (the name of APPLICATION object from the configuration file)

<MEM_REGION> -

- Empty for MemData0 and _FAST_ type
- '1' for MemData1
- '2' for MemData2

<TYPE> -

- VAR, used for global or static variables
- CONST, used for global or static constants
- VAR(CONST)_FAST, used for all global or static variables / constants that are placed into "small" data memory (allows fast addressing mode). The assignment of these data to dedicated memory regions is responsibility of the User.

<SIZE> -

- 8BIT, used for variables and constants of size 8 bit
- 16BIT, used for variables and constants of size 16 bit
- 32BIT, used for variables and constants of size 32 bit
- UNSPECIFIED, used for variables and constants of unknown size

- < APP_NAME>_START_SEC_CODE_<TYPE>

2. Inclusion of the memory mapping header file:

According to AUTOSAR OS v.3.0 specification the memory mapping file name is 'MemMap.h'

3. Declaration / definition of code, variables or constants belonging to the specified section.

4. Definition of stop macro for OS-Application memory section:

- <APP_NAME>_STOP_SEC_VAR<MEM_REGION>_<TYPE>_<SIZE>
- <APP_NAME>_STOP_SEC_CONST_<TYPE>_<SIZE>
- < APP_NAME>_STOP_SEC_CODE_<TYPE>

The OS is not responsible for usage of other START* / STOP* macros described in *AUTOSAR Specification of Memory Mapping v.1.1.0, 24 January 2007*

5. Inclusion of the memory mapping header file

Sample Application contains the example of memory protection usage:

```
...
#define SND_APP_START_SEC_VAR_FAST_16BIT
#include "MemMap.h"
volatile unsigned short ind, taskSnd1, taskSnd2, taskCnt;
#define SND_APP_STOP_SEC_VAR_FAST_16BIT
#include "MemMap.h"
...
#define APP_START_SEC_CODE
#include "MemMap.h"
...
/* the application code (TASKSND1, TASKSND2 and TASKCNT) */
...
#define APP_STOP_SEC_CODE
#include "MemMap.h"
```

Memory mapping files contains lists of sections for the assignment of variables and functions:

```
...
#if defined(APP_START_SEC_CODE)

#undef APP_START_SEC_CODE
#undef MEMMAP_ERROR
...
#pragma section code_type ".appcode"
...
#elif defined(APP_STOP_SEC_CODE)
...
/* Types generated by OS Sysgen: VAR_*, VAR_FAST_*, CONST_* and
CONST_FAST_* */
#include "osmemmap.h"
...
```

The generated ‘osmemmap.h’ file contatins lists of sections for OS-Applications variables and constants allocation.

```
..
/*  --- CONST DATA ---  */
#if defined(SND_APP_START_SEC_CONST_8BIT)
...
#pragma section const_type ".sapp_c8" ".sapp_c8"
```

```
...
#elif defined(SND_APP_STOP_SEC_CONST_8BIT)
...
#pragma section const_type
...
/*  --- FAST DATA (in direct page)  ---  */
#if defined(SND_APP_START_SEC_VAR_FAST_16BIT)
...
#pragma section data_type ".app0_v16" ".app0_v16"
...
#elif defined(SND_APP_STOP_SEC_VAR_FAST_16BIT)
...
#pragma section data_type
...
#endif
...
```

Sections generated by SysGen in osmemap.h file are named by the rule below.

- For constants <prefix>app_<type>c<size>
- For variables <prefix>app<app_number>_<type>v<size><_mem_region>

Where:

<prefix> - optional

– 's' for small data section for variables from trusted OS-Applications or constants

<app_number>

Not applicable for small data section (<prefix> ‘s’ is defined)

OS-Application ID associated with OS-Application in the application configuration header generated by SysGen(2. on page 169).

<size>

– '8' 8bit
– '16' 16 bit
– '32' 32 bit
– 'u' unknown size

<type> - optional

- 'f' fast data memory segments

<_mem_region> - optional

- empty for MemData0
- '1' for MemData1
- '2' for MemData2

The assignment of the sections to dedicated memory areas / address ranges is not the scope of the memory mapping file and is done via linker command files.

The User shall create Linker command file with special comments:

```
MEMORY
{...
  /* The memory areas for applications. Do not edit or delete this
comment: it is used by the SysGen for <compiler> */
  /* The end of the memory areas for applications. Do not edit or
delete this comment: it is used by the SysGen for <compiler> */
...}
SECTIONS
{...
  /* Begin of the RAM area <mem_region> for applications. Do not
edit or delete this comment: it is used by the SysGen for
<compiler> */
  /* End of the RAM area <mem_region> for applications. Do not
edit or delete this comment: it is used by the SysGen for
<compiler> */
...}
```

Where:

<mem_region>:

- Empty for MemData0
- '#1' for MemData1
- '#2' for MemData2

This linker command file is used as “template” to generate an output linker configuration file. SysGen inserts required directives between special comments. This output linker file contains assignment of sections defined in the generated 'osmemmap.h' file to dedicated memory areas / address ranges.

All code and constant sections shall be placed between .osbegincode and .osendcode sections. It is strongly recommended for non-trusted OS-Applcations.

MPU has 32-byte granularity so SysGen generates Linker directives to align non-trusted OS-Application data to 32-byte. Trusted OS-Application data are aligned by default.

Size of stacks of Tasks / ISRs from non-trusted OS-Application may be increased due to alignment to 32-byte (SC3 and SC4) .

The user can check data allocation correctness in the map file.

Timing Protection

For safe and accurate timing protection it is necessary to enforce limits on the factors that determine whether or not Tasks / ISRs meet their respective deadlines. These limits are:

(1) the execution time of each Task / Category 2 ISR;

(2) the execution time of each Task / Category 2 ISR while holding shared resources / disabling interrupts;

NOTE SuspendAll/DisableAllInterrupts are not controlled via Time Protection.

(3) the inter-arrival rate of each Task / Category 2 ISR.

Timing protection only applies to Tasks or Category 2 ISRs, but not to Category 1 ISRs. The Timing Protection is available in SC2 and SC4 classes.

NOTE The User shall apply Timing Protection for any Task/Category 2 ISR belonging to non-trusted OS-Application in SC4 class.

Task Timing Protection

The Task attribute ***TIMING_PROTECTION*** specifies whether Time Protection is applied to the Task.

The execution time protection can be applied to Task via defining Task Execution Budget (the value of ***EXECUTIONBUDGET*** attribute).

The OS resets a Task Execution Budget on a transition to the *SUSPENDED* or *WAITING* states.

The time that the Task locks Resource is controlled via Time Protection with the following static configured attributes:

- ***LOCKINGTIME*** specifies whether Time Protection is applied to the Resource(s).
- ***RESOURCE*** (reference) specifies to which Resource the locking time protection is applied.
- ***MAXRESOURCELOCKTIME*** defines the maximum locking time for the Resource.

The time disables OS interrupts via Suspend / ResumeOSInterrupts may be controlled via Time Protection with the following static configured attribute:

- ***MAXOSINTERRUPTLOCKTIME*** defines the maximum locking time of the OS interrupts (ISRs of category 2 and OS System timers).

An inter-arrival time protection is controlled via defining the length of timeframe for inter-arrival rate protection (the value of ***TIMEFRAME*** attribute).

The OS starts an timeframe when a task is activated / released successfully. If an attempt is made to activate / release a task before the end of an timeframe then the OS does not perform the activation / release.

The OS limits the inter-arrival time of Task to one per timeframe.

ISR Timing Protection

The ISR attribute ***TIMING_PROTECTION*** specifies whether Time Protection is applied to the Category 2 ISR.

The execution time protection can be applied to Category 2 ISR via defining Category 2 ISR Execution Budget (the value of ***EXECUTIONTIME*** attribute).

The OS rests an ISR's Execution Budget when the ISR returns control to the Operating terminates.

The time that the Category 2 ISR locks Resource may be controlled via Time Protection with the following static configured attributes:

- ***LOCKINGTIME*** specifies whether Time Protection is applied to the Resource(s).
- ***RESOURCE*** (reference) specifies to which Resource the locking time protection is applied.
- ***MAXRESOURCELOCKTIME*** defines the maximum locking time for the Resource.

The time disables OS interrupts via Suspend / ResumeOSInterrupts may be controlled via Time Protection with the following static configured attribute:

- ***MAXOSINTERRUPTLOCKTIME*** defines the maximum locking time of the OS interrupts (ISRs of category 2 and OS System timers).

An inter-arrival time protection is controlled via defining the length of timeframe for arrival rate protection (the value of ***TIMEFRAME*** attribute).

The OS limits the inter-arrival time of Category 2 ISR to one per timeframe.

The OS measures the start of an timeframe from the point at which it recognises the interrupt.

If Category 2 ISR occurs before the end of the timeframe then the OS does not execute the user provided ISR.

Protection Hook

In case of timing violation the OS calls the ProtectionHook() with the following parameters:

- *E_OS_PROTECTION_TIME*

  Task / Category 2 ISR Execution Budget is reached.
- *E_OS_PROTECTION_ARRIVAL*
  - An attempt is made to activate / release a task before the end of an timeframe.

  - Category 2 ISR occurs before the end of the timeframe.
- *E_OS_PROTECTION_LOCKED*
  - A Task / Category 2 ISR holds an Resource and exceeds the maximum locking time for the Resource
  - A Task / Category 2 ISR disables interrupts and exceeds the configured maximum locking time

Three special variables are available for reading inside ProtectionHook to find what has caused the violation:

- *OsViolatorAppId* - contains the Application ID that caused violation or INVALID_OSAPPLICATION
- *OsViolatorTaskId* - contains the ID of the Task that caused violation or INVALID_TASK
- *OsViolatorISRId* - contains the ID of the ISR that caused violation or INVALID_ISR

This 3 variables are not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.

Service Protection

As OS-Applications interact with the OS through services, it is essential that the service calls will not corrupt the OS itself. Service Protection guards against such corruption at runtime. The AUTOSAR OS specification [1] requires Service Protection in SC3 and SC4 classes, but the Freescale AUTOSAR OS provides it also in SC1 and SC2 when configured with *EXTENDED* Status.

There are a number of cases considered within Service Protection: An OS-Application makes an API call

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

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

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

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

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

All this cases are handled by AUTOSAR OS Service protection. The codes returned in case of error are defined in each service description in the capter "System Services"

The Table 2.4 indicates the allowed context for the OS service calls.

Table 2.4 Allowed Services Contexts

System service	Task	ISR Cat.1	ISR Cat.2	ErrorHook	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	Alarm Callback	ProtectionHook
ActivateTask	•		•							
TerminateTask	•		•							
ChainTask	•		•							
Schedule	•		•							
GetTaskID	•		•	•	•	•				•
GetTaskState	•		•	•	•	•				
EnableAllInterrupts	•	•	•	•	•	•	•	•	•	•
DisableAllInterrupts	•	•	•	•	•	•	•	•	•	•
ResumeAllInterrupts	•	•	•	•	•	•	•	•	•	•
SuspendAllInterrupts	•	•	•	•	•	•	•	•	•	•
ResumeOSInterrupts	•	•	•	•	•	•	•	•	•	•
SuspendOSInterrupts	•	•	•	•	•	•	•	•	•	•
GetResource	•		•							

Table 2.4 Allowed Services Contexts

System service	Task	ISR Cat.1	ISR Cat.2	ErrorHook	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	Alarm Callback	ProtectionHook
ReleaseResource	•		•							
SetEvent	•		•							
ClearEvent	•		•							
GetEvent	•		•	•	•	•				
WaitEvent	•		•							
GetAlarmBase	•		•	•	•	•				
GetAlarm	•		•	•	•	•				
SetRelAlarm	•		•							
SetAbsAlarm	•		•							
CancelAlarm	•		•							
GetActiveApplicationMode	•		•	•	•	•	•	•		
StartOS										
ShutdownOS	•		•	•			•			
GetApplicationID	•		•	•	•	•	•	•		•
GetISRID	•		•	•						•
CallTrustedFunction	•		•							
CheckISRMemoryAccess	•		•	•						•
CheckTaskMemoryAccess	•		•	•						•
CheckObjectAccess	•		•	•						•
CheckObjectOwnership	•		•	•						•
StartScheduleTableRel	•		•							
StartScheduleTableAbs	•		•							

Table 2.4 Allowed Services Contexts

System service	Task	ISR Cat.1	ISR Cat.2	ErrorHook	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	Alarm Callback	ProtectionHook
StopScheduleTable	•		•							
NextScheduleTable	•		•							
SyncScheduleTable	•		•							
GetScheduleTableStatus	•		•							
SetScheduleTableAsync	•		•							
IncrementCounter	•		•							
TerminateApplication	•		•	• [a]						
DisableInterruptSource	•		•							
InitCounter[b]	•									
GetCounterValue[a]	•		•							
GetCounterInfo[a]	•		•							
SendMessage[c]	•		•							
ReceiveMessage[b]	•		•							
GetMessageStatus[b]	•		•							
ReadFlag[b]	•		•							
ResetFlag[b]	•		•							
StartCOM[b]	•		•							
StopCOM[b]	•		•							
GetCOMApplicationMode[b]	•		•							
InitMessage[b]	•		•							
GetRunningStackUsage[a]	•		•	•	•	•				

Table 2.4 Allowed Services Contexts

System service	Task	ISR Cat.1	ISR Cat.2	ErrorHook	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	Alarm Callback	ProtectionHook
GetStackUsage[a]	•		•	•	•	•				
GetTimeStamp[a]	•		•	•	•	•				

a. only in application specific ErrorHooks.

b. this service is not specified inAUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS

c. this service is specified in OSEK COM [4]

The Freescale AUTOSAR OS checks the allowed context in *EXTENDED* status for all Scalability Classes.

Tasks/ISRs with incorrect behavior

The OS dispatcher handles the cases when the Task returns w/o call to TerminateTask and when Task/ISR does not releases Resource or fails to call Resume*Interrupts() after Suspend*Interrupts(). In this cases the OS does required cleanup, releases locked objects and calls ErrorHook if it is configured.

3

Task Management

This chapter describes the task concept of AUTOSAR and all other questions related to tasks.

This chapter consists of the following sections:

Task Concept

Complex control software can be conveniently subdivided into parts executed according to their real-time requirements. These parts can be implemented by means of tasks. The task provides the framework for execution of functions. The Operating System provides parallel and asynchronous task execution organization by the scheduler.

Two different task concepts are provided by the AUTOSAR OS:

- Basic Tasks (BT);
- Extended Tasks (ET).

The Basic Tasks release the processor only if:

- they are being terminated,
- the AUTOSAR OS is executing higher-priority tasks, or
- interrupt occurred.

The Extended Tasks differ from the Basic Tasks by being allowed using additional operating system services which may result in *waiting* state. *Waiting* state allows the processor to be freed and

reassigned to a lower-priority task without the necessity to terminate the Extended Task.

The task type is determined automatically. If a TASK object has a reference to EVENT, the task is considered to be Extended.

Both types of tasks have their advantages which must be compared in context of application requirements. They both are justified and supported by the AUTOSAR operating system.

Every task has a set of related data: task description data located in ROM and task state variables in RAM. Also, every extended task has its own stack assigned. In Scalability Classes with protecttion (SC2..SC4) each task has its own stack.

Every running task is represented by its run-time context. This refers to the CPU general purpose registers and some special registers. When the task is interrupted or preempted by another task, the run-time context is saved.

The task has several states since the processor can execute only one instruction of the task at any time, but at the same time several tasks may compete for the processor. The AUTOSAR OS is responsible for saving and restoring the task context in conjunction with state transitions whenever necessary.

Extended Tasks

The Extended Tasks have four task states:

- *running*
  In 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 the same point in time, while all the other states can be adopted simultaneously by several tasks.
- *ready*
  All functional prerequisites for transition into running state are met, and the task only waits for allocation of the processor. The scheduler decides which of the ready tasks is executed next.
- *waiting*
  A task cannot be executed (any longer), because it has to wait for at least one event (see “Events”).

- *suspended*
  In suspended state the task is passive and does not occupy any resources, merely ROM.

Figure 3.1 Status Model with Task Transitions for an Extended Task

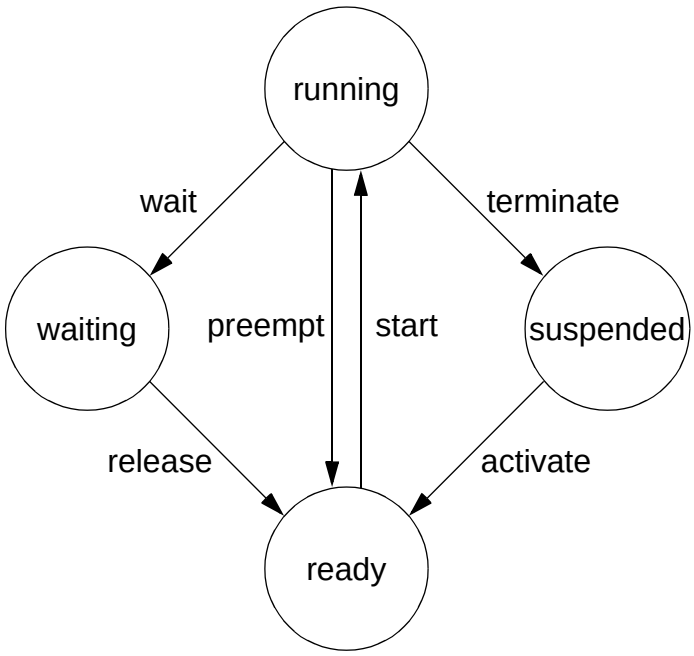


Table 3.1 States and Status Transitions for an Extended Task

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 an operation, the *running* task requires an event. It causes its transition into *waiting* state by using a system service.
release	waiting	ready	Events have occurred which a task has been waiting for.

Table 3.1 States and Status Transitions for an Extended Task

Transition	Former state	New state	Description
preempt	running	ready	The scheduler decides to start another task. The *running* task is put into *ready* state.
terminate	running	suspended	The *running* task causes its transition into *suspended* state by a system service.

Termination of tasks is possible only if the task terminates itself ('self-termination').

There is no provision for a direct transition from *suspended* into *waiting* state. This transition is redundant and would make the scheduler more complicated. *Waiting* state is not directly entered from *suspended* state since the task starts and explicitly enters *waiting* state on its own.

Basic Tasks

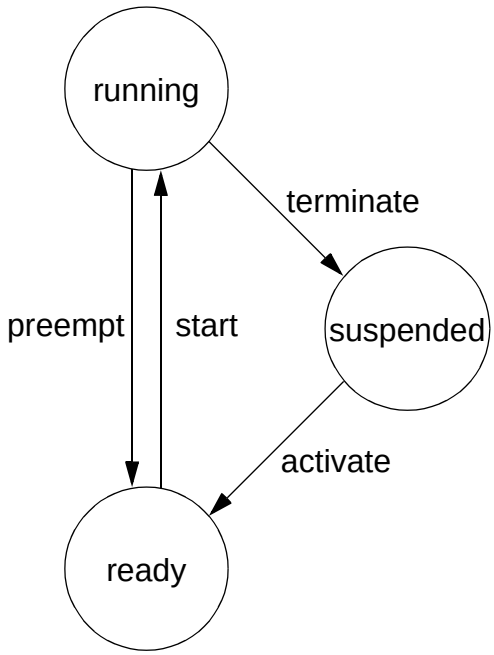
The state model for the Basic Tasks is nearly identical to the one for the Extended Tasks. The only exception is the absence of *waiting* state.

- *running*
  In 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 the same point in time, while all the other states can be adopted simultaneously by several tasks.
- *ready*
  All functional prerequisites for transition into running state are met, and the task only waits for allocation of the processor. The scheduler decides which of the ready tasks is executed next.
- *suspended*
  In suspended state the task is passive and does not occupy any resources, merely ROM.

Table 3.2 States and Status Transitions for a Basic Task

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.
preempt	running	ready	The scheduler decides to start another task. The *running* task is put into *ready* state.
terminate	running	suspended	The *running* task causes its transition into *suspended* state by a system service.

Figure 3.2 Status Model with Task Transitions for a Basic Task



Task Priorities

The AUTOSAR OS specifies the value 0 as the lowest task priority in the operating system. Accordingly bigger numbers define higher task priorities.

The task priorities defined in OIL are not intersected with ISR priorities. Interrupts have a separate priority scale. All task priorities are lower than any ISR priorities and the scheduler priority.

In the AUTOSAR OS the priority is statically assigned to each task and it cannot be changed at run-time. A dynamic priority management is not supported. However, in particular cases the operating system can change task priority. In this context, please refer to "Priority Ceiling Protocol".

When rescheduling is performed, the scheduler always switches to the task with the highest priority among the *ready* tasks and the *running* one.

Tasks Stacks

Stack Allocation

In the Scalability Classes SC2 .. SC4 each Task has its own stack.

In the SC1 each extended task has its own statically allocated stack, while Basic Tasks use common stack.

The minimal size of the task stack depends on:

- the scheduling policy (non-preemptable or preemptable task);
- the services used by the task;
- the interrupt and error handling policy;
- the processor type.

The recommended values of the minimal task stack size are provided in "Stack Size".

NOTE If the task stack is less than the required value for the given application, it may cause unpredictable behavior of the task and a system protection error or crash.

Single Stack

The single stack is used for the basic tasks. In the BCC1 conformance class within SC1 there is only one single stack in the OS. It is used for all tasks, interrupts and dispatcher.

In Scalability Classes higher then SC1 all Tasks and ISRs has theier own stacks, the Single Stack is used only in SC1.

In the ECC1, SC1 class all basic tasks use the single stack. But the extended tasks have their own stacks. Interrupts have a separate stack, common for all ISRs of category 2 and system timers ISRs.

In SC1 class Freescale AUTOSAR OS uses the main application stack for a single stack. Its size is defined in the linker command file.

Tasks Timing protection

The timing protection can be applied to Task via defining Task Execution Budget and a timeframe for arrival rate protection. Also the time that the Task locks Resource or disables OS interrupts via Suspend / ResumeOSInterrupts services may be controlled via Time Protection. See Task Definition for the parameters of Timing Protection.

Programming Issues

Configuration Options

The following system configuration options affect the task management:

- *STATUS*
  Specifies error checking at run-time.

- *StackOverflowCheck*
  Turns on stack overflow runtime checking and stack usage services.
- *Pattern*
  Defines the pattern used to fill stack memory. Possible values are 0..0xFFFFFFFF
- *PatternSize*
  Defines the size of the area to be checked in 32-bit words.
- *CC*
  Specifies the conformance class. If AUTO, the conformance class is defined according to the Tasks definitions.

Data Types

The AUTOSAR OS establishes the following data types for the task management:

- *TaskType*
  The abstract data type for task identification;
- *TaskRefType*
  The data type to refer the variables of the *TaskType* data type. Reference to the *TaskType* variable can be used instead of the *TaskRefType* variable;
- *TaskStateType*
  The data type for the variables for storage of the task state;
- *TaskStateRefType*
  The data type to refer the variables of the *TaskStateType* data type. Reference to the *TaskStateType* variable can be used instead of the *TaskStateRefType* variable.

Only these data types may be used for operations with tasks.

Task Definition

Every task in an application is generated using the *TASK* system generation object with the set of properties in OIL file. These properties define the task behavior and the resource allocation method. Each task property has its own name, and the user defines the task's features by setting the corresponding properties in the task definition. See also "System Objects Definition".

The task definition looks like the following:

```
TASK TASKSENDER {
  PRIORITY = 5;
  SCHEDULE = FULL;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  RESOURCE = MYRESOURCE;
  RESOURCE = SECONDRESOURCE;
  EVENT = MYEVENT;
  STACKSIZE = 256;
  MESSAGE = MYMESSAGE;
  TIMING_PROTECTION = FALSE;
};
```

A description of possible task properties is indicated in Table 3.3.

Table 3.3 Task Properties

Object Parameters	Possible Values	Description
Standard Attributes		
PRIORITY	integer [0...0xFFFFFFFF]	Defines the task priority. The lowest priority has the value 0
SCHEDULE	FULL, NON	Defines the run-time behavior of the task
AUTOSTART	TRUE, FALSE	Defines whether the task is activated during the system start-up procedure or not
APPMODE	name of APPMODE	Defines the application mode in which the task is auto-started
ACTIVATION	1	Specifies the maximum number of queued activation requests for the task (Freescale AUTOSAR OS does not allow multiply activations)
RESOURCE	name of RESOURCE	Resources accessed by the task. There can be several resource references
EVENT	name of EVENT	Event owned by the task. There can be several event references
ACCESSOR	SENT, RECEIVED	Defines the type of usage of the message
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the task
STACKSIZE	integer	Defines the size of the task stack in bytes.

Table 3.3 Task Properties

TIMING_PROTECTION	TRUE, FALSE	Specifies is the Timing Protection applied to the TASK

The application definition file contains one such statement per task. The task generation statement is described in detail in "System Objects Definition".

Run-time Services

The AUTOSAR OS provides a set of services for the user to manage tasks. A detailed description of these services is provided in "System Services". Below is only a brief list of them.

Table 3.4 Task Management Run-time Services

Service Name	Description
ActivateTask	Activates the task, i.e. moves it from *suspended* to *ready* state
TerminateTask	Terminates the running task, i.e. moves it from *ready* to *suspended* state
ChainTask	Terminates the running task and activates the new one immediately
Schedule	Yields control to a higher-priority ready task (if any)
GetTaskID	Gets the identifier of the running task
GetTaskState	Gets the status of a specified task

Examples of using the run-time services are provided in "ISR Management Services".

Constants

The following constants are used within the AUTOSAR Operating System to indicate the task states:

- *RUNNING*
  The constant of data type *TaskStateType* for task state *running*
- *WAITING*
  The constant of data type *TaskStateType* for task state *waiting*
- *READY*
  The constant of data type *TaskStateType* for task state *ready*

- *SUSPENDED*
  The constant of data type *TaskStateType* for task state *suspended*

These constants can be used for the variables of *TaskStateType*.

The following constant is used within the AUTOSAR OS to indicate the task:

- *INVALID_TASK*
  The constant of data type *Task Type* for an undefined task

Conventions

Within the AUTOSAR OS application a task should be defined according to the following pattern:

```
TASK ( TaskName )
{
...
}
```

The name of the task function is generated from *TaskName* by macro *TASK*.

4

Scheduler

This chapter provides a description of scheduling policies defined for OSEK OS.

This chapter consists of the following sections:

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 the 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 (see section "Task Priorities") and the scheduling policy used.

The scheduler is activated whenever a task switch is possible according to the scheduling policy. The principle of multitasking allows the operating system to execute various tasks concurrently. The sequence of their execution depends on the scheduling policy, therefore it has to be clearly defined.

Scheduler also provides the endless idle loop if there is no task ready to run. It may occur, when all tasks are in the *suspended* or *waiting* state until the awakening signal from an Interrupt Service Routine occurs. In this case there is no currently running task in the system, and the scheduler occupies the processor performing an endless loop until the ISR awakes a task to be executed.

The scheduler can be treated as a specific resource that can be occupied by any task. See "Scheduler as a Resource" for details.

The scheduling policy and some scheduler-related parameters are defined by the user, see "Global System Attributes".

Scheduling Policy

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. The scheduling policy affects the system performance and memory resources. In the OSEK Operating System, all listed scheduling policies are supported. Each task in an application may be preemptable or not. It is defined via the appropriate task property (preemptable / non-preemptable).

Note that the interruptability of the system depends neither on the Conformance Class, nor on the scheduling policy.

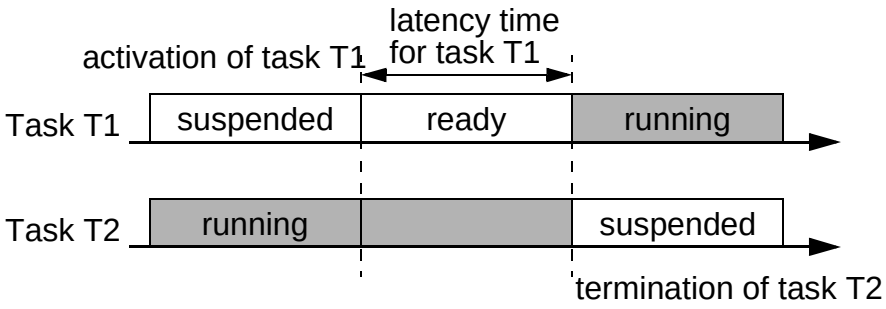
The desired scheduling policy is defined by the user via the tasks configuration option *SCHEDULE*. The valid values are – *NON and FULL*. If all tasks use NON scheduling, scheduler works as non-preemptive. If all tasks use FULL scheduling, scheduler works as full-preemptive. If some tasks use NON and other tasks use FULL scheduling, scheduler works as mixed-preemptive.

Non-preemptive Scheduling

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 lower priority non-preemptable section of a running task 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 4.1 Non-preemptive Scheduling



Task T2 has lower priority than task T1. Therefore, it delays task T1 up to the point of rescheduling (in this case termination of task T2).

Only four following *points of rescheduling* exist in the OSEK OS for non-preemptive scheduling:

- Successful termination of a task (via the *TerminateTask* system service);
- Successful termination of a task with explicit activation 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-preemptable and the task switching will take place exactly in the listed cases.

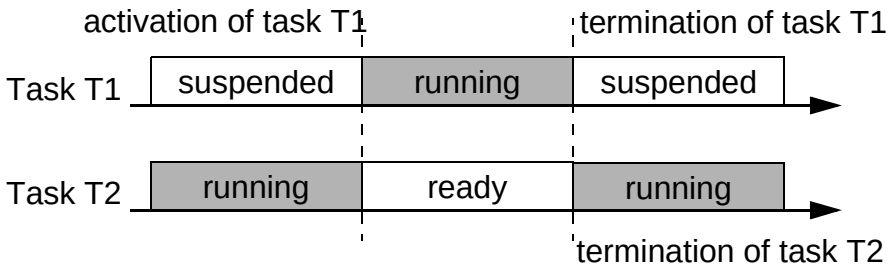
Full-preemptive Scheduling

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 enhanced complexity of features necessary for synchronization between tasks. As each task can theoretically be rescheduled at any location, access to data that are used jointly with other tasks must be synchronized.

In a full-preemptive system all tasks are preemptable.

Figure 4.2 Full-preemptive Scheduling



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 (preemptable / non-preemptable) of the running task.

The definition of a non-preemptable 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 as the time of the task switch,
- if the task must not be preempted.

Many applications comprise only a few parallel tasks with a long execution time, for which a full-preemptive scheduling policy 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.

Groups of Tasks

The operating system allows tasks to combine aspects of preemptive and non-preemptive scheduling by defining groups of tasks. For the tasks which have the same or lower priority as the highest priority within a group, the tasks within the group behave like non-preemptable tasks: rescheduling will only take place at the points of rescheduling described in "Non-preemptive Scheduling" For tasks with a higher priority than the highest priority within the group, tasks within the group behave like preemptable tasks (see "Full-preemptive Scheduling").

Chapter "Internal resources" describes the mechanism of defining groups by the instrumentality of internal resources.

Programming Issues

Configuration Options

The following system configuration options are intended to define scheduler properties:

- *CC*
  Specifies conformance class. If AUTO, conformance class is defined according to tasks definitions.
- *USERESSCHEDULER*
  If this option is set to FALSE then *RES_SCHEDULER* is not supported by OS.

Run-time Services

The scheduler is not accessed by the user directly. The user can only pass the CPU control to the scheduler by means of the *Schedule* system service. This leads to task rescheduling if there is a ready task of priority higher than the running one.

The scheduler can be used by the programmer as a resource. To provide this possibility, the services *GetResource* and *ReleaseResource*

with the constant *RES_SCHEDULER* as a parameter can be called by a task. It means that the task cannot be preempted by any other task after the scheduler occupation, before the corresponding call *ReleaseResource* is performed. While the task occupies the scheduler, it has the highest priority and, therefore, cannot be preempted by other tasks (only ISRs can get the CPU control during this period). Such programming practice can be used for important critical sections of code.

See the example:

```
GetResource( RES_SCHEDULER );
...
/* Critical section */
/* this code cannot be interrupted by any other task */
...
ReleaseResource( RES_SCHEDULER );
```

5

Interrupt Processing

This chapter highlights AUTOSAR OS approach to the interrupt handling.

This chapter consists of the following sections:

General

Interrupt processing is an 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.

In the Freescale AUTOSAR OS / MPC56xxAM in SC3 and SC4 classes all ISRs of category 2 use the configured stacks. In SC1 with ECC1 class and in SC2 all ISRs of category 2 use the “ISR stack” which is used only by ISRs during their execution. The size of the ISR stack(s) is defined by the user. In the classes with memory protection (SC3, SC4) all ISRs of category 2 with the same PRIORITY value, including System and Second Timer ISRs, shares the stack memory. The OS protection handlers has theire own stacks, not configurable by the user.

According to the *OSEK/VDX Operating System, v.2.2.2, 5 July 2004* specification there are no services for manipulation of CPU and / or OS interrupt levels directly. Therefore nested interrupts with the

same hardware level can not occur. For CPU with one hardware level nested interrupts are forbidden. For multilevel modes nested interrupts with different priority are allowed. Application is not allowed to manipulate interrupt enabling bits in the CPU state registers.

WARNING! It is strictly forbidden for application to manipulate directly the registers that control CPU state and interrupts, i.e. INTC registers and MSR.

The warning above also means that the code inside ISR functions shall not write to INTC_EOIR register.

ISRs can communicate with Tasks in the AUTOSAR OS by the following means:

- ISR can activate a task;
- ISR can send / receive messages;
- ISR can increment a counter;
- ISR can get Task ID;
- ISR can get state of the task;
- ISR can set event for a task;
- ISR can get event mask of the task;
- ISR can manipulate alarms;

Interrupts cannot use any OS services except those which are specially allowed to be used within ISRs. When using other services, in the Extended Status of the Operating System the error will be reported. See Table 5.1 on page 66 and "System Services" for details.

ISR Categories

In the AUTOSAR Operating System two types of Interrupt Service Routines are considered.

ISR Category 1

ISRs of this type are executed on the stack of the current runnable . In this case, if the ISR uses the stack space for its execution, the user is responsible for the appropriate stack size. Only 6 Interrupt Management services (enabling / disabling interrupts) are allowed in ISRs of category 1 (see Table 5.1). After the ISR is finished, processing continues exactly at the instruction where the interrupt occurred, i.e. the interrupt has no influence on task management.

The following statements are used to define ISR category 1.

```
ISR( ISR_handler )
{
...
/* the code without any OS service calls */
/* except Suspend/ResumeAllInterrupts    */
...
}
```

NOTE ISR category 1 should have the priority higher than any ISR of category 2. If ISR category 1 is interrupted by ISR category 2, rescheduling may take place at the end of ISR category 2 execution and ISR category 1 execution will be suspended therefore.

ISR Category 2

In ISR category 2 the Operating System provides an automatic switch to the ISR stack (except in BCC1, SC1) and enters OS ISR execution context. After that, any user's code can be executed, including allowed OS calls (to activate a task, send a message or increment a counter). See "Service Protection" for the list of services allowed for ISR. At the end of the ISR, the System automatically switches back to the stack of interrupted Task / ISR and restores context.

The following statements are used to define ISR category 2.

```
ISR( ISR_handler )
{
  /* the code with allowed OS calls */
}
```

Inside the ISR, no rescheduling will take place. Rescheduling may only take place after termination of the ISR of category 2 if a preemptable task has been interrupted.

NOTE Interrupts which can not be disabled, such as NMI or any synchronous exception, shall not be assigned to ISRs of category 2. If OS can not disable ISR category 2, then OS will not be able to protect it's critical code sections and will crash.

Interrupt Level Manipulation

Direct manipulation with the CPU interrupt flags or levels is strictly forbidden. The user can not define values of the interrupt masks directly. Interrupts are enabled during task execution if there are any ISR or any Timer is configured, otherwise OS dispatcher does not control interrupt levels of MCU. Interrupts can be disabled via disable/enable interrupt API functions or by using resource mechanism.

DisableAllInterrupts service can be used to temporary disable all interrupts. To return to previous interrupt status *EnableAllInterrupts* service must be called after it in the body of the same Task or ISR where *DisableAllInterrupts* is called.

SuspendAllInterrupts and *ResumeAllInterrupts* pair has the same effect as *DisableAllInterrupts* - *EnableAllInterrupts* pair but allows nesting of pairs.

SuspendOSInterrupts service can be used to temporary disable all interrupts of category 2. To return to previous interrupt status *ResumeOSInterrupts* service must be called after it in the body of the same Task or ISR where *SuspendOSInterrupts* is called.

Resources can be used to temporary disable interrupts. If a Task or ISR occupies resource which is referenced by ISR of priority ‘P’ then all ISRs with priority equal or lower than ‘P’ are disabled and task scheduling is disabled. Interrupts and task scheduling are reenabled after releasing the resource.

ISR Stack

In AUTOSAR OS classes SC3 and SC4 each ISR of category 2 has its own stack definition, but the ISRs of the same *PRIORITY* shares the stack memory. The ISRs stack size is configured in OIL file.For the ISRs that shares the same stack memory the largest value is taken as the stack size.

The purpose of the ISR stack(s) is to save memory and provide ability to protected ISR stacks memory in SC3, SC4. The separate ISR stack(s) are used in the Freescale AUTOSAR OS. Switching to the ISR stack is performed by the OS at the beginning of ISR category 2. This stack(s) are used only by ISRs of category 2. In SC2 and ECC1 within SC1 if nested interrupt occurs after the stack has been switched, they will continue to use the ISR stack. After completion of ISRs category 2 OS switches back from ISR stack.

In BCC1 class within SC1 ISRs use common single stack and no stack switch performed.

The *interrupt stack frame* usually consists of the CPU registers, and optionally some compiler-dependent ‘virtual’ registers. The CPU registers are pushed onto the stack under hardware or software control. In the latter case the compiler generates a stack frame by means of adding special sequences of machine instructions before the first statement in the function.

NOTE The OS saves only 32-bit registers content on Task switch and when interrupt occurs. 64-bit registers are not supported by the OS.

Most compilers use function modifiers (like ‘interrupt’) to generate stack frames. In turn, the *ISR* keyword, specified in OSEK (see “Conventions”), is a macro for this modifier.

ISRs Timing protection

The timing protection can be applied to ISR category 2 via defining ISR Execution Budget and / or ISR's arraival rate. Also the time that ISR locks Resource or disables OS interrupts via Suspend / ResumeOSInterrupts services may be controlled via Time Protection. See ISR Definition for the parameters of Timing Protection.

Interrupt Dispatcher

Freescale AUTOSAR OS / MPC56xxAM provides interrupt dispatcher for the external interrupt distinction. It is necessary for each external interrupt to define an ISR object in the OIL file and assign the *EXTERNAL* value to the *IrqChannel* attribute, set appropriated value for the *IrqNumber* attribute (For the correspondence between *IrqNumber* value and external interrupt sources see the Hardware Technical References) and assign the *PRIORITY* attribute value. The ISRs functions are called via OS Interrupt Dispatcher as conventional C functions. The ISRs shall not access the Interrupt Controller (INTC) HW in any way.

For more detailed information about MPC interrupt controller see MPC HW Technical Reference(s).

Programming Issues

Run-time Services

AUTOSAR OS provides the set of services for interrupt management. These services are shown in the Table 5.1.

Table 5.1 Interrupt Management Services in AUTOSAR OS

Service Name	Description
DisableAllInterrupts	Disable all interrupts, does not allow nesting

Table 5.1 Interrupt Management Services in AUTOSAR OS

Service Name	Description
EnableAllInterrupts	Restore state of interrupts saved by DisableAllInterrupts service
SuspendAllInterrupts	Disable all interrupts, allows nesting
ResumeAllInterrupts	Restore state of interrupts saved by SuspendAllInterrupts service
SuspendOSInterrupts	Disable interrupts of category 2, allows nesting
ResumeOSInterrupts	Restore state of interrupts saved by SuspendOSInterrupts service

Not all OS services may be used inside ISRs. Please refer to the Table 2.4 on page 39 for the list of allowed services.

Constants

For each ISR defined in oil file the constant with name <IsrName>PRIORITY equal to this ISR *PRIORITY* is defined. This is an Freescale OS extension of the AUTOSAR OS

Conventions

Within the application an Interrupt Service Routine should be defined according to the following pattern:

```
ISR( <ISRName> )[1]
{
...
}
```

The keyword *ISR* is the macro for compiler specific interrupt function modifier, which is used to generate valid code to enter and exit ISR.

It is possible to use one function as handler for several ISRs via ISR attribute *IsrFunction*. The value of the attribute shall exactly

[1] OSEK/VDX does not specifies using of keyword *ISR* for ISRs of category 1, it is Freescale AUTOSAR OS specific for this category.

corresponds to the function name. This function shall be defined as *void <FuncName>(void)*, OS macro '*ISR*' shall not be used in this case.

ISR Definition

To define common ISR parameters like ISR stack size (used in SC1, ECC1) the corresponding OS properties should be specified in the configuration file.

Definition of Interrupt related properties looks like:

```
OS <name> {
  .....
  IsrStackSize = 800;
};
```

Definition of specific ISR object looks like:

```
ISR Handler {
  PRIORITY = 3;
  STACKSIZE = 256;
  CATEGORY = 2;
  RESOURCE = ISRresource;
  MESSAGE = messageA;
  TIMING_PROTECTION = FALSE;
  IrqChannel = EXTERNAL {
    IrqNumber = 146;   /* SCI_A combined interrupt request */
  };
};
```

NOTE PRIORITY of ISRs of category 2 shall be less then PRIORITY of any ISR of category 1. In SC2 class the TP ISR has the priority higher than each ISR of category 2, so ISRs of categore 1 priority shall be higher at least by 2 then of ISRs cat.2 to provide "gap" for TP ISR.

See "ISR Definition" for details.

6

Resource Management

This chapter describes resource management and task coordination by means of resources.

This chapter consists of the following sections:

General

The resource management is used to coordinate concurrent access of several tasks or / and ISRs to shared resources, e.g. management entities (scheduler), program sequences (critical sections), memory or hardware areas. In the Freescale AUTOSAR OS the resource management is provided in all Conformance Classes.[1]

The resource management ensures that

- two modules (tasks or ISRs) cannot occupy the same resource at the same time,
- priority inversion cannot arise while resources are used,
- deadlocks do not occur due to the use of these resources,
- access to resources never results in *waiting* state.

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

- full-preemptable tasks,
- non-preemptable tasks, if the user intends to have the application code executed under other scheduling policies too.

[1] This is the Freescale OS extension of the AUTOSAR OS, which fully supports resources only in BCC2 and ECC conformance classes.

- resource sharing between tasks and / or ISRs.

Resources cannot be occupied by more than one task or ISR at a time. The resource which is now occupied by a task or ISR must be released before another task or ISR can get it. The AUTOSAR operating system ensures that tasks are only switched from *ready* to *running* state if all the resources which might be occupied by that task during its execution have been released. The AUTOSAR operating system ensures that an ISR is enabled if all the resources which might be occupied by that ISR during its execution have been released. Consequently, no situation occurs in which a task or an ISR tries to access an occupied resource. A special mechanism is used by the AUTOSAR Operating System to provide such behavior, see "Priority Ceiling Protocol" for details**.**

In case of multiple resource occupation, the task or ISR must request and release the resources following the LIFO principle (stack). For example, if the task needs to get the communication hardware and then the scheduler to avoid possible preempts, the following code may be used:

```
GetResource( SCI_res );       /* occupy the SCI resource */
...                           /* user's code */
GetResource( RES_SCHEDULER ); /* occupy the scheduler resource */
...                           /* user's code */
ReleaseResource( RES_SCHEDULER ); /* release the scheduler */
ReleaseResource( SCI_res );       /* release the SCI resource */
```

The AUTOSAR OS resource management allows the user to prevent such situations as priority inversion and deadlocks which are the typical problems of common synchronization mechanisms in real-time applications (e.g., semaphores).

It is not allowed to occupy RES_SCHEDULER resource in the ISR.

Access to Resources

Before they can be used, the resources must be defined by the user at the system configuration stage through the *RESOURCE* definition, see "Resource Definition". The resource must be referenced in all TASKs and ISRs which can occupy it. (A special

resource RES_SCHEDULER is referenced and can be used by any TASK by default.) After that the task or ISR can occupy and release the resource using the *GetResource* and *ReleaseResource* services. While the resource is occupied, i.e. while the code between these services is executed, this resource cannot be occupied by another task or ISR.

In the AUTOSAR Operating System the resources are ranked by priority. The priority which is statically assigned to each resource is called *Ceiling Priority*. The resource priority is calculated automatically during the system generation. It is possible to have resources of the same priority, but the Ceiling Priority of the resource is higher or equal to the highest task or ISR priority with access to this resource.

Restrictions when using resources

*TerminateTask, ChainTask, Schedul*e and *WaitEvent* must not be called while a resource is occupied. The interrupt service routine must not be completed with the resource occupied.

AUTOSAR strictly forbids the nested access to the same resource. In the rare cases when the nested access is required, it is recommended to use a second resource with the same behaviour as the first resource. The OIL language especially supports the definition of resources with identical behaviour (so-called ‘linked resources’).

Priority Ceiling Protocol

The Priority Ceiling Protocol is implemented in the AUTOSAR Operating System as a resource management discipline.

The priority ceiling protocol elevates the task or ISR requesting a resource to a resource priority level. This priority can be simply calculated during the system generation. As shown in “General” the Ceiling Priority is:

- Higher or equal to the highest task or ISR priority with access to this resource (task T1);
- Lower than the priority of those tasks or ISR which priority is higher than the one of task T1.

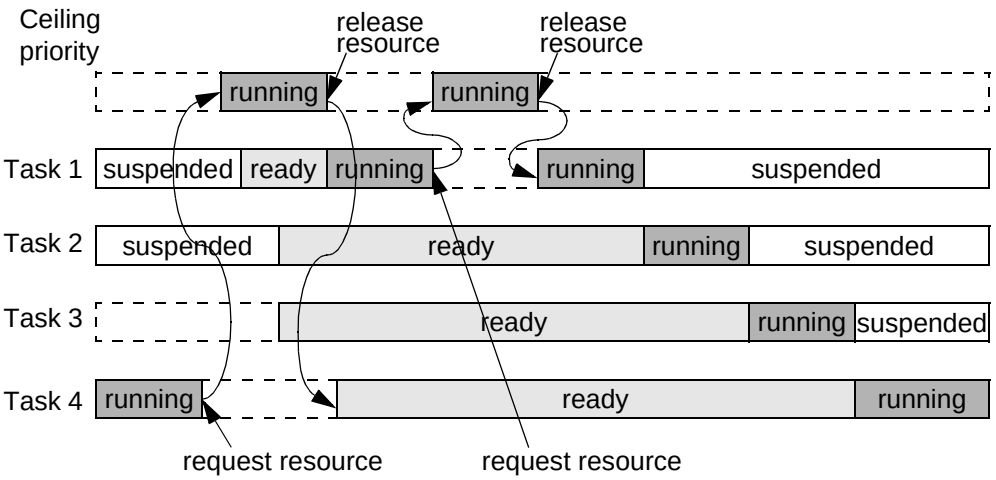
Note that all ISR priorities are higher than any task and scheduler priorities.

When a task or ISR occupies a resource, the system temporarily changes its priority. It is automatically set to the Ceiling Priority by the resource management. Any other task or ISR which might occupy the same resource does not enter *running* state (ISR stays pending and cannot start) due to its lower or equal priority. If the resource occupied by the task or ISR is released, the task (ISR) returns to its former priority level. Other tasks which might occupy this resource can now enter *running* state (ISR can start).

Hardware interrupt levels are used by resources which can be occupied in the ISR. When such a resource is occupied by a task or ISR, the interrupts of the corresponding priority are disabled and the AUTOSAR OS scheduler is switched off. Therefore, the *running* task can not be switched to *ready* state while such a resource is occupied. Releasing the resource causes enabling interrupts of the corresponding level and switching on the AUTOSAR OS scheduler.

The example shown on Figure 6.1 illustrates the mechanism of the Priority Ceiling Protocol.

Figure 6.1 Priority Ceiling Protocol



In the figure above Task 1 has the highest priority and Task 4 has the lowest priority. The resource has a priority greater than or equal

to the Task 1 priority. When Task 4 occupies the resource, it gets a priority not less than the Task 1 has, therefore it cannot be preempted by *ready* Task 1 until it releases the resource. As soon as 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 turn, occupies the resource, its priority is also changed to the Ceiling Priority.

Scheduler as a Resource

The AUTOSAR operating system treats the scheduler as a specific resource which is accessible to all tasks. Therefore, a standard resource with the predefined identifier *RES_SCHEDULER* is generated, and it is supported in all Conformance Classes. If a task calls the services *GetResource* or *ReleaseResource* with this identifier as a parameter, the task will occupy or release the scheduler in the manner of a simple resource. See the code example in "General".

If a task wants to protect itself against preemptions by all other tasks, it can occupy the scheduler exclusively. When it is occupied, interrupts are received and processed normally. However, it prevents the tasks rescheduling.

NOTE If a task gets the scheduler as a resource, it must release it before the point of rescheduling!

Reference to RES_SCHEDULER from TASK object is optional. The user may define the resource with the name RES_SCHEDULER in the OIL file but this resource will have the maximal (scheduler) prioriry regardless of which tasks have the reference to it. The RES_SCHEDULER is referenced and can be used from any task by default. The RES_SCHEDULER resource cannot be occupied from the ISR.

Resource timing protection

Timing protection can be applied to resorce locking time, i.e. the time from call to GetResource(resource) to ReleaseResource(resource) in the Task / ISR on per Task / ISR basis. The *RESOURCELOCKTIME* is defined inside the Task or ISR

definition. Timing protection can not be applied to the *Internal Resource*.

Internal resources

The internal resources are the resources which are not visible to the user and therefore they can not be addressed by the system functions GetResource and ReleaseResource. Instead, they are managed strictly internally within a clearly defined set of the system functions. Besides, the behaviour of the internal resources is exactly the same as the behaviour of standard resources (the priority ceiling protocol etc.). At most one internal resource can be assigned to a task during the system generation. If an internal resource is assigned to a task, the internal resource is managed as follows:

- The resource is automatically taken when the task enters running state, except when it has already taken the resource. As a result, the priority of the task is automatically changed to the ceiling priority of the internal resource.
- At the points of rescheduling, defined in chapter "Scheduling Policy", the resource is automatically released.

The tasks which have the same internal resource assigned form a group of tasks. The tasks within a certain group behave like the non preemptable tasks - they can not preempt each other; while the tasks with the priority higher than the highest priority within the group preempt the tasks within the group.

The non preemptable tasks may be considered as a special group with an internal resource of the same priority as the RES_SCHEDULER priority (chapter "Non-preemptive Scheduling"). The internal resources can be used in all cases when it is necessary to avoid unwanted rescheduling within a group of tasks. More than one internal resource can be defined in a system thus defining more then one group of tasks.

The general restriction on some system calls that they must not be called with the resources occupied (see "Restrictions when using resources") is not applied to the internal resources, as the internal resources are handled within those calls.

The tasks which have the same assigned internal resource cover a certain range of priorities. It is possible to have the tasks which do not use this internal resource in the same priority range, but these tasks will not belong to the group. So they are preemptable by the tasks of the group.

Programming Issues

Configuration Option

The following system configuration option is intended to decrease the amount of RAM and ROM used by the OS:

- *USERESSCHEDULER*
  If this option is set to FALSE, *RES_SCHEDULER* is not supported by the OS.

Data Types

The AUTOSAR OS determines the following data type for the resource management:

- *ResourceType*
  The abstract data type for referencing a resource.

The only data type must be used for operations with resources.

Run-time Services

The AUTOSAR OS provides a set of services for the user to manage resources. Detailed descriptions of these services are provided in "Resource Management Services". Below is only a brief list.

Table 6.1 Resource Management Run-time Services

Service Name	Description
GetResource	This call serves to occupy the resource (to enter critical section of the code, assigned to the resource)
ReleaseResource	Releases the resource assigned to the critical section (to leave the critical section)

Resource Definition

To define a resource, the following definition statement should be specified in the application configuration file:

```
RESOURCE ResourceName {
   RESOURCEPROPERTY = STANDARD;
   };
```

For more details see "Resource Definition".

7

Counters and Alarms

This chapter describes usage of these control mechanisms in AUTOSAR OS.

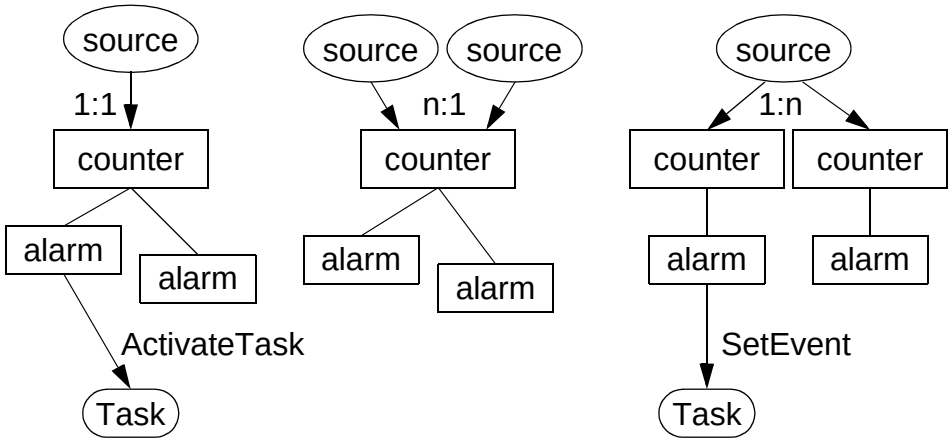
This chapter consists of the following sections:

General

The AUTOSAR operating system comprises a two level concept to make use of recurring occasions like periodic interrupt of timers, interrupt of the sensors on rotating angles, or any recurring software occasions. To manage such situations, counters and alarms are provided by the Freescale AUTOSAR OS. Additionally AUTOSAR OS provides a Schedule Table mechanism for fast tasks activations in accordance with statically defined schedule. The recurring occasions (sources) can be registered by counters. Based on counters, the OS offers an alarm mechanism to the application software. Counters and alarms are provided by the AUTOSAR OS in all Conformance and Scalability Classes.

Freescale AUTOSAR OS provides two types of counters: software (SW) counters and hardware (HW) counters. HW counters allow more precise timing while decreases system overhead time because the timer interrupts are occurs only when the alarm(s) attached to the counter expiries.

Figure 7.1 Counters and Alarms



Counters

Any occasion in the system can be linked with a counter, so that when the occasion has 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.

The AUTOSAR OS defines 2 types of counters:

- Hardware Counter - a counter that is advanced by hardware (e.g. timer). The counter value is “in hardware”.
- Software Counter - a counter which may be incremented by the *IncrementCounter*() API call and keeps it’s value in the variable.

The counter is represented by a current counter value and some counter specific parameters: *counter initial value*, *conversion constant* and *maximum allowed counter value*. They are defined by the user. The last two parameters are constants and they are defined at system generation time. The counter initial value is the dynamic parameter. The user can initialize the counter with this value and thereafter on task or on interrupt level advance it using the system service *IncrementCounter*.

The HW counters may use only *System* and *Second* Timers and has a maximum allowed value defined by number of bits in the timer HW. Only eMIOS and STM timers with output compare registers

may be used for HW counters. The source(s) for HW counters is hardware itself. For HW counters one tick of hardware timer is equivalent to a *Period* for SW counter, thus enabling more precise timing while keeping system overhead on interrupt processing low because timer interrupts are raised only when alarms attached to this counter are expired.

The maximum allowed counter value specifies the number after which the counter rolls over. After a counter reaches its maximum allowed possible value (or rolls over the predefined size - 32 bits), it starts counting again from zero.

The conversion constant *TICKSPERBASE* can be used to convert the counter value into an appropriate user specific unit of measurement, e.g. seconds for timers, angular degrees for rotating axles. The conversion is done by the user's code and the parameter can be treated as a counter-specific reference value, it is not used by the OS for any purposes.

The operating system provides the service *GetCounterInfo* to read these counter specific configuration values. Also the service *GetCounterValue* is provided to read the current counter value.

Freescale AUTOSAR OS provides two timers (the internal system clocks): the *System Timer* and the *Second Timer*. The timers attributies are not defined in AUTOSAR OS specifications, this is Freescale OS extension of the AUTOSAR OS. User can turn on or turn off the system timer using the *SysTimer* attribute and the second timer using the *SecondTimer* attribute. The timer can be assigned to a standard counter with the following additions:

- special constants are defined to describe counter parameters and to decrease access time;
- the user defines the source of hardware interrupts for the counter attached to the timer.

In the system definition statement for the system (second) timer the user should define one of possible hardware interrupt sources. Parameters to tune the hardware can be also defined by the user in this statement. This possibility allows the user to exactly tune the system (see "Power Architecture Platform-Specific Features" for details).

While hardware related parameters are defined, the code to initialize the system (second) timer hardware and the interrupt handler are automatically provided for the user as a part of the OS. The handler is an ISR category 2 but it is not needed to define the ISR in OIL file. In that case the user does not have to care about handling of this interrupt and the provided code can not be changed.

Software Counters may be triggered from user defined ISR(s). Hardware interrupts which are used to trigger counters have to be handled in usual manner. To perform any actions with the counter the application software processing the occasion should call the *IncrementCounter* system service.

The user is free to assign one source exactly to one counter (1:1 relationship), several sources to one counter (n:1 relationship), or one source to several counters (1:n relationship), see Figure 7.1. Meaning that it is possible to advance the same counter in different software routines.

Alarms

The alarm management is built on top of the counter management. The alarm management allows the user to link task activation or event setting or a call to callback function to a certain counter value. These *alarms* can be defined as either single (one-shoot) or cyclic alarms.

The AUTOSAR OS allows the user to set alarms (relative or absolute), cancel alarms and read information out of alarms by means of system services. Alarm is referenced via its symbolic name which is assigned to the alarm statically at the configuration stage.

Examples of possible alarm usage are:

- ‘Activate a certain task, after the counter has been advanced 60 times’, or
- ‘Set a certain event, after the counter has reached a value of 90’.

The counter addressed in the first example might be derived from a timer which is advanced every second. The task in the example is

then activated every minute. The counter addressed in the second example might be derived from a rotating axle. The event is set on a 90 degree angle.

The AUTOSAR OS takes care of the necessary actions of managing alarms when a counter is advanced.

Counters and Alarms are defined statically. The assignment of alarms to counters, as well as the action to be performed when an alarm expires (task and event), are also defined statically. An application can use an alarm after it has been defined and assigned to a counter. Alarms may be either in the stop state or running state. To run an alarm, the special system services are used, which set dynamic alarm parameters to start it.

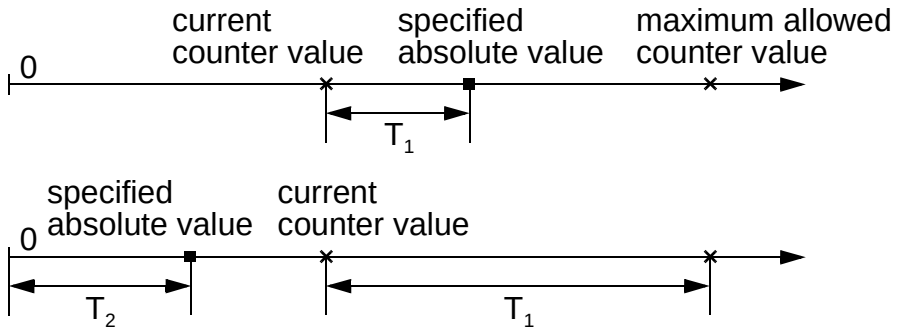
Dynamic alarm parameters are:

- the counter value when an alarm has to expire.
- the cycle value for cyclic alarms.

An alarm can be started at any moment by means of system services *SetAbsAlarm* or *SetRelAlarm*. An alarm will expire (and predefined actions will take place) when a specified counter value is reached. This counter value can be defined relative to the actual counter value or as an absolute value. The difference between relative and absolute alarms is the following:

- Relative alarm expires when the specified number of counter ticks has elapsed, starting from the current counter value at the moment the alarm was set.
- Absolute alarm expires when the counter reaches the specified number of ticks, starting from zero counter value no matter which value the counter had at the moment the alarm was set. If the specified number of ticks is less than the current counter value, the counter will roll over and count until the specified value. If the specified value is greater than the current value, the alarm will expire just after the counter reaches the desired number. This is illustrated by Figure 7.2. In the latter case, the total time until the alarm expires is the sum of T_1 and T_2.

Figure 7.2 Two Cases for the Absolute Alarm



If a cycle value is specified for the alarm, it is logged on again immediately after expiry with this relative value. Specified actions (task activation or event setting) will occur when the counter counts this number of ticks, starting from the current value. This behavior of the cyclic alarm is the same both for relative and absolute alarms. If the cycle value is not specified (it equals zero) the alarm is considered as a single one.

WARNING! It is not recomended to use values of cycle and/or increment parameters close to 0xFFFF (hardware counter *MAXALLOWEDVALUE*) for Alarms configured on a Hardware Timer. The difference between *MAXALLOWEDVALUE* and this values should be greater than interrupt latency of the system (time spent in the longest ISR).

Alarm Callback

The user can define an alarm callback function for each alarm. The function is placed in the user application and its name is added to the ALARM object definition as value of *ALARMCALLBACKNAME* attribute.The alarm callback is the usual user's function. It can have neither parameter(s) nor return value.

Alarm CallBack is not allowed in Scalability Classes SC2 .. SC4

Only the *SuspendAllInterrupts* and *ResumeAllInterrupts* services may be used within alarm callback.

The callback function shall have next definition:

```
ALARMCALLBACK(CallbackName)
{
  /* user application code */
}
```

Schedule Table

AUTOSAR OS provides a special feature for fast tasks activations in accordance with statically defined schedule named *Schedule Table*. User can define in a configuration file a sequence of tasks and / or events to be cyclically or one-time activated at predefined time points. Only one *Task* may be activated (or *Event* set) for each *Action* of *Schedule Table* but the *Offset* of several *Action*s may be set to the same value to achieve a simultaneous activation of two or more tasks.

Freescale AUTOSAR OS does not limits the number of *Schedule Tables* assigned to the *COUNTER* and running simultaneously.

Schedule Table may be syncronized to external time source via *SyncScheduleTable* OS service in accordance with statically configured *SYNCSTRATEGY* - *EXPLICIT* or *IMPLICIT*. Freescale AUTOSAR OS / MPC56xxAM supports Global Time Synchronization in all Scalability Classes.

Programming Issues

Configuration Options

The following system configuration options affect the counter and alarm management:

- *SysTimer*
  If this option is turned on the System Timer is used.
- *SecondTimer*
  If this option is turned on the Second Timer is used.

Data Types

The following data types are established by AUTOSAR OS to operate with counters:

- *CounterType*
  The data type references a counter
- *TickType*
  The data type represents a counter value in system ticks.
- *TickRefType*
  The data type references data corresponding to the data type *TickType*. Reference to *TickType* variable can be used instead of *TickRefType* variable.
- *CtrInfoType*
  This data type represents a structure for storage of counter characteristics. This structure has the following fields:
  - *maxallowedvalue*
    maximum possible allowed count value;
  - *ticksperbase*
    number of ticks required to reach a counter-specific significant unit (it is a user constant, OS does not use it);
  - *mincycle*
    minimum allowed number of ticks for a cyclic alarm (only for system with Extended Status).

    All fields have the data type *TickType*. The following code may illustrate usage of this data type:

```
CtrInfoType CntData;
TickType maxV, minC, cons;
GetCounterInfo( CntID, &CntData );
maxV = CntData.maxallowedvalue;
minC = CntData.ticksperbase;
cons = CntData.mincycle;
```

- *CtrInfoRefType*
  This data type references data corresponding to the data type *CtrInfoType*. Reference to *CtrInfoType* variable can be used instead of *CtrInfoRefType* variable

The following data typs are established by AUTOSAR OS to operate with alarms:

- *AlarmBaseType*
  This data type represents a structure for storage of alarm characteristics. It is the same as *CtrInfoType*;
- *AlarmBaseRefType*
  This data type references data corresponding to the data type *AlarmBaseType*;
- *AlarmType*
  The data type represents an alarm element.

Counters and Alarm Generation

To generate a counter in an application, the *COUNTER* definition is used, it looks like the following:

```
COUNTER CounterName {
  MINCYCLE = 5;
  MAXALLOWEDVALUE = 1000;
  TICKSPERBASE = 10;
};
```

To define system and second timer hardware-specific parameters, the following properties may / should be defined in the OS definition statement:

```
OS <name> {
  ...
  SysTimer = HWCOUNTER {
    COUNTER = <CounterName>;
    ISRPRIORITY = <priority>;
    TimerHardware = <TypeOfTimer> {
      Prescaler = OS {
        Value = <PrescalerValue>;
      };

    };
  };
  SecondTimer = SWCOUNTER {
    COUNTER = <CounterName>;
    ISRPRIORITY = <priority>;
    TimerHardware = <TypeOfTimer> {
      Prescaler = OS {
        Value = <PrescalerValue>;
```

```
      };
      TimerModuloValue = <TimerModuloValue>;
    };
  };
  ...
};
```

The system timer hardware parameters are described in detail in the section "CPU Related Attributes".

To generate an alarm in an application, the *ALARM* definition is used, it looks like the following:

```
ALARM AlarmName {
  COUNTER = CounterName;
  AUTOSTART = FALSE;
  ACTION = SETEVENT {
    TASK = TaskName;
    EVENT = EventName;
  };
};
```

Detailed counter and alarm generation statements are described in "Counter Definition" and "Alarm Definition".

To generate a ScheduleTable in Application the SCHEDULETABLE definition is used, it looks like the following:

```
SCHEDULETABLE ScheduleTableName {
  COUNTER = CounterName;
  AUTOSTART = FALSE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION =FALSE;
  PERIODIC = TRUE;
  LENGTH = <integer>;
  ACTION = SETEVENT{
      OFFSET = integer;
      TASK = TaskName;
      EVENT = EventName;
  };
};
```

Detailed ScheduleTable generation statement is described in "Schedule Table definition".

Run-time Services

AUTOSAR OS grants a set of services for the user to manage counters and alarms. Detailed descriptions of these services are provided in "System Services". Here only a brief list is given.

Table 7.1 Counter and Alarm Management Run-time Services

Service Name	Description
InitCounter	Sets the initial value of the counter
IncrementCounter	Increments the counter value and process attached alarms
GetCounterValue	Gets the counter current value
GetCounterInfo	Gets counter parameters
SetRelAlarm	Sets the alarm with a relative start value
SetAbsAlarm	Sets the alarm with an absolute start value
CancelAlarm	Cancels the alarm: the alarm is transferred into the STOP state
GetAlarm	Gets the time left before the alarm expires
GetAlarmBase	Gets alarm base (attached counter) parameters
StartScheduleTableRel	Starts Schedule Table processing at relative time offset
StartScheduleTableAbs	Starts Schedule Table processing at absolute time
StopScheduleTable	Stops Schedule Table processing
NextScheduleTable	Plans start of another Schedule Table
SyncScheduleTable	Synchronizes Schedule Table to external time
SetScheduleTableAsync	Sets Schedule Table status to asynchronous
GetScheduleTableStatus	Returns Schedule Table status

NOTE InitCounter, GetCounterValue and GetCounterInfo services are not defined in *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. This is Freescale OS extension of the AUTOSAR OS.

Constants

For all counters, the following constants are defined:

- *OSMAXALLOWEDVALUE_cname*
  Maximum possible allowed value of counter <cname> in ticks.
- *OSTICKSPERBASE_cname*
  Number of ticks required to reach a specific unit of counter <cname> (it is a user constant, OS does not use it).
- *OSMINCYCLE_cname*
  Minimum allowed number of ticks for a cyclic alarm of counter <cname>. This constant is not defined in *STANDARD* status

For the counters attached to *SysTimer* and *SecondTimer* special constants are provided by the operating system:

- *OSMAXALLOWEDVALUE*
  maximum possible allowed value of the system timer in ticks;
- *OSMAXALLOWEDVALUE2*
  maximum possible allowed value of the second timer in ticks;
- *OSTICKSPERBASE*
  number of ticks that are required to reach a counter-specific value in the system counter (it is a user constant, OS does not use it);
- *OSTICKSPERBASE2*
  number of ticks that are required to reach a counter-specific value in the second counter (it is a user constant, OS does not use it);
- *OSTICKDURATION*
  duration of a tick of the system counter in nanoseconds;
- *OSTICKDURATION2*
  duration of a tick of the second counter in nanoseconds;
- *OSMINCYCLE*
  minimum allowed number of ticks for a cyclic alarm attached to the system counter (only for system with Extended Status);
- *OSMINCYCLE2*
  minimum allowed number of ticks for a cyclic alarm attached to the second counter (only for system with Extended Status).

NOTE *OSMAXALLOWEDVALUE2* , *OSTICKSPERBASE2*, *OSTICKDURATION2*, and *OSMINCYCLE2* constants are not

defined in the OSEK/VDX Operating System specification. These constants are Freescale OS extension of the AUTOSAR OS.

8

Events

This chapter is devoted to event management and task coordination by means of events.

This chapter consists of the following sections:

- General
- Events and Scheduling
- Programming Issues

General

Within the AUTOSAR operating system, tasks and ISRs can be synchronized via occupation of a resource (see "Resource Management"). Another means of synchronization is the event mechanism, which is provided for Extended Tasks only. Events are the only mechanism allowing a task to enter the *waiting* state.

An *event* is a synchronization object managed by the AUTOSAR Operating System. The interpretation of the event is up to the user. Examples are: the signalling of a timer's expiry, the availability of data, the receipt of a message, etc.

Within the operating system, events are not independent objects, but allocated to Extended Tasks. Each event is represented by a bit in event masks which belong to Extended Tasks. Maximum number of events for Extended Task is 32. Each Extended Task has the mask of a "set" events and the mask of events the task is waiting for ("wait" mask). When the Extended Task is activated all its events are cleared.

An Extended Task can wait for several events simultaneously and setting at least one of them causes the task to be transferred into the *ready* state. When a task wants to wait for one event or several ones, the corresponding bits in its "'wait" event mask are set by the system service *WaitEvent* which is designed to force a task to wait

for an event. When another task sets an event, it sets the specified bits of the “set” event mask and if some bits in both “wait” and “set” masks are the same the task is transferred into the *ready* state. The task can clear its own events in the “set” event mask using *ClearEvent* service.

All tasks can set any events of any Extended Task. Only the appropriate Extended Task (the owner of the particular event mask) is able to clear events and to wait for the setting (receipt) of events. Basic Tasks must not use the operating system services for clearing events or waiting for them.

An alarm can also be set for an Extended Task, which in turn sets an event at a certain time. Thus, the Extended Task can delay itself.

It is not possible for an interrupt service routine or a Basic Task to wait for an event, since the receiver of an event is an Extended Task in any case. On the other hand, any task or ISR can set an event for an Extended Task, and thus inform the appropriate Extended Task about any status change via this event.

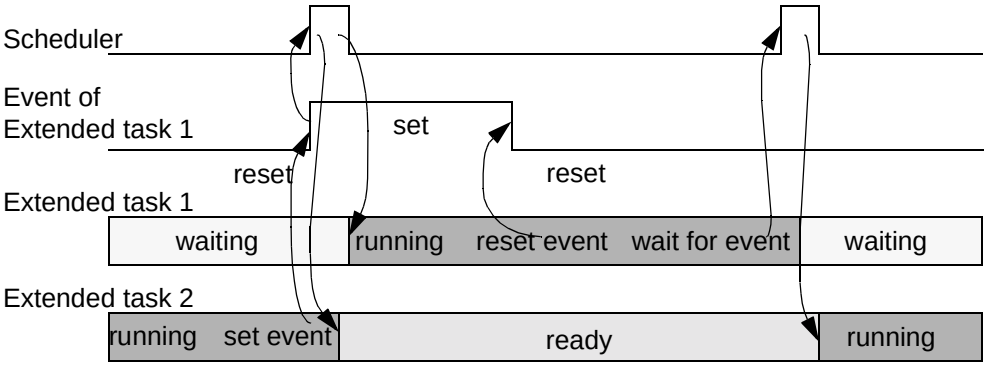
Events and Scheduling

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 Task is in the *waiting* state if an event for which the task is waiting is not set. If an Extended Task tries to wait for an event and this event has already been set, the task remains in the *running* state.

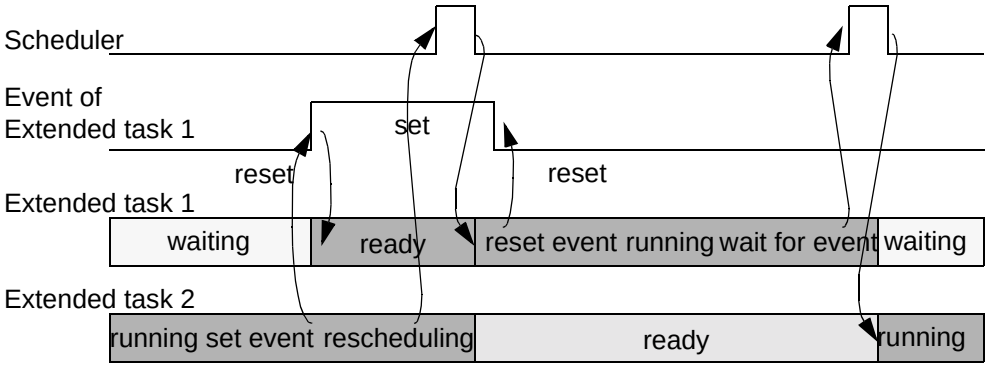
Figure 8.1 illustrates the procedures which are effected by setting an event: Extended Task 1 (with higher priority) waits for an event. Extended Task 2 sets this event for Extended Task 1. The scheduler is activated. Subsequently, Task 1 is transferred from the *waiting* state into the *ready* state. 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 8.1 Synchronization by Events for Full-preemptive Scheduling



If non-preemptive scheduling is supposed, rescheduling does not take place immediately after the event has been set, as shown on Figure 8.2.

Figure 8.2 Synchronization by Events for Non-preemptive Scheduling



Programming Issues

Configuration Options

There are no any system configuration options controlling event management in the system.

Data Types

The AUTOSAR Operating System establishes the following data types for the event management:

- *EventMaskType*
  The data type of the event mask;
- *EventMaskRefType*
  The data type to refer to an event mask. Reference to *EventMaskType* variable can be used instead of *EventMaskRefType* variable.

The only data types must be used for operations with events.

Events Definition

To generate an event in an application the *EVENT* definition is used, it looks like the following:

```
EVENT EventName {
  MASK = 0x01;
};
```

Some task events which used by the task as internal flags can be undefined in the OIL file. But it is strictly recommended to define all events and reference them in TASK object. Missing of an event in OIL file can lead to wrong mask assignment. If task has no references to events the task is considered to be *Basic* one.

Run-time Services

AUTOSAR OS grants a set of services for the user to manage events. A detailed description of these services is provided in "Event Management Services". Here only a brief list is given.

Table 8.1 Event Management Run-time Services

Service Name	Description
SetEvent	Sets events of the given task according to the event mask
ClearEvent	Clears events of the calling task according to the event mask

Table 8.1 Event Management Run-time Services

Service Name	Description
GetEvent	Gets the current event setting of the given task
WaitEvent	Transfers the calling task into the waiting state until specified events are set

Examples of the run-time services usage are provided in section "Event Management Services".

9

Communication

OSEK COM is not part of AUTOSAR OS specification. It is supported in the Freescale AUTOSAR OS for compatibility with OSEKturbo implementations.

This chapter describes message concept of OSEK COM and its usage in Freescale AUTOSAR OS.

This chapter consists of the following sections:

Message Concept

AUTOSAR concept defines communication mechanism as part of the AUTOSAR RTE. Freescale AUTOSAR OS implements the internal communications for the case when there is no RTE available, i.e. the OS is used as “stand-alone”.

Communication concept and message management system services are implemented according to *OSEK/VDX Communication, v.3.0.3, 20 July 2004* specification. Freescale AUTOSAR OS supports CCCB (Communication Conformance Class B) which includes support of internal *Unqueued* and *Queued Messages.*

The message may be defined as sending or receiving - *Queued* or *Unqueued* - using attribute *MESSAGEPROPERTY*. Several receiving messages can be configured for one sending message.

An Unqueued Message represents the current value of a system variable, e.g. engine temperature, wheel speed, etc. Unqueued Messages are not buffered but overwritten with their actual values.

The receive operation reads the Unqueued Message value. Thereby the message data is not consumed.

By contrast, Queued Messages are stored in a FIFO buffer and they are read by the application in the order they arrived. A particular Queued Message can therefore be read only once, as the read operation removes it from the queue. A Queued Message would normally be used to track changes of state within a system, where it is important that receiver maintains synchronisation of state information with the sender.

In OSEK COM message objects are referenced by tasks and ISRs via the message identifiers defined by the user at the configuration stage.

The Operating System ensures data consistency of message data during task operation, uniform in all types of scheduling. The received unqueued message data remains unchanged until a further send operation is performed, unless the task or function using the data overwrites the data with a *SendMessage* or *InitMessage* services.

As an option, *task activation*, *event signalling*, *flag* or *callback* function can be defined statically to be performed at message arrival to notify a task. Task activation or event signalling can be used to inform tasks that shall act immediately on new message information. There is no special operating system service to wait for messages, but the normal event mechanism is used. Only one notification method can be assigned for certain receiving message. COM *callback* method is allowed only for SC1 Conformance Class.

OSEK COM communication services provide all means for internal message transfers. To transfer data over the network, the external OSEK Communication System (COM) shall be used (not included in Freescale AUTOSAR OS), which is designed to handle all other types of communication through the network. OSEK COM provides the same API for external messages (with some extentions) as for internal messages. Thus, messages serve as interface for both internal and network communication. Uniform services with identical interfaces are offered (network transparency).

Unqueued Messages

Unqueued Messages represent the current value of a state variable. A message can be accessed only via OS services. Message data is accessed indirectly through the application message copy. Access through the message copy guarantees consistency of information in the message between adjacent operations of send / receive. The send operation overwrites the current value of a message. The receive operation reads the current value of an Unqueued Message whereby the message data is not consumed. The user can place copies of messages into the global or into the local memory of the functions.

The *SendMessage* and *ReceiveMessage* services ensure that the consistent writing and reading of message data within the send and receive operation (also in preemptive systems).

The *InitMessage* service allows setting of initial value of the *Unqueued* message. An other way to set initial value is to specify it in message configuration via attribute *INITIALVALUE*, but the range of this attribute is restricted to UINT64.

1:N communication for Unqueued Messages does not have any difference from 1:1 communication, since any task can read the Unqueued Messages if its identifier is known.

Queued Messages

Queued Messages are stored in a FIFO buffer and they are read by the application in the order they arrive.

The send operation adds a message data to the end of the message queue. The receive operation reads the first value of a Queued Message in the queue and then removes it from the queue. The user can place copies of messages into the global memory or into the local memory of the functions.

The *SendMessage* and *ReceiveMessage* services ensure that the consistent writing and reading of message data within the send and receive operation (also in preemptive systems).

When a Queued Message is received the information from the first message in the message queue is copied to the message copy and further work with the message information is performed using this copy.

The *InitMessage* service resets the number of messages in queue to 0, it does not initialize Queued message with data.

Data Consistency

Data consistency means that the content of a given application message correlates unambiguously to the operations performed onto the message by the application. This means that no unforeseen sequence of operations may alter the contents of the application message. Thus data consistency means that it can be guaranteed that a task can complete the calculation with the same data set. Data consistency is guaranteed using access with copy or using external synchronization mechanisms, e.g. events.

Programming Issues

Configuration Options

The following message related options are defined inside COM object:

- *COMERRORHOOK*
  The attribute specifies if the COM ErrorHook mechanism is used. It is recomended to set this attribute to the same value as OS attribute *ERRORHOOK*
- *COMUSEGETSERVICEID*
  The attribute defines whether the macro to get service ID is provided for COMErrorHook. It is recomended to set this attribute to the same value as OS attribute *USEGETSERVICEID*
- *COMUSEPARAMETERACCESS*
  The attribute defines whether macros to get the first parameter of the service is provided for COMErrorHook. It is recomended to set this attribute to the same value as OS attribute *USEPARAMETRACCESS*

- *COMSTARTCOMEXTENSION*
  The attribute specifies if the COM startup extention mechanism is used If this attribute is set TRUE then StartCOM service calls the function StartCOMExtension which shall be provided by user.

Data Types

The OSEK COM determines the following data types to operate with messages:

- *MessageIdentifier* - type for OSEK COM message object identifier.
- *ApplicationDataRef* - type for reference to the application message data:
- *FlagValue* - type for the current state of a message flag.
- *COMApplicationModeType* - type for COM application mode.
- *COMShutdownModeType* - type for COM shutdown mode.
- *COMServiceIdType* - type for identifier of an OSEK COM service.

Message Definition

Each message in an application is generated by means of using statements like the following:

- for sending message:

```
MESSAGE MsgA {
     MESSAGEPROPERTY = SEND_STATIC_INTERNAL{
       CDATATYPE = "MSGATYPE";
     };
};
```

- for receiving message:

```
MESSAGE MsgA_rcv {
     MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL{
       SENDINGMESSAGE = MsgA;
     };
     NOTIFICATION = ACTIVATETASK {
       TASK = TASK_A;
     };
};
```

In detail message configuration statements is described in "Message Definition".

There are no constructional elements defined for messages.

Run-time Services

Freescale AUTOSAR OS OS grants a set of services for the user to manage messages. Detailed descriptions of these services are provided in section "Communication Management Services". Here only a brief list is presented.

Table 9.1 Communication Management Run-time Services

Service Name	Description
SendMessage	Updates the message
ReceiveMessage	Gets the message
GetMessageStatus	Gets message status
ReadFlag	Returns the value of the specified notification flag
ResetFlag	Clears the specified notification flag
InitMessage	Initializes message data to given value
StartCOM	Initializes COM internal data
StopCOM	Stops communications
GetCOMApplicationMode	Returns current COM application mode

In Freescale AUTOSAR OS behaviour of all COM messages, except *GetCOMApplicationMode*, is identical before and after call to *StartCOM*.

Callback Function

The user can define callback function for each message. The function is placed in the user application and its name added to MESSAGE object definition as value of *CALLBACKROUTINENAME* attribute, in this case *ACTION* shall be defined as *COMCALLBACK*. The function is called when message arrives.

The COM CallBack function(s) is allowed only for SC1.

The callback function is usual user's function. It is executed on OS or ISR level and only COM services, except StartCOM and StopCOM, are allowed in it.

The callback function shall have next definition:

```
void <CallBackName> (void)
{
  /* user code */
}
```

Usage of Messages

Messages are identified via a symbolic name. This identifier is used for references to the message when the system service is used.

The variables to hold message's copies must be defined within the user's code by means of using the regular C-language definitions.

For example, if the user defines the message *MsgA* having type *int,* and *MsgB* as some receiving message, then user's code may access message using the following statements:

```
int _MsgA;

ReceiveMessage( MsgA_rcv, &_MsgA );
if( _MsgA == 2 ) { _MsgA = 1; }
SendMessage( MsgA, &_MsgA );
```

10

Error Handling and Special Routines

This chapter describes support provided to the user to debug an application and handle errors.

This chapter consists of the following sections:

General

The AUTOSAR Operating System provides the user with tools for error handling and simple debugging at run time. These are special hook routines with names specified by AUTOSAR OS that are to be written by the user. In this section, error handling at the system configuration stage is not considered; it is described in "System Objects Definition".

Hook Routines

The AUTOSAR Operating System supports system specific *hook routines* to allow user-defined actions within the OS internal processing.

These hook routines in AUTOSAR OS are:

- Called by the operating system, in a special context depending on the implementation of the operating system
- Can not be preempted by tasks
- not interrupted by category 2 interrupt routines
- Part of the operating system, but user defined
- Implemented by the user
- Standardized in interface by AUTOSAR OS, but not standardized in functionality (environment and behavior of the hook routine itself), therefore usually hook routines are not portable
- Only allowed to use a subset of API functions
- Optional

In the AUTOSAR OS hook routines are intended for:

- System startup. The corresponding hook routine(s) (*StartupHook*) is called after the operating system startup and before the scheduler is running. Each OS-Application can has its own routine (*StartupHook_AppName*) in addition to System one.
- Tracing or application dependent debugging purposes as well as user defined extensions of the context switch
- Error handling. The corresponding hook routine (*ErrorHook*) is called if a system call returns a value not equal to *E_OK*. Each OS-Application can has its own routine (*errorHook_AppName*) in addition to System one.
- System shutdown. The corresponding hook routine(s) (*ShutdownHook*) is called. Each OS-Application can has its own routine (*ShutdownHook_AppName*) in addition to System one.
- AUTOSAR OS defines *ProtectionHook* hook routine

The Freescale AUTOSAR OS OS provides the following hook routines:
ErrorHook, *PreTask*Hook, *PostTaskHook*, *StartupHook*, *ShutdownHook* and *ProtectionHook*. The user must create the code of these routines, the OS only provides description of function prototypes.

- *ErrorHook* – this hook is called by the Operating System at the end of a system service which has a return value not equal to *E_OK* (see "Error Interface"). It is called before returning from the service. It is also called when an alarm expires and an error is detected during task activation or event setting. If the applicatio

specific ErrorHooks are defined they are called after the OS ErrorHook.

- *PreTaskHook* – this hook is called before the operating system enters the Task, but in the Task context. This hook is called from the scheduler when it passes control to the given task. It may be used by an application to trace the sequences and timing of the tasks' execution.
- *PostTaskHook* – This hook is called after the operating system leaves the Task, it is called in the Task context. It is called from the scheduler when it switches from the current task to another. It may be used by an application to trace the sequences and timing of tasks' execution.
- *PreIsrHook* – this hook is called before the operating system enters the ISR of category 2 or System Timer ISR. This hook is called in the context of the ISR, but with all interrupts disabled. It may be used for debugging purposes only. This hook is not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.
- *PostIsrHook* – This hook is called after the operating system leaves the ISR of category 2 or System Timer ISR. This hook is called in the context of the ISR, but with all interrupts disabled. It may be used for debugging purposes only. This hook is not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.
- *StartupHook* – This hook is called after the operating system startup and internal structures initialization and before initializing System Timer and running scheduler. It may be used by an application to perform initialization actions and task activation. If the applicatio specific StartupHooks are defined they are called after the OS StartupHook.
- *ShutdownHook* – This hook is called when the service *ShutdownOS* has been called. It is called before the Operating System shuts down itself. If the applicatio specific ShutdownHooks are defined they are called before the OS ShutdownHook.
- *ProtectionHook* – This hook is called by the OS in case of memory or timing protection violation. It is also called in case of stack overflow in SC3, SC4 classes. Three special variables are available for reading inside ProtectionHook to find what has caused the violation:
  - OsViolatorAppId - contains the Application ID that caused violation or INVALID_OSAPPLICATION.

- OsViolatorTaskId - contains the ID of the Task that caused violation or INVALID_TASK
- OsViolatorISRId - contains the ID of the ISR that caused violation or INVALID_ISR

This 3 variables are not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS

The Freescale AUTOSAR OS places error hooks code into the memory section ".ostext".

Error Handling

Error Interface

The hook routine *ErrorHook* is provided to handle possible errors detected by the AUTOSAR Operating System. Its basic framework is predefined and must be completed by the user. This gives the user a choice of efficient centralized or decentralized error handling.

ErrorHook is called before the OS service returns error code not equal E_OK and from the OS kernel itself in case of wrong *Task* or *ISR* termination - for example the *Task* ends without call to *TerminateTask* or *ISR* does not release resource. *ErrorHook* is also called by the OS when the *ALARM* action can not be performed.

The special system routine *ShutdownOS* is intended to shut down the system. *ShutdownOS* may be called by the user on experiencing a fatal error. User hook *ShutdownHook,* if configured, is called by *ShutdownOS.* The Application-specific shutdown hooks, if configured, are called by the OS before the OS *ShutdownHook.*

NOTE Call to *ShutdownOS* from non-Trusted Application is ignored.

The AUTOSAR OS *ErrorHook* is called with a parameter that specifies the error. It is up to the user to decide what to do, depending on the error has occurred. If the Application-specific error hooks are configured, they are called after the OS *ErrorHook.* According to AUTOSAR OS specification, if system service is called from *ErrorHook* user's hook and this service does not return E_OK

error code, then *ErrorHook* is not called. Therefore nested *ErrorHook* calls are blocked by AUTOSAR OS.

The Freescale AUTOSAR OS specifies the following error codes:

Table 10.1 Freescale AUTOSAR OS Error Codes

Name	Value	Type
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_SERVICEID	9	Service can not be called
E_OS_RATE	10	Task activation rate exceeds limit
E_OS_ILLEGAL_ADDRESS	11	An invalid address is given as a parameter to a service
E_OS_MISSINGEND	12	Tasks terminates without a TerminateTask() or ChainTask() call
E_OS_DISABLEDINT	13	OS service is called inside an interrupt disable/enable pair
E_OS_STACKFAULT	14	Stack fault detected via stack monitoring by the OS
E_OS_PROTECTION_MEMORY	15	Memory access violation occurred
E_OS_PROTECTION_TIME	16	Task/Category 2 ISR exceeds its execution time budget
E_OS_PROTECTION_LOCKED	17	Task/Category 2 ISR exceeds Resource or ISRs lock time
E_OS_PROTECTION_EXCEPTION	18	Trap occurred
E_OS_PROTECTION_ARRIVAL	19	Task/Category 2 ISR arrives before its timeframe has expired
E_OS_SYS_FATAL	21	Fatal error in the OS code
E_OS_SYS_ORDER	23	Incorrect order of function call sequence
COM specific error codes		
E_COM_ID	35	Invalid message name passed as parameter
E_COM_LIMIT	36	Overflow of FIFO associated with queued messages
E_COM_NOMSG	41	No message available

Errors committed by the user in direct conjunction with the Operating System can be intercepted to a large extent via the Extended Status of the Operating System, and displayed. This results in an extended plausibility check on calling OS services.

Macros for ErrorHook

The special macros are provided by OS to access the ID of the service that caused an error and it's first parameter (only if it is an object ID) inside *ErrorHook* routine:

- the macro *OSErrorGetServiceId*() returns the service identifier where the error has been risen. The service identifier is of type *OSServiceIdType*. Possible values are *OSServiceId_xxxx*, where xxxx is the name of the system service, if the error occured not inside a service function but detected by the OS kernel then the special value *OSServiceId_NoService*[1] is returned.
- the macros of type <*OSError_serviceID_parameterID*()>, where serviceID is the name of the service and parameterID is the name of the first parameter, returns the value of the first parameter. For example, *OSError_ActivateTask_TaskID*() returns the task identifier if *ErrorHook* was called because of error detected in ActivateTask.

In order use the *OSErrorGetServiceId* macro, the user has to set the *USEGETSERVICEID* attribute to TRUE. To operate with macros for parameter access, the *USEPARAMETERACCESS* attribute shall be set to TRUE. The list of service identifiers and macros for parameter access are provided in *Quick Reference* section of the User Manual.

Extended Status

The AUTOSAR Operating System version with `Extended Status` requires more execution time and memory space than the release version due to the additional plausibility checks it offers. However, many errors can be found in a test phase. After they have all been eliminated, the system can be recompiled with `Standard Status` to the release version.

[1] it is Freescale OS extension of the AUTOSAR OS, not specified in AUTOSAR OS v.3.0 specification

AUTOSAR OS specifies that in Scalability Classes SC3 and SC4 only extended OS Status is used.

The following example can illustrate Extended Status usage:

- If a task is activated in the release version, only 'OK' is returned. In the Extended Status version, the additional status like 'Task not defined', 'Task already activated' can be returned. These extended messages must no longer occur in the target application at the time of execution, i.e., the corresponding errors are not intercepted in the operating system's release version.

Possible Error Reasons

Errors in the application software are typically caused by:

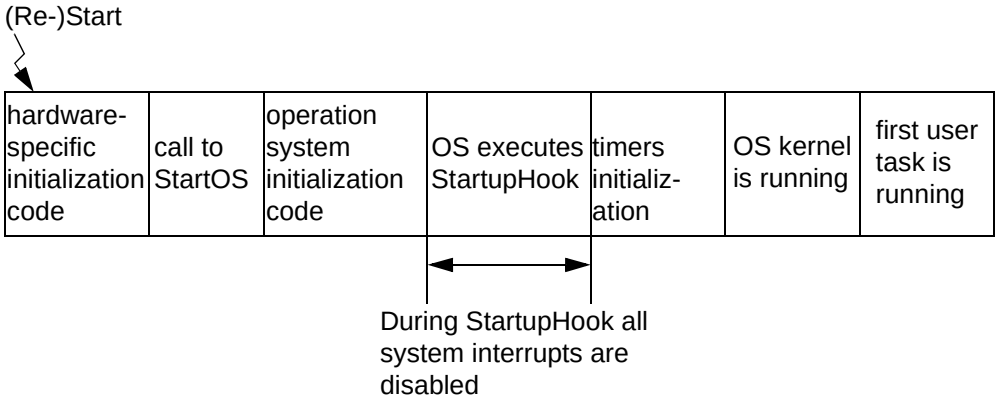
- Errors on handling the operating system, i.e. incorrect configuration / initialization / dimensioning of the operating system or non-observance of restrictions regarding the OS service.
- Error in software design, i.e. unwise choice of task priorities, generation of deadlocks, unprotected critical sections, incorrect dimensioning of time, inefficient conceptual design of task organization, etc.

Start-up Routine

The special system routine *StartOS* is implemented in the AUTOSAR Operating System to allocate and initialize all dynamic system and application resources in RAM. This routine is called from the main() function of the application with the application mode as parameter (see "Application Modes") and pass the control to the scheduler to schedule the first task to be running. User hook *StartupHook* is called after operating system startup and before the system and second timers initialization and running scheduler. See "Sample Application" for details.

The figure below shows system startup.

Figure 10.1 System Startup in the AUTOSAR OS



Initializing system and second timers after *StartupHook* avoids loss of timer interrupts during *StartupHook* execution.

If for any of timers the attribute *Prescaler* is set to *USER* then it is the User responsibility to initialize this timer hardware.

Application Modes

Application modes are intended to allow AUTOSAR OS to come under different modes of operation. The application modes differ in the set of autostarted Tasks and Alarms. Once the operating system has been started, it is not possible to change the application mode.

Freescale AUTOSAR OS supports up to 8 application modes.

System Shutdown

The special *ShutdownOS* service exists in AUTOSAR OS to shut down the operating system. This service could be requested by an application or requested by the operating system due to a fatal error.

When *ShutdownOS* service is called with a defined error code, the operating system will shut down and call the hook routine *ShutdownHook*. The user is free to define any system behavior in *ShutdownHook* e.g. not to return from the routine. If *ShutdownHook*

returns, the operating system enters endless loop with all interrupts disabled.

It is possible to restart OS by calling setjmp() before calling *StartOS*() and then calling longjmp() in *ShutdownHook*.

Programming Issues

Configuration Options

The following configuration options affect error handling and hook routines:

- *SCALABILITYCLASS*
  Specifies the Scalability Class. If AUTO, the Scalability class is defined according to the Application and Task definitions.
- *ERRORHOOK*
  If this option is turned on, the *ErrorHook* is called by the system for error handling
- USEGETSERVICEID

  If this option is turned on, it allows to get the service ID inside ErrorHook.
- USEPARAMETERACCESS

  If this option is turned on, it allows to get the first service parameter value inside ErrorHook.
- *PRETASKHOOK*
  If this option is turned on, the *PreTaskHook* is called by the system before context switching
- *POSTTASKHOOK*
  If this option is turned on, the *PostTaskHook* is called by the system before context switching
- *STARTUPHOOK*
  If this option is turned on, the *StartupHook* is called by the system at the end of the system initialization
- *SHUTDOWNHOOK*
  If this option is turned on, the *ShutdownHook* is called by the system when the OS service *ShutdownOS* has been called

- *PROTECTIONHOOK*
  If this option is turned on, the *ProtectionHook* is called in case of memory or timing violation.

11

System Configuration

This chapter describes possible AUTOSAR OS versions, configuration options and the configuration mechanism.

This chapter consists of the following sections:

General

The AUTOSAR Operating System is fully statically configured . All system properties, the number of system objects and their parameters (characteristics of tasks, counters, alarms, messages, etc.), run time behavior are defined by the user. Such approach allows the user to create various range of applications with exactly defined characteristics. Different memory and performance requirements can be easily satisfied with such modular approach.

The AUTOSAR specifies usage of XML for configuration, but for the OS the OIL file also may be used. It provides compatibility with OSEK / VDX standard. OIL and XML OS configuration files contains the same information, but in different format.

All application parameters are defined in the special configuration file. This file must conform OIL grammar rules. It is processed by the separate *System Generator utility (SG)*. SG is written in Java and requires the Java run-time environment to be installed on the developer machine. The System Generator analyzes statements in the configuration file and builds output C-language files needed to compile OS source files and to compile and link an application with the specified features. During its execution SG reports to the user about the errors. The System Generator produces header and source code files that defines all properties and objects in terms of the C

language. These files are to be compiled and linked together with the user's and OS source code.

Application Configuration File

Application configuration file contains the statements which define the system properties and objects. Such file can have any extension and the extension ".oil" is suggested by default. The file of this format is processed by the SG utility. The extension ".oin" is suggested for the included files. Freescale AUTOSAR OS provides the special utility for conversion of the XML files, generated by General Configuration Editor, to OIL format.

As the result of application configuration file processing SG produces three types of standard C-language files as it is described in "Application Configuration" and optional ORTI file as it described in "Using OS Extended Status for Debugging". SG produces two header files and one source file. These files provides the code for all system tables, descriptors, arrays etc. both in ROM and RAM according to the user specified application configuration.

OIL Concept

OSEK Implementation Language (OIL) is the specially designed language for development of embedded applications based on OSEK and AUTOSAR concept. OIL is used to describe the application structure (application configuration) as a set of system objects with defined links. OIL allows the user to write an application configuration as a text file. These files have predefined structure and special (standard) grammar rules.

All system objects specified by AUTOSAR and relationships between them can be described using OIL. OIL defines standard types for system objects. Each object is described by a set of attributes and references.

All keywords, attributes, object names, and other identifiers are case-sensitive.

OIL File

The OIL file contains two parts – one for the definition of implementation specific features (Implementation Definition) and another one for the definition of the structure of the application located on the particular CPU (Application Definition).

In the very beginning of an OIL file the number of the version of OIL is indicated. The keyword OIL_VERSION is used for this purpose. For example:

```
OIL_VERSION = "3.0";
```

OIL Format

The Standard OIL is intended to configure AUTOSAR OS Operating System (OS). It is strictly defined by the *OSEK/VDX System Generation OIL: OSEK Implementation Language, v.2.5, 01 July 2004* specification with additions defined in *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS).*

Implementation Definition

The Implementation Definition defines implementation specific features for the particular AUTOSAR OS implementation for which this application is developed.

The implementation can limit the given set of values for object attributes (e.g. restrict the possible OS conformance classes).

It is not allowed to exclude any standard attributes from the particular AUTOSAR OS implementation. Additional non-standard attributes can be defined for the objects for the particular AUTOSAR OS implementation.

The include mechanism (see “Include Directive”) can be used to define the implementation definition as a separate file. Thus corresponding implementation definition files are developed and delivered with particular AUTOSAR OS implementations and then included in user's OIL files. The Freescale AUTOSAR OS / MPC56xxAM implementation is described in the osmpc56xxAM.oin

file which is delivered in the package. Extention ".oin" is used for OIL files that should be used only as include files.

Implementation Definition Grammar

Implementation Definition part starts with keyword *IMPLEMENTATION* and implementation name.

The structure for Implementation Definition part is shown using the following syntax:

```
IMPLEMENTATION <name> {
    <object_descriptions>
};
```

All objects within Implementation Definition part are described using the same syntax.

```
<object_type> {
    <property_definitions>
};
```

Object type is defined by the object keyword. For Freescale AUTOSAR OS / MPC56xxAM implementation the following object types are implemented:

```
OS, APPMODE, APPLICATION, TASK, ISR, RESOURCE, EVENT, COUNTER,
ALARM, SCHEDULETABLE, MESSAGE, COM, NM
```

The set of object properties are to be defined within the object description. Both implementation specific attribute and reference shall be defined before it is used.

The attribute definition has the following structure:

```
<attr_type> [ WITH_AUTO ] [ <attr_range> ]
<attr_name> [ = <default_value> ] [
<multiple_specifier> ];
```

The attribute type and attribute value range (if it exists) shall be defined. The range of attribute values can be defined in two ways: either the minimum and maximum allowed attribute values are

defined (the [0..12] style) or the list of possible attribute values are presented (like C enumeration type).

The WITH_AUTO specifier can be combined with any type. If WITH_AUTO can be specified it means that this attribute can have the value AUTO and the possibility of automatic assignment.

Data types defined for OIL are listed below. Note that these data types are not necessarily the same as the corresponding C data types.

- UINT32 - any unsigned integer number (possibly restricted to a range of numbers. This data type allows to express any 32 bit value in the range of $[0..(2^{32}-1)]$.
- INT32 - any signed integer number in the range of $[-2^{31}..(2^{31}-1)]$.
- UINT64 - any unsigned integer number in the range $[0..(2^{64}-1)]$.
- INT64 - any signed integer number in the range $[-2^{63}..(2^{63}-1)]$.
- FLOAT - any floating point number according to IEEE-754 standard (Range: + / - 1,176E-38 to + / -3,402E+38).
- ENUM – the list of possible values shall be presented. Any value from the list can be assigned to an attribute of this type. ENUM types can be parameterized, i.e. the particular enumerators can have parameters. The parameter specification is denoted in curly braces after the enumerator. Any kind of attribute type is allowed as parameter of an enumerator.
- BOOLEAN – The attribute of this type can have either TRUE or FALSE value. BOOLEAN types can be parameterized, i.e. the particular boolean values can have parameters. Parameter specification are denoted in curly braces after an explicit enumeration of the boolean values. Any kind of attribute type is allowed as parameter of a boolean value.
- STRING – Any 8-bit character sequence enclosed in double-quotes, but not containing double-quotes can be assigned to this attribute.

A reference is a special type of value intended to define links between system objects. The reference definition has the following structure:

```
<object_type> <reference_name> [
<multiple_specifier> ];
```

The reference type is taken from the referenced object (e.g. a reference to a task shall use the TASK_TYPE keyword as reference type). A reference can “point to” any system object.

Multiple reference is the possibility to refer to several objects of the same type with one OIL statement. For example the task can refer to several events. If the reference shall be defined as a “multiple” reference then the '[]' brackets shall be present after the reference name.

An attribute can have a subattributes which are described in curly brackets.

Application Definition

In an application definition the AUTOSAR OS application is composed from a set of system objects. In general an application can contain more than one system object of the same type.

Since an application is performed on the CPU the entity called *CPU* is introduced as the top of the description. This entity encompasses all local objects such as tasks, messages, etc. Therefore, *CPU* can be considered as a container for application objects. This concept is introduced to provide future OIL evolution towards to distributed system support. This entity is identified by the keyword CPU.

Object Definition

All objects are described using the same syntax.

```
<object_type> <object_name> {
    <property_definitions>
};
```

Objects are labeled by keywords which shall be written in the upper case. Object attributes and references are also labeled by the keywords. The keywords are introduced in “System Objects Definition”. After an object keyword the object name must follow. Name is combined from any symbols up to 32 symbols long.

A set of attributes and references belonging to an object is enclosed in curly brackets, like in C language.

All assignments are made via the '=' operator. Each statement ends with semicolon - ';' like in the C language. A reference is represented as a reference type keyword assigned with a name of the object referenced. If multiple reference pointed to the set of objects several references shall be used. Here is the example for task referencing to own events:

```
EVENT = MyEvent1;
EVENT = MyEvent2;
```

Include Directive

The preprocessing directive to include other OIL files is allowed in any place of the OIL file. The Freescale AUTOSAR OS OIL files that is intended to be included into the user files has an extention ".oin". Include statement has the same syntax as in ANSI-C:

```
#include <filename.oin>
#include "[path]filename.oin"
```

The file name can be optionally preceded by a directory specification. The quoted form means that a header file is being looked for in the current directory first, then along the path specified by the "/I" command-line option, then along paths specified by the special environment variable. The angle-bracket form means that a header file is being looked for first along the path specified by the "/I" command-line option, then along paths specified by the special environment variable.

Comments

An OIL file may contain comments. The '/*' and '//' characters define the start of a comment. Any characters after '//' are considered as a comment and the end of line (EOL) terminates the comment. Any characters after '/*' are considered as comments and the end of the comment is defined by '*/'. Nested comments are not allowed in OIL.

File Structure

Any file in the Standard OIL format describes an application for a single CPU and, in general, must have the following structure:

```
OIL_VERSION = <version>;
IMPLEMENTATION <name> {// Implementation definition
  <OBJECT_TYPE> {
    ...list of implementation specific object attributes...
  };
  ...
};

CPU <name> {          // Definition of the application on CPU
  <OBJECT_TYPE> <object_name>
  {                   // System object definition
    <ATTRIBUTE> = <value>;
    <REFERENCE> = <object_name>;
    ... list of object attributes and references ...
  };
  ... list of objects ...
};
```

Configuration File Rules

The application configuration files must conform some simple rules to be successfully processed. The rules are:

- All objects are described using the OIL syntax;.
- Each object must have a unique name. Each object may be divided into several parts;
- All object names are accessible from the application;
- An attribute defines some object properties (for example, the task priority). Attributes that take a single value may only be specified once per object. Attributes that take a list of values must be specified as multiple statements;
- An object can have a set of references to other system objects. Per object there may be more than one reference to the same type of object;
- Values must be defined for all standard attributes of all objects, except for multiple attributes which can be empty;
- All statements must be written without errors;

- It is recommended to avoid conflicting statements (e.g., in SC1 the STACKSIZE is defined and no EVENT(s) is defined for a task) since it leads to error or warning messages.

12

System Objects Definition

This chapter describes objects that are controlled by the Operating System - tasks, resources, alarms, messages, counters, ISRs and even the OS itself - are considered as system objects.

This chapter consists of the following sections:

General

All objects that are controlled by the Operating System – tasks, resources, alarms, messages, counters, ISRs and even the OS itself - are considered as system objects. Each of them has its unique characteristics defined by the user. To specify parameters for each system object the special statements are used for each object. All statements are described below in detail.

Each group of attributes has the scheme which describes attributes nesting. Possible attribute values are placed in angle brackets and

separated by slash. Default value is marked as bold text. If default value is present, the attribute can be omitted. Scheme includes all attributes, but some of them can be used if parent attribute has the determined value only. The description of these dependencies is included into attribute definitions below.

OIL attributes names mapping to XML configuration

The AUTOSAR prescribes usage of XML notation for all BSW modules, including the OS. In each OIL attribute description the correspondent XPath name is provided in square brackets.

OS Definition

The *Operating System* is the mandatory object for any application. This object is used to define OS configuration parameters. The keyword OS is used for this object type. Only one OS object can be defined in the application. See "Operating System Architecture" for more detailed information about OS. The syntax of the OS object definition is as follows:

```
OS <name>{
       <attributes>
    };
```

OS object's attributes can be divided into the groups which correspond to appropriate system objects and their interaction. Nested structure of OS object definition is displayed on the syntax schemes of each attribute group.

Different groups of related attributes are described below. Brief explanations are provided. All these attributes should be defined inside the scope of the OS object.

Global System Attributes

This group of attributes represents system features which are common for the whole system. The attributes should be defined inside the scope of the OS object in accordance with the following syntax (default attribute value are shown in bold type):

```
STATUS = <STANDARD / EXTENDED>;
SCALABILITYCLASS = <SC1 / SC2 / SC3 / SC4 / AUTO>;
CC = <BCC1 / ECC1 / AUTO>;
DEBUG_LEVEL = <0 / 1 / 2 / 3 / 4>;
FastTerminate = <TRUE / FALSE>;
USERESSCHEDULER = <TRUE / FALSE>;
```

- ***STATUS*** (ENUM)
  [OsOS/OsStatus]
  The attribute specifies the debugging ability of OS, defines whether additional run-time error checks are supported by OS or not. If the system has the *Extended Status* additional checks are performed to detect run-time errors. The *Extended Status* adds approximately 20% of code, and increases timing accordingly. This mode is considered to be a debugging mode for SC1,2 classes without Memory Protection. In the *Standard Status* OS performs only very limited set of checks, the performance is increased and the amount of consumed memory is decreased. For AUTOSAR OS Scalability Classes 3 and 4 *Extended* Status is mandatory.

  As a general approach, it is recommended to start application development with the *extended status* to discover configuration and run-time problems. For debugged applications the *status* may be set to *standard* to reduce code size and improve timing.

- ***SCALABILITYCLASS*** (ENUM or AUTO)
  [OsOS/OsScalabilityClass]
  The attribute specifies the Scalability Class which is supported by the OS. Class *SC1* is the base class, classes *SC2* and *SC4* provides Timing Protection while *SC3* and *SC4* provides Memory Protection.

  If the value is *AUTO* then scalability class is defined according to APPLICATIONs, TASKs and ISRs definitions.

- ***CC*** (ENUM or AUTO)
  [OsOS/OsCC]
  This Freescale AUTOSAR OS specific attribute specifies the conformance class which is supported by the OS. However all features of the OS may be selected by means of using other OS additional properties. Therefore, even for given conformance class the functionality may be reduced or increased according to user's needs.

If the value is *AUTO* then conformance class is defined according to TASKs definitions.

- ***DEBUG_LEVEL*** (ENUM)
  [OsOS/OsDebugLevel]
  This Freescale AUTOSAR OS specific attribute specifies the ORTI support in OS. If the system has the DEBUG_LEVEL = 0, ORTI is not supported. If the attribute is set to 1 or higher, the static ORTI mode is turned on. If the attribute is set to 2 or 4, static and run time ORTI (running Task, ISR and system calls identification) modes will be used. When DEBUG_LEVEL = 3 the static ORTI and running Task and ISR identification are supported (system calls tracing is not supported). The value 4 is retained for compatibility with previous versions of OSEKturbo. See "Debugging Application" for details.
- ***FastTerminate*** (BOOLEAN)
  [OsOS/OsFastTerminate]
  This Freescale AUTOSAR OS specific attribute specifies whether the fast versions of *Terminate* / *ChainTask* are used in BCC1 class. This attribute may be set to TRUE only for SC1
- ***USERESSCHEDULER*** (BOOLEAN)
  [OsOS/OsUseResScheduler]
  This attribute specifies whether *RES_SCHEDULER* should be supported or not.

Memory Related Attributes

This group of attributes is Freescale OS extension of the AUTOSAR OS and describes the phisical memory available on ECU and the memory used by the OS itself. See also "OS-Application Definition," for Application memory definition.

NOTE If at least one of memory attributes is set to *TRUE* then the linker file shall be defined as a SysGen input

This attributes may be ommited in SC1, SC2.

The attributes are defined in the scope of the *TargetMCU* object in accordance with the following syntax:

```
InternalROM = <TRUE / FALSE> {
    Address = <integer / 0x00000000>;
```

```
    Size    = <integer / 0x200000>;
};
InternalRAM = <TRUE / FALSE> {
    Address = <integer / 0x40000000>;
    Size    = <integer / 0x10000[1]>;
};
ExternalROM = <TRUE / FALSE> {
    Address = <integer>;
    Size    = <integer>;
};
ExternalRAM = <TRUE / FALSE> {
    Address = <integer>;
    Size    = <integer>;
};
```

- ***InternalROM*** (BOOLEAN)
  [OsOS/OsTargetMC/OsInternalROM]
  This Freescale AUTOSAR OS specific attribute specifies is internal MCU ROM is used by the OS or user application.
- ***InternalRAM*** (BOOLEAN)
  [OsOS/OsTargetMC/OsInternalRAM]
  This Freescale AUTOSAR OS specific attribute specifies is internal MCU RAM is used by the OS or user application.
- ***ExternalROM*** (BOOLEAN)
  [OsOS/OsTargetMC/OsExternalROM]
  This Freescale AUTOSAR OS specific attribute specifies is external ROM is used by the OS or user application.
- ***ExternalRAM*** (BOOLEAN)
  [OsOS/OsTargetMC/OsExternalRAM]
  This Freescale AUTOSAR OS specific attribute specifies is external RAM is used by the OS or user application.
- ***Address*** (UINT32)
  [OsOS/OsTargetMC/Os{Int|Ext}ernal{ROM|RAM}/OsAddress]
  This Freescale AUTOSAR OS specific attribute specifies the address of memory area.
- ***Size*** (UINT32)
  [OsOS/OsTargetMC/Os{Int|Ext}ernal{ROMRAM}/Size]

1. default value depends on TargetMCU

This Freescale AUTOSAR OS specific attribute specifies the size of memory area in bytes.

CPU Related Attributes

This group of attributes provides with possibility to tune the selected hardware. The attributes should be defined inside the scope of the OS object in accordance with the following syntax (default attribute value are shown in bold type):

```
TargetMCU = < MPC5674F / MPC5644A / MPC5634M> {
     ClockFrequency = <integer / 8000>;
     TPTimer = <TRUE / FALSE> {
       Period = <integer / AUTO>;
       TimerHardware = STM {
         Prescaler = <USER / OS> {
           Value = <integer / AUTO>;
         };
       };
     };
     SysTimer = <HWCOUNTER / SWCOUNTER / NONE> {
       COUNTER = <name of COUNTER>;
       ISRPRIORITY = <integer>;
       Period = <integer / AUTO>;
       TimerHardware = <name of hardware timer> {
         Prescaler = <USER / OS> {
           Value = <integer / AUTO>;
         };
         GlobalMIOSPrescaler = <USER / OS> {
         Channel = <integer>;
         TimerModuloValue = <integer / AUTO>;
         PeripheralClockDivider = <integer / 1>;
       };
     };
     SecondTimer = <HWCOUNTER / SWCOUNTER / NONE> {
       COUNTER = <name of COUNTER>;
       ISRPRIORITY = <integer>;
       Period = <integer / AUTO>;
       TimerHardware = <name of hardware timer> {
         Prescaler = <USER / OS> {
           Value = <integer / AUTO>;
         };
```

```
            GlobalMIOSPrescaler = <USER / OS> {
            Channel = <integer>;
            TimerModuloValue = <integer / AUTO>;
            PeripheralClockDivider = <integer / 1>;
            Freeze = TRUE / FALSE;
          };
        };
};
```

- ***TargetMCU*** (ENUM)
  [OsOS/OsTargetMCU]
  This Freescale AUTOSAR OS specific attribute defines the target for which the OS will be configured.
- ***ClockFrequency*** (UINT32)
  [OsOS/OsTargetMCU//OsClockFrequency]
  This Freescale AUTOSAR OS specific attribute specifies oscillator frequency in kHz for calculating prescaler value and timer modulo value. This attribute shall be defined if any of *Period*, *Prescaler/Value* or / and any *TimerModuloValue* is *AUTO*. Otherwise this attribute is used to calculate OSTICKDURATION.
- ***ClockDivider*** (UINT32)
  [OsOS/OsTargetMCU//OsClockDivider]
  This Freescale AUTOSAR OS specific attribute specifies PLL divider for calculating input timer frequency. It is the user responsibility to initialize hardware to appropriate values. Value of the attribute shall be equal to actual divider, not the value written into the HW register.
- ***ClockMultiplier*** (UINT32)
  [OsOS/OsTargetMCU//OsClockMultiplier]
  This Freescale AUTOSAR OS specific attribute specifies PLL multiplier for calculating input timer frequency. It is the user responsibility to initialize PLL hardware with appropriate values. Value of the attribute shall be equal to actual multiplier, not the value written into the HW register.
  In general, the value of (*ClockFrequency* * *ClockMultiplier* / *ClockDivider*) shall be equal to actual frequency of system clock.
- ***TPTimer*** (ENUM)
  [OsOS/OsTargetMCU//OsTPTimer]
  This Freescale AUTOSAR OS specific attribute specifies the Time Protection timer hardware. It shall be defined only for SC2 and SC4.

TP Timer uses STM hardware.

The ISR priority for TP Timer is calculated by SysGen so that it is greater by one then the highest priority of all ISRs of category 2, including System and Second Timers.

- ***SysTimer*** (ENUM)
  [OsOS/OsTargetMCU//OsSysTimer]
  This Freescale AUTOSAR OS specific attribute specifies whether the internal OS system timer is used or not. If *SysTimer* is set to SWCOUNTER or HWCOUNTER, interrupt services are turned on automatically.

  If this attribute is SWCOUNTER or HWCOUNTER, then specific subattributes can be defined for the system timer.

- ***SecondTimer*** (ENUM)
  [OsOS/OsTargetMCU//OsSecondTimer]
  This Freescale AUTOSAR OS specific attribute specifies whether the internal OS second timer is used or not.

  If this attribute is SWCOUNTER or HWCOUNTER, then specific subattributes can be defined for the second timer. The *SecondTimer* attribute and its subattributes can be defined only if *SysTimer* value is equal to SWCOUNTER or HWCOUNTER. The *SecondTimer* attribute can not be set to HWCOUNTER if *SysTimer* value is not HWCOUNTER.

- ***COUNTER*** (reference)
  [OsCounter]
  The reference specifies the *COUNTER* which shall be attached to the system or second timer. This attribute shall be defined for system timer and for second timer if respectively *SysTimer* value and *SecondTimer* value is SWCOUNTER or HWCOUNTER. The same counter can not be attached to both System and Second timers.

- ***ISRPRIORITY*** (UINT32)
  [.../{HWCOUNTER | SWCOUNTER}/OsIsrPriority]
  The attribute specifies priority of the system timer (second timer) interrupt . This attribute shall be defined inside the scope of the *SysTimer* and *SecondTimer* attributes if these attributes are set to SWCOUNTER or HWCOUNTER. The rules defined for *PRIORITY* attribute of ISR object are also applicable for this parameter. The value of the attribute is to be set in the range [1..15].

- ***Period*** (UINT32)
  [.../{HWCOUNTER | SWCOUNTER}/OsPeriod]
  This Freescale AUTOSAR OS specific attribute specifies period of a tick of the system (second) counter in nanoseconds. This

attribute can be defined inside the scope of the *SysTimer* and *SecondTimer* attributes if these attributes are set SWCOUNTER or HWCOUNTER.

The *Period* attribute shall be defined if the corresponded *Prescaler/Valu*e or / and *TimerModuloValue* is AUTO. This attribute is ignored if corresponded *Prescaler/Valu*e and *TimerModuloValue* are not AUTO.

NOTE *OSTICKDURATION* and *OSTICKDURATION2* constants are calculated from the *SysTimer/Period* value or *SecondTimer/Period* value respectively if any of corresponding *Prescaler/Value* and *TimerModuloValue* is AUTO. Otherwise *OSTICKDURATION* and *OSTICKDURATION2* are calculated from the values of the corresponding *Prescaler/Value* and *TimerModuloValue* attributes.

TimerHardware (ENUM)
[.../{HWCOUNTER | SWCOUNTER}/OsTimerHardware]
This Freescale AUTOSAR OS specific attribute is intended to select the hardware interrupt source for the system counter or the second counters (among the accessible MCU devices). This attribute inside the *SysTimer* and *SecondTimer* attributes can be defined if the values of these attributes are HWCOUNTER or SWCOUNTER only. The possible values for SWCOUNTER are PIT, eMIOS and STM (if Timing Protection is not used); the possible value for HWCOUNTER are eMIOS and STM. In SC2,4 STM is reserved for TP timer. The *TimerHardware* attributes in *SysTimer* and *SecondTimer* blocks can have the same value if *Channel* value is different. See "Timer Hardware" for details about possible meanings of these parameters.

- ***Channel*** (UINT32)
[.../{STM | eMIOS | PIT}/OsChannel]
The attribute specifies timer channel number. The attribute shall be defined inside the scope of the *SysTimer* and *SecondTimer* attributes.If the *TimerHardware* attribute is set to *eMIOS* then the *Channel* attribute allowed values are for
  - MPC5634M: 0, 8, 9, 10, 12, 14, 15, 23;
  - MPC5674F: 0..31;
  - MPC5644A: 0, 7..12, 14..23.

  For *STM* valid values of *Channel* are 0..3 for MPC5634M and MPC5644A; 0..1 for MPC5674F. For PIT valid values are

[0..3]. If the *TimerHardware* attributes of the *SysTimer* and *SecondTimer* are the same then the *Channel* values shall be different.

- ***GlobalMIOSPrescaler*** (ENUM)
  [.../{HWCOUNTER | SWCOUNTER}/OsTimerHardware/eMIOS/OsGlobalMIOSPrescaler]
  This Freescale AUTOSAR OS specific attribute specifies whether Global eMIOS prescaler value shall be initialized during OS startup or Global prescaler will be set by the user application.This attribute may be defined for the *Sys(Second)Time*r if *TimerHardware* is eMIOS. If the same timer module is used for both System and Second Timers then these attributes shall have the same value or it may be ommited for *SecondTimer*.

  If the *Prescaler* attribute in the *Sys(Second)Time*r attribute scope has the *USER* value then the corresponding *GlobalMIOSPrescaler* inside of this scope shall have the same value (*USER*).

- ***Prescaler*** (ENUM)
  [.../{eMIOS |STM}/OsPrescaler]
  This Freescale AUTOSAR OS specific attribute specifies whether prescaler value shall be initialized during OS startup or prescaler will be set by the user application. This attribute shall be defined if the SysTimer (SecondTimer) value is HWCOUNTER only.

  The *Prescaler* attribute in the *SysTimer* attribute scope and the *Prescaler* attribute inside the *SecondTimer* attribute shall have the same value (USER or OS).

- ***Value*** (UINT32 or AUTO) – inside *Prescaler* attribute.
  [.../OsPrescaler/{OS | USER}/OsValue]
  This Freescale AUTOSAR OS specific attribute specifies initial prescaler value for the selected system or second timer hardware. The value of this attribute is fully hardware-dependent. For more details see "Power Architecture Platform-Specific Features". Note that the *Prescaler/Value* attribute value is not equal to divide factor of timer hardware.

  This attribute can be *AUTO* only if *Prescaler* value is *OS*. In case of *AUTO*, prescaler value is calculated during system generation. This attribute shall be defined if *Prescaler* value is USER. If the *TimerModuloValue* attribute is not *AUTO*, the *Value* shall be defined by numeric value. In case of *AUTO* prescaler value is calculated from the values of

ClockFrequency, *ClockDivider*, *ClockMultiplier* and corresponding *Period* attributes during system generation.

If the *Prescaler* is set to USER then it's value is used by OS only for time calculations, assuming that the appropriate HW register are set by User before call to StartOS.

- ***TimerModuloValue*** (UINT32)
  [.../{DEC | eMIOS | FIT | PIT | STM | PIT_RTI}/ OsTimerModuloValue]
  This Freescale AUTOSAR OS specific attribute specifies timer modulo value. The value of this attribute is fully hardware-dependent. For more details see "Power Architecture Platform-Specific Features".

  This attribute can be defined if *SysTimer* or *SecondTimer* value is *SWCOUNTER* only. If the attribute is set *AUTO*, timer modulo value is calculated from the values of the *ClockFrequency* and corresponded *Period* attributes during system generation.

- ***Freeze*** (BOOLEAN)
  [.../{eMIOS | PIT | STM | PIT_RTI}/OsFreeze]
  This attribute specifies timer freeze bit value. If this attribute has value TRUE, the timer will be frozen while the MCU is in the debug mode. If the same timer module is used for both System and Second Timers then this attributes shall have the same value.

- ***PeripheralClockDivider*** (UINT32)
  [.../{eMIOS | PIT | STM | PIT_RTI}/ OsPeripheralClockDivider]
  This Freescale AUTOSAR OS specific attribute specifies peripheral clock divider for peripheral frequency calculating. It is defined only for MPC5674F. The value of the attribute is to be set in the range [1..2].

Stack Related Attributes

These attributes define stack support in the system. The attributes should be defined inside the scope of the OS object in accordance with the following syntax (default attribute value are shown in bold type):

```
IsrStackSize = <integer>;
STACKMONITORING = <TRUE / FALSE> {
    Pattern = <integer / 0x55555555>;
```

```
        PatternSize = <1 / 2 / 4>;
    };
```

- ***IsrStackSize*** (UINT32)
  [OsOS/OsIsrStackSize]
  This Freescale AUTOSAR OS specific attribute specifies the ISR stack size in bytes. This stack is used for all ISRs category 2 and OS Timers ISRs in ECC1 within SC1 and in SC2 class (in SC3,. SC4 each ISR has its own stack definition). In this case the current status of the processor is saved onto the current stack, and stack is switched to the interrupt stack. This stack should be big enough to hold all nested interrupt stack frames in the system. In BCC1 within SC1 the Single Stack is used for Basic tasks and ISRs and this attribute is ignored.
- ***STACKMONITORING*** (BOOLEAN)
  [OsOS/OsStackMonitoring]
  This attribute turns on stack overflow runtime checking and stack usage services in all classes. OS performs check of the main, task and ISR stacks on context switch. If OS is configured to perform stack check then when OS detects stack overflow the *ProtectionHook* (or *ShutdownOS*, depending on Scalability Class, is called with *E_OS_STACKFAULT* error status code. In the classes with memory protection (SC3, SC4) the stack errors might be also catched by Memory Protection subsystem.
- ***Pattern*** (UINT32)
  [OsOS/OsStackMonitoring/TRUE/OsPattern]
  This Freescale AUTOSAR OS specific attribute defines the pattern used to fill stack memory. Possible values are 0..0xFFFFFFFF
- ***PatternSize*** (ENUM)
  [OsOS/OsStackMonitoring/TRUE/OsPatternSize]
  This Freescale AUTOSAR OS specific attribute defines the number of patterns (each pattern is 32bit width) to be checked.

Hook Routines Related Attributes

These attributes define hook routines support in the system. The attributes should be defined inside the scope of the OS object in accordance with the following syntax (default attribute value are shown in bold type):

```
STARTUPHOOK = <TRUE / FALSE>;
```

```
SHUTDOWNHOOK = <TRUE / FALSE>;
PRETASKHOOK = <TRUE / FALSE>;
POSTTASKHOOK = <TRUE / FALSE>;
IsrHooks = <TRUE / FALSE>;
ERRORHOOK = <TRUE / FALSE>;
USEGETSERVICEID = <TRUE / FALSE>;
USEPARAMETERACCESS = <TRUE / FALSE>;
PROTECTIONHOOK = <TRUE / FALSE>;
```

- ***STARTUPHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsStartupHook]
  The attribute defines whether the user's-provided hook is called by the system after startup but before starting dispatcher and initializing system timer or not (the *StartupHook* hook routine). This hook may be used by an application to perform hardware initialization actions, etc.

  The alternative way is to make such initialization steps in the task, which starts automatically. The hook is called with disabled interrupts.

- ***SHUTDOWNHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsShutdownHook]
  The attribute defines whether the user's-provided hook is called by the system during system shutdown or not (the *ShutdownHook* hook routine). The main purpose of this hook is to shutdown initialized hardware devices like timers, network controllers, etc. Besides, the reason for shutdown may be obtained through the error code.

  This hook is called after system timer shutdown (if system timer is configured in the system). Interrupts are disabled in this hook .

- ***PRETASKHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsPreTaskHook]
  The attribute defines whether the user's-provided hook is called from the scheduler code before the operating system enters context of the task or not (the *PreTaskHook* hook routine). In general, this hook is designed for debugging applications by means of tracing current task..

  It is not recommended to use this hook in normal working applications, because it adds significant timing overhead. If the attribute is defined, this hook is called for each task, i.e. it is not allowed to use this hook for particular task(s) only.

- ***POSTTASKHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsPostTaskHook]
  The attribute defines whether the user's-provided hook is called from the scheduler code after the operating system leaves context of the task or not (the *PostTaskHook* hook routine). In general, this hook is designed for debugging applications by means of tracing current task.

  It is not recommended to use this hook in normal working applications, because it adds significant timing overhead. If the attribute is defined, this hook is called for each task, i.e. it is not allowed to use this hook for particular task(s) only.

- ***IsrHooks*** (BOOLEAN)
  [OsOS/OsHooks/OsIsrHooks]
  This Freescale AUTOSAR OS specific attribute specifies whether the user-defined *PreIsrHook* and *PostIsrHook* functions are called before and after each ISR of category 2 and OS Timers ISRs.

- ***ERRORHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsErrorHook]
  The attribute defines whether the user's-provided hook is called by the system at the end of each system service which returns status not equal to E_OK. (the *ErrorHook* hook routine). This hook is designed for debugging applications by means of tracing error code, returned by the system service instead of checking error code after each call of system service. This hook increases the OS code size by approximately 10% and increases the timing in case of error during the service call.

  There is no need to check the error status of the each OS service call if this hook is used. This hook is useful as a temporary feature of a working (debugged) applications when some troubles occur. If the attribute is defined, this hook is called from the system service in which error occurs, i.e. it is not allowed to use this hook for particular service(s) only.

- ***USEGETSERVICEID*** (BOOLEAN)
  [OsOS/OsUseGetServiceId]
  This attribute specifies the possibility of usage the access macros to the service ID in the error hook.

- ***USEPARAMETERACCESS*** (BOOLEAN)
  [OsOS/OsUseParameterAccess]
  This attribute specifies the possibility of usage the access macros to the context related information in the error hook.

- ***PROTECTIONHOOK*** (BOOLEAN)
  [OsOS/OsHooks/OsProtectionHook]
  This attribute defines whether the user's-provided hook is called by OS in case of memory protection or timing violation.

OS-Application Definition

APPLICATION object is used in the OIL file to define OS-Application properties and data. It defines other system objects that are owned by OS-Application and data memory placement. Links with other system objects are defined via references.

```
APPLICATION <name of APPLICATION>{
     MemData0 = <TRUE / FALSE>;
     MemData1 = <TRUE / FALSE>;
     MemData2 = <TRUE / FALSE>;
     TRUSTED = <FALSE / TRUE>{
         TRUSTED_FUNCTION = <TRUE / FALSE>{
           NAME = <name of FUNCTION>;
         };
     };
     STARTUPHOOK = <TRUE / FALSE>;
     SHUTDOWNHOOK = <TRUE / FALSE>;
     ERRORHOOK = <TRUE / FALSE>;
     HAS_RESTARTTASK = <TRUE / FALSE>{
         RESTARTTASK = <name of TASK>
     };
     TASK = <name of TASK>;
     ISR = <name of ISR>;
     ALARM = <name of ALARM>;
     SCHEDULETABLE = <name of SCHEDULETABLE>;
     COUNTER = <name of COUNTER>;
     RESOURCE = <name of RESOURCE>;
     MESSAGE = <name of MESSAGE>;
};
```

Attributes

The APPLICATION object has the following attributes:

- ***MemData*<*0,1,2*>** (BOOLEAN)
  [OsApplication/MemData<0/1/2>]

These Freescale-specific attributes defines whether the OS-Application uses the given (0,1,2) memory region for data placement It allows to place data into different physical RAM.

- ***TRUSTED*** (BOOLEAN)
  [OsApplication/OsTrusted]
  The attribute defines whether the OS-Application is trusted or not.
- ***TRUSTED_FUNCTION*** (string)
  [OsApplication/OsTrusted/TRUE/OsTrusted_function]
  The attribute defines whether the *TRUSTED APPLICATION* has a “public” trusted functions that may be called from other OS-Applications.
- ***NAME*** ()
  [OsApplication/OsTrusted/TRUE/OsTrusted_function/TRUE/OsName]
  The attribute defines the name of the trusted function.
- ***STARTUPHOOK*** (BOOLEAN)
  [OsApplication/OsAppHooks/OsAppStartupHook]
  The attribute defines whether the user-provided application-specific hook (StartupHook_<App>) is called by the system after system *StartupHook* (if configured).

  The hook is called with all interrupts disabled.
- ***SHUTDOWNHOOK*** (BOOLEAN)
  [OsApplication/OsAppHooks/OsAppShutdownHook]
  The attribute defines whether the user-provided application-specific hook (ShutdownHook_<App>) is called by the system during system shutdown before the OS-wide *ShutdownHook* (if configured).

  The hook is called with all interrupts disabled.
- ***ERRORHOOK*** (BOOLEAN)
  [OsApplication/OsAppHooks/OsAppErrorHook]
  The attribute defines whether the user-provided application-specific error hook (ErrorHook_<App>) is called by the OS after the system *ErrorHook* (if configured).

  The hook is called with OS (category 2) interrupts disabled.
- ***HAS_RESTARTTASK*** (BOOLEAN)
  [OsApplication/OsHas_restarttask]
  The attribute defines whether the OS-Application has a “restart” *TASK* wich is called at the OS-Application restart.

- ***RESTARTTASK*** (reference)
  [OsApplication/OsRestartTask]
  The attribute defines the name of TASK to be restarted.
- ***TASK*** (reference)
  [OsApplication/OsTask]
  This reference is used to define a task owned by the OS-Application. There can be several *TASK* references. This parameter can be omitted.
- ***ISR*** (reference)
  [OsApplication/OsIsr]
  This reference is used to define an ISR owned by the OS-Application. There can be several *ISR* references. This parameter can be omitted.
- ***ALARM*** (reference)
  [OsApplication/OsAlarm]
  This reference is used to define an alarm owned by the OS-Application. There can be several *ALARM* references. This parameter can be omitted.
- ***SCHEDULETABLE*** (reference)
  [OsApplication/OsScheduletable]
  This reference is used to define a scheduletable owned by the OS-Application. There can be several *SCHEDULETABLE* references. This parameter can be omitted.
- ***COUNTER*** (reference)
  [OsApplication/OsCounter]
  This reference is used to define a counter owned by the OS-Application. There can be several *COUNTER* references. This parameter can be omitted.
- ***RESOURCE*** (reference)
  [OsApplication/OsResource]
  This reference is used to define a resource owned by the OS-Application. There can be several *RESOURCE* references. This parameter can be omitted.
- ***MESSAGE*** (reference)
  [OsApplication/OsMessage]
  This reference is used to define a message owned by the OS-Application. There can be several *MESSAGE* references. This parameter can be omitted.

Task Definition

Task object is used in the OIL file to define task data. Attributes of this object define task properties. Links with other system objects are defined via references. The keyword TASK is used for this object type. See "Task Management" for more detailed information about the Tasks. The syntax of the task object is as follows:

```
TASK <name of TASK> {
     PRIORITY = <integer>;
     SCHEDULE = <FULL / NON>;
     AUTOSTART = <TRUE / FALSE>{
       APPMODE = <name of APPMODE>;
     };
     ACTIVATION = <1>;
     STACKSIZE = <integer>;
     RESOURCE = <name of RESOURCE>;
     EVENT = <name of EVENT>;
     MESSAGE = <name of MESSAGE>;
     };
     TIMING_PROTECTION = < TRUE / FALSE >{
       EXECUTIONBUDGET = <integer>;
       TIMEFRAME = <integer>;
       MAXOSINTERRUPTLOCKTIME = <integer>;
       MAXALLINTERRUPTLOCKTIME = 0;
       LOCKINGTIME = RESOURCELOCK{
         RESOURCE = <name of RESOURCE>{
         MAXRESOURCELOCKTIME = <integer>;
       }
     }
     ACCESSING_APPLICATION = <name of
Application>;
};
```

Attributes

The TASK object has the following attributes:

- ***PRIORITY*** (UINT32)
  [OsTask/OsTaskPriority]
  The attribute specifies priority of the task. The value of this attribute is to be understood as a relative value; this means that values of the PRIORITY attribute show only the relative

ordering of the tasks. AUTOSAR defines the lowest priority as zero (0), the bigger value of the *PRIORITY* attribute corresponds to the higher priority. The value range is from 0 to 0xFFFFFFFF.

- ***SCHEDULE*** (ENUM)
  [OsTask/OsTaskSchedule]
  The attribute specifies the run-time behavior of the task. If the task may be preempted by another one at any point of execution - this task attribute shall have the *FULL* value (preemptable task). If the task can be preempted only at specific points of execution (explicit rescheduling points) the attribute shall have the *NON* value (non-preemptable task).
- ***AUTOSTART*** (BOOLEAN)
  [OsTask/OsTaskAutostart]
  The attribute determines whether the task is activated during the system start-up procedure or not.
- ***APPMODE*** (reference)
  [OsTask/OsTaskAutostart/TRUE/OsTaskAppModeRef]

  The attribute defines the application mode in which the task is auto-started. The attribute can be defined if the *AUTOSTART* attribute is set to *TRUE*. It may be ommited if only one *APPMODE* is defined in the *OS*.
- ***ACTIVATION*** (UINT32)
  [OsTask/OsTaskActivation]
  The attribute defines the maximum number of queued activation requests for the task. Freescale AUTOSAR OS does not support multiple activation, so this value is restricted to 1.
- ***STACKSIZE*** (UINT32)
  [OsTask/OsStackSize]
  This Freescale AUTOSAR OS specific attribute defines the task stack size in bytes. In SC1 it is applicable for extended tasks only. The *STACKSIZE* can not be set less then 64 bytes.
- ***RESOURCE*** (reference)
  [OsTask/OsTaskResourceRef]
  This reference is used to define a resource accessed by the task. If the task accesses a resource at run-time this resource shall be pointed. The resource Ceiling priority is calculated as the highest priority of tasks or ISRs accessing this resource. There can be several *RESOURCE* references. This parameter can be omitted.
- ***EVENT*** (reference)
  [OsTask/OsTaskEventRef]
  This reference is used to define an event the extended task may

react on. The task is considered as extended, if any event is reverenced. Otherwise the task considered as basic.

There can be several *EVENT* references. These events can be cleared and waited for by the task. All task events shall be pointed to define the event mask in case of auto-assignment (see section "Event Definition").

- ***MESSAGE*** (reference)
  [OsTask/OsTaskMessageRef]
  The reference specifies the message to be sent or received by the task.
- ***TIMING_PROTECTION***(BOOLEAN)
  [OsTask/OsTiming_protection]
  The attribute specifies whether Time Protection is applied to the Task
- ***EXECUTIONBUDGET***(UINT64)
  [OsTask/OsTaskExecutionBudget]
  The attribute specifies the Task execution budget in nanoseconds (in seconds in XML format). If it is set to 0 then no timing protection applied to this Task budget.
- ***TIMEFRAME***(UINT64)
  [OsTask/OsTaskTimeFrame]
  The attribute defines the length of timeframe (period) in nanoseconds (in seconds in XML format).
- ***LOCKINGTIME***(ENUM)
  [OsTask/OsTiming_protection/TRUE/OsLockingtime]
  The attribute specifies whether Time Protection is applied to the Resource(s).
- ***RESOURCE*** (reference)
  [OsTask/OsTaskLockingTime/OsTaskResourceLock/
  OsTaskResourceLockResourceRef]
  The attribute specifies to which *Resource* the locking time protection is applied.
- ***MAXRESOURCELOCKTIME***(UINT64)
  [OsTask/OsTiming_protection/TRUE/OsLockingtime/
  RESOURCELOCK / OsMaxresourcelocktime]
  The attribute defines the maximum locking time for the *Resource* in nanoseconds (in seconds in XML format).
- ***MAXOSINTERRUPTLOCKTIME***(UINT64)
  [OsTask/OsTiming_protection/TRUE/
  OsMaxosinterruptlocktime]
  The attribute defines the maximum locking time for the OS

interruts (ISRs of category 2) in nanoseconds (in seconds in XML format). If it is set to 0 then interrupt locking time is not controlled for this Task.

- ***MAXALLINTERRUPTLOCKTIME***(UINT64) **[OsTask/OsTiming_protection/TRUE/ OsMaxallinterruptlocktime]**
  The attribute is intended to define the maximum locking time for the ISRs of category 1. Its value always shall be set to 0 - OS can not control execution time of ISRs category 1.
- ***ACCESSING_APPLICATION*** (ENUM) **[OsTask/OsTaskAccessingApplication]**
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

ISR Definition

This object presents an Interrupt Service Routine. The keyword ISR is used for this object type. The syntax of the ISR object is as follows:

```
ISR <name of ISR> {
     CATEGORY = <1 / 2>;
     PRIORITY = <integer>;
     IsrFunction = <string / AUTO>;
     STACKSIZE = <integer>;
     IrqChannel = <enum>{;
       IrqNumber = <integer>;
     };
     RESOURCE = <name of RESOURCE>;
     MESSAGE = <name of MESSAGE>;
     };
     TIMING_PROTECTION = < TRUE / FALSE >{
       EXECUTIONTIME = <integer>;
       TIMELIMIT = <integer>;
       MAXOSINTERRUPTLOCKTIME = <integer>;
       MAXALLINTERRUPTLOCKTIME = 0;
       LOCKINGTIME = RESOURCELOCK {
         RESOURCE = <name of RESOURCE>{
         MAXRESOURCELOCKTIME = <integer>;
       }
     }
};
```

The same ISR name shall be used for corresponding ISR object declaration and definition (see "Conventions").

Attributes

This object has the following attributes:

- ***CATEGORY*** (UINT32)
  [OsIsr/OsIsrCategory]
  The attribute specifies category of the Interrupt Service Routine. (see "ISR Categories" for Interrupt Service Routine Categories details).
- ***PRIORITY*** (UINT32)
  [OsIsr/OsPriority]
  The attribute specifies the priority of the interrupt request for this ISR [1..15]. Freescale AUTOSAR OS initializes interrupt priority level registers for all configured external ISRs with defined *PRIORITY* values.

  PRIORITY of ISRs of category 2 shall be less then PRIORITY of any ISR of category 1. In SC2,4 class the TP ISR has the priority higher than each ISR of category 2, so ISRs of categore 1 priority shall be higher at least by 2 than maximum of ISRs cat.2 priority to provide "gap" for TP ISR.
- ***IsrFunction***(STRING)
  [OsIsr/OsIsrFunction]
  This Freescale AUTOSAR OS specific attributte specifies the name of ISR handler function. By default it is the name of ISR object itself. It may be used to assign one handler to several ISR objects. If the User defines this attribute its value shall exactly corresponds to the function name. This function shall be defined as *void <FuncName>(void)*.
- ***STACKSIZE*** (UINT32)
  [OsIsr/OsStackSize]
  The attribute specifies the ISRs stack size in bytes. It shall be defined for ISRs of category 2 in classes SC3 .. SC4. The *STACKSIZE* can not be set less then 64 bytes. Note that all ISRs of the same priorities shares the same stack memory in SC3,4. In this cases the larges value is used as the size of shared stack memory area.
- ***IrqChannel*** (ENUM)
  [OsIsr/OsIrqChannel]
  The attribute specifies the interrupt channel.

- ***IrqNumber*** (UINT32)
  [OsIsr/OsIrqChannel/EXTERNAL/OsIrqNumber]
  The attribute specifies the interrupt source for the EXTERNAL *IrqChannel*. The available values of this attribute are
  - 0..475 for MPC5644A,
  - 0..479 for MPC5674F,
  - 0..363 for MPC5634M.

  (For the correspondence between IrqNumber value and external interrupt sources please see the appropriate Hardware Technical References)
- ***RESOURCE*** (reference)
  [OsIsr/OsIsrResourceRef]
  The reference specifies resource accessed by the ISR. There can be several *RESOURCE* references. This parameter can be omitted. The reference can not be defined if *CATEGORY* value is equal to 1.
- ***MESSAGE*** (reference)
  [OsIsr/OsIsrMessageRef]
  The reference specifies the message to be sent or received by the ISR.
- ***TIMING_PROTECTION***(BOOLEAN)
  [OsIsr/OsTiming_protection]
  The attribute specifies whether Time Protection is applied to the ISR. Applicable only in SC2 and SC4 classes.
- ***EXECUTIONTIME***(UINT64)
  [OsIsr/OsTiming_protection/TRUE/OsExecutiontime]
  The attribute specifies the ISR worst execution time (budget) in nanoseconds (in seconds in XML format). If it is set to 0 then no timing protection applied to this ISR.
- ***TIMEFRAME*** (UINT64)
  [OsIsr/OsIsrTimeLimit]
  The attribute defines the length of timeframe for arrival rate protection in nanoseconds (in seconds in XML format).
- ***LOCKINGTIME***(ENUM)
  [OsIsr/OsIsrLockingTime]
  The attribute specifies whether Time Protection is applied to the Resource(s).
- ***RESOURCE*** (reference)
  [OsIsr/OsIsrResourceRef]

The attribute specifies to which *Resource* the locking time protection is applied.

- ***MAXRESOURCELOCKTIME***(UINT64)
  [OsIsr/OsIsrLockingTime/OsIsrResourceLock/ OsIsrResourceLockTime]
  The attribute defines the maximum locking time for the *Resource* in nanoseconds (in seconds in XML format).
- ***MAXOSINTERRUPTLOCKTIME***(UINT64)
  [OsIsr/OsTiming_protection/TRUE/ OsMaxosinterruptlocktime]
  The attribute defines the maximum locking time for the OS interruts (ISRs of category 2) in nanoseconds (in seconds in XML format). If it is set to 0 then interrupt locking time is not controlled for this ISR.
- ***MAXALLINTERRUPTLOCKTIME***(UINT64)
  [OsIsr/OsTiming_protection/TRUE/ OsMaxallinterruptlocktime]
  The attribute is intended to define the maximum locking time for the ISRs of category 1. Its value always shall be set to 0 - OS can not control execution time of ISRs category 1.

Schedule Table definition

This group of attributes controls task features. The attributes should be defined inside the scope of the OS object in accordance with the following syntax (default attribute value are shown in bold type):

```
SCHEDULETABLE <name of Schedule Table> {
     COUNTER = <name of Counter>;
     AUTOSTART = < TRUE / FALSE > {
       TYPE = <ABSOLUTE / RELATIVE / SYNCHRON>{
         ABSVALUE = <intenger>;
         RELOFFSET = <integer>};
       };
       APPMODE = <name of APPMODE>;
     }
     SYNC = <TRUE / FALSE>{
       SYNCSTRATEGY = <EXPLICIT / IMPLICIT>{
         EXPLICITPRECISION = <integer>;
       };
     };
```

```
      REPEATING = <TRUE / FALSE>;
      DURATION = <integer>;
      EXPIRYPOINTS = EXPIRYPOINT {
        OFFSET = <integer>;
        ACTION = <TASKACTIVATION / EVENTSETTING>{
          TASK = <name of TASK>;
          EVENT = <name of EVENT>;
        };
        ADJUSTABLEEXPPOINT = <TRUE / FALSE>{
          MAXADVANCE = <integer>;
          MAXRETARD = <integer>;
        };
      };
ACCESSING_APPLICATION = <name of Application>;

};
```

Attributes

This object has the following attributes:

- ***COUNTER*** (reference)
  [OsScheduleTable/OsScheduleTableCounterRef]
  The reference specifies the assigned counter. A Schedule Table shall be assigned to a particular counter to have an ability to operate.
- ***AUTOSTART*** (BOOLEAN)
  [OsScheduleTable/OsScheduleTableAutoStart]
  The attribute defines whether the Schedule Table is started automatically at system start-up depending on the application mode.
- ***TYPE*** (ENUM)
  [OsScheduleTable/OsScheduleTableAutoStart/ OsScheduleTableAutostartType]
  The attribute defines whether the Schedule Table is automatically started at system start-up syncronously or with given offset or absolute value of the assigned counter
- ***ABSVALUE*** (UINT32) (inside *AUTOSTART*)
  [OsScheduleTable/OsScheduleTableAutoStart/ OsScheduleTableAbsValue]
  This attribute defines the value of assigned counter at which the Schedule Table will start.

- ***RELOFFSET*** (UINT32) (inside *AUTOSTART*)
  [OsScheduleTable/OsScheduleTableAutoStart/ OsScheduleTableRelOffset]
  This attribute defines the offset from OS startup when Schedule Table will start in counter ticks.
- ***APPMODE*** (reference)
  [OsScheduleTable/OsAutoStart/OsAppMode]
  The attribute defines the application mode for which the ScheduleTable shall be started automatically at system start-up. The attribute should be defined if the *AUTOSTART* attribute is set to *TRUE*. It may be ommited if only one *APPMODE* is defined in the *OS*.
- ***SYNC*** (BOOLEAN)
  [OsScheduleTable/OsScheduleTableSync]
  The attribute defines if synchronization of schedule table is required.
- ***SYNCSTRATEGY*** (ENUM)
  [OsScheduleTable/OsScheduleTblTimeSync/ OsScheduleTblSyncStrategy]
  The attribute defines if the type of syncronization - *EXPLICIT* or *IMPLICIT*.
- ***EXPLICITPRECISION*** (UINT32)
  [OsScheduleTable/OsScheduleTblTimeSync/TRUE/ OsScheduleTblExplicitPrecision]
  The value of this attribute defines the threshold between synchronous and asynchronous state of ScheduleTable in ticks.
- ***REPEATING*** (BOOLEAN)
  [OsScheduleTable/OsScheduleTableRepeating]
  This attribute specifies wether the Schedule Table shall be executed once or periodically.
- ***DURATION*** (UINT32)
  [OsScheduleTable/OsScheduleTableDuration]
  This attribute specifies the length of the Schedule Table period.
- ***EXPIRYPOINTS=EXPIRYPOINT***
  [OsScheduleTable/OsScheduleTableExpiryPoint}
  Container for expire point. There shall be at least one *EXPIRYPOINT* in the Schedule Table.
- ***OFFSET*** (UINT64) (inside *ACTION*)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTblExpPointOffset]
  This attribute defines the offset from Schedule Table start for given *EXPIRYPOINT*.

- ***ACTION*** (UINT64)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTableTaskActivation or ../OsScheduleTableEventSetting]
  The attribute defines type of task notification used when the expire point reached. If the *ACTION* attribute is defined as *TASKACTIVATION*, the *TASK* reference defines the task to be activated when the point expires. If the *ACTION* attribute is defined as *EVENTSETTING*, then the *TASK* reference defines the task to be awaken, and *EVENT* reference defines the event to be set when the point expires.

  There might be several *ACTION*s defined inside one *EXPIRYPOINT*

- ***TASK*** (reference) (inside *ACTION*)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTableTaskActivation/ OsScheduleTableActivateTaskRef]
  The reference to a task which is to be notified via activation or event setting at the expiration point.

- ***EVENT*** (reference) (inside *ACTION*)
  [OsScheduleTable/OsScheduleTableAction/ OsScheduleTableSetEvent/OsScheduleTableSetEventRef]
  The reference specifies the event mask to be set at the expiration point. The event is considered as an inseparable pair of the task and the event belonging to this task, so the reference to the task which owns the events shall be also defined for this *ACTION*.

  The reference shall be defined if the *ACTION* value is *SETEVENT*.

- ***ADJUSTABLEEXPPOINT*** (BOOLEAN)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTblAdjustableExpPoint/ OsScheduleTableMaxAdvance]
  This attribute defines is the offset value of the EXPIRYPOINT is adjustable while synchronizing.

- ***MAXADVANCE*** (UINT32)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTblAdjustableExpPoint/ OsScheduleTableMaxAdvance]
  The attribute defines the maximum allowed value of offset increase while the schedule table is synchronized in the *EXPIRYPOINT*.

- ***MAXRETARD*** (UINT32)
  [OsScheduleTable/OsScheduleTableExpiryPoint/ OsScheduleTblAdjustableExpPoint/ OsScheduleTableMaxRetard
  The attribute defines the maximum allowed value of offset decrease while the schedule table is synchronized in the *EXPIRYPOINT*.
- ***ACCESSING_APPLICATION*** (ENUM)
  [OsScheduleTable/OsScheduleTableAccessingApp]
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

Resource Definition

This object is intended for the resource management. The resource *Ceiling priority* is calculated automatically on the basis of information about priorities of tasks using the resource. The keyword RESOURCE is used for this object type. Section "Resource Management" describes resource concept in AUTOSAR. The syntax of the resource object is as follows:

```
RESOURCE <name of resource> {
  RESOURCEPROPERTY = <STANDARD / LINKED /  INTERNAL> {
    LINKEDRESOURCE = <name of RESOURCE>
  };
};
```

- ***RESOURCEPROPERTY*** (ENUM)
  [OsResource/OsResourceProperty]
  The attribute specifies a property of the resource. The *STANDARD* value corresponds to a normal resource which is not linked to another resource and is not an internal resource. The *LINKED* value corresponds to a resource linked to another resource with the property *STANDARD* or *LINKED*. The *INTERNAL* value is appropriate to an internal resource which cannot be accessed by an application.

  Performance decreases if the RESOURCE object with the INTERNAL value of the RESOURCEPROPERTY subattribute is defined.
- ***LINKEDRESOURCE*** (reference)
  [OsResource/OsResourceProperty/LINKED/ OsResourceLinkedResource]

The attribute specifies the resource to which the linking shall be performed. The OS System Generator resolves chains of linked resources. This reference should be defined only if the value of the *RESOURCEPROPERTY* attribute is *LINKED*.

- ***ACCESSING_APPLICATION*** (ENUM)
  [OsResource/OsResourceAccessingApplication]
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

Event Definition

This object is intended for the event management. The event object has no references. The keyword EVENT is used for this object type. Section "Events" describes events in AUTOSAR. The syntax of the event object is as follows:

```
EVENT <name of EVENT> {
  MASK = <integer / AUTO>;
};
```

Attribute

The object has one standard attribute:

- ***MASK*** (UINT64)
  [OsEvent/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. The other way to assign event mask is to declare it as *AUTO*. In this case event masks will be assigned automatically according to their distribution among the tasks.

Counter Definition

This object presents AUTOSAR Operating system counters. Attributes of this object type define counter properties. A counter has no references, it is referenced to by another object. The keyword COUNTER is used for this object type. AUTOSAR OS counters are described in section "Counters and Alarms". The syntax of the counter object is:

```
COUNTER <name of COUNTER> {
     MINCYCLE = <integer>;
     MAXALLOWEDVALUE = <integer>;
     TICKSPERBASE = <integer>;
     TYPE = <SOFTWARE / HARDWARE>{
       DRIVER = OSINTERNAL,
       TIMECONSTANTS = TIMECONSTANT{
         NS = <integer>;
         CONSTNAME = “name”;
       };
};   ACCESSING_APPLICATION = <name of
Application>;
};
```

Attributes

The object has the following standard attributes:

- ***MINCYCLE*** (UINT32)
  [OsCounter/OsCounterMinCycle]
  The attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter. This parameter is always defined in ticks. The parameter has a sense only for systems with extended OS status since it is checked in this case only.
- ***MAXALLOWEDVALUE*** (UINT32)
  [OsCounter/OsCounterMaxAllowedValue]
  The attribute defines the maximum allowed counter value. After the counter reaches this value it rolls over and starts count again from zero. This parameter is always defined in ticks.
- ***TICKSPERBASE*** (UINT32)
  [OsCounter/OsCounterTicksPerBase]
  This is the user constant for representation of the number of ticks that are required to reach some counter-specific value. This value cannot be derived automatically from other counter related attributes. The interpretation is up to the user; it is not used by OS itself, OS just keeps this value.
- ***TYPE*** (ENUM)
  [OsCounter/OsCounterType]
  specifies the counter type, for counters assignrd to System / Second Timers the type shall match the type of the Timer.

- ***DRIVER*** (ENUM)
  [OsCounter/OsCounterType/HARDWARE/OsCounterDriver]
  - shall be *OSINTERNAL*, GPT is not supported.
- ***TIMECONSTANTS***
  [OsCounter/OsCounterType/HARDWARE/
  OsCounterTimeConstants]
  - allows to define the User constants, there may be several definitions or no one. The constants are defined in timer ticks
- ***NS*** (UINT64)
  [OsCounter/OsCounterType/HARDWARE/
  OsCounterTimeConstants/OsCounterTimeConstant/NS]
  - defines the time value for constant in nanoseconds
- ***CONSTNAME*** (STRING)
  [OsCounter/OsCounterType/HARDWARE/
  OsCounterTimeConstants/OsCounterTimeConstant/
  CONSTNAME]
  - specifies the name of generated constant. The value of generated constant in ticks corresponds to NS value.
- ***ACCESSING_APPLICATION*** (ENUM)
  [OsCounter/OsCounterAccessingApplication]
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

Alarm Definition

This object presents alarms. Links with other system objects are defined via references. The referenced counter and task must be already defined. The keyword ALARM is used for this object type. See section "Alarms" for information about alarms.

The syntax of an alarm object is as follows:

```
ALARM <name of ALARM> {
     COUNTER = <name of COUNTER>;
     ACTION = <SETEVENT / ACTIVATETASK /
ALARMCALLBACK / INCREMENTCOUNTER> {
       TASK = <name of TASK>;
       EVENT = <name of EVENT>;
       ALARMCALLBACKNAME = <string>;
       COUNTER = <name of COUNTER>;
     };
```

```
    AUTOSTART = <TRUE / FALSE> {
      ALARMTIME = <integer>;
      CYCLETIME = <integer>;
      TYPE = <ABSOLUTE / RELATIVE>;
      APPMODE = <name of APPMODE>;
    };
    ACCESSING_APPLICATION = <name of
Application>;
};
```

Attributes

The object has the following attributes:

- ***COUNTER*** (reference)
  [OsAlarm/OsAlarmCounterRef]
  The reference specifies the assigned counter. An alarm shall be assigned to a particular counter to have an ability to operate. Only one counter has to be assigned to the alarm.
- ***ACTION*** (ENUM)
  [OsAlarm/OsAlarmAction]
  The attribute defines which type of task notification is used when the alarm expires. For one alarm only one action is allowed. If the *ACTION* attribute is defined as *ACTIVATETASK*, the *TASK* reference defines the task to be activated when the alarm expires. If the *ACTION* attribute is defined as *SETEVENT*, then the *TASK* reference defines the task to be activated, and *EVENT* reference defines the event to be set when the alarm expires. If the *ACTION* attribute is defined as *ALARMCALLBACK*, then the *ALARMCALLBACKNAME* subattribute specifies the name of the callback routine called when the alarm expires. *ALARMCALLBACK* is not allowed in *SC2*, *SC3* and *SC4* classes. If the *ACTION* attribute is defined as *INCREMENTCOUNTER*, the *COUNTER* reference defines the counter to be incremented when the alarm expires.
- ***ALARMCALLBACKNAME*** (STRING)
  [OsAlarm/OsAlarmAction/OsAlarmCallback/ OsAlarmCallbackName]
  The attribute specifies the name of the callback routine called when the alarm expires. The parameter should be specified if the *ACTION* attribute is set as *ALARMCALLBACK*.
- ***COUNTER*** (reference) (inside *ACTION*)
  [OsAlarm/OsAlarmAction/OsAlarmIncrementCounter/

OsAlarmIncrementCounterRef]
The attribute specifies the counter to be incremented as ALARM action. It shall be different from the *COUNTER* to which this *ALARM* is assigned.

- ***TASK*** (reference)
  [OsAlarm/OsAlarmAction/OsAlarmActivateTask/ OsAlarmActivateTaskRef]
  The reference to a task which is to be notified via activation or event setting when the alarm expires.
- ***EVENT*** (reference)
  [OsAlarm/OsAlarmAction/OsAlarmSetEvent/ OsAlarmSetEventRef]
  The reference specifies the event mask to be set when the alarm expires. The event is considered as an inseparable pair of the task and the event belonging to this task, so the reference to the task which owns the events shall be also defined for this alarm. The reference shall be defined if the *ACTION* value is *SETEVENT*.
- ***AUTOSTART*** (BOOLEAN)
  [OsAlarm/OsAlarmAutostart]
  The attribute defines whether an alarm is started automatically at system start-up depending on the application mode. If the alarm should be started at the system start-up, the value is set to TRUE otherwise the value is set to FALSE. When the *AUTOSTART* attribute set to *TRUE*, the *ALARMTIME*, *CYCLETIME*, and *APPMODE* parameters should be defined.
- ***ALARMTIME*** (UINT32)
  [OsAlarm/OsAlarmAutostart/TRUE/OsAlarmAlarmTime]
  The attribute defines the time (in ticks) when the alarm shall expire first. The attribute should be be defined if the *AUTOSTART* attribute is set to *TRUE*.
- ***CYCLETIME*** (UINT32)
  [OsAlarm/OsAlarmAutostart/TRUE/OsAlarmCycleTime]
  The attribute defines the cycle time of a cyclic alarm in ticks. The attribute should be defined if the *AUTOSTART* attribute is set to *TRUE*.
- ***TYPE*** (ENUM)
  [OsAlarm/OsAlarmAutostart/TRUE/OsAlarmAutostartType]
  The attribute defines is the *ALARMTIME* atribute presents an absolute or relative (to the StratOS moment) value. The attribute should be defined if the *AUTOSTART* attribute is set to *TRUE*.

- ***APPMODE*** (reference)
  [OsAlarm/OsAlarmAutostart/TRUE/OsAlarmAppModeRef]
  The attribute defines the application mode for which the alarm shall be started automatically at system start-up. The attribute should be defined if the *AUTOSTART* attribute is set to *TRUE*. It may be ommited if only one *APPMODE* is defined in the *OS*.
- ***ACCESSING_APPLICATION*** (ENUM)
  [OsAlarm/OsAlarmAccessingApplication]
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

Message Definition

This object is intended to present either a Unqueued or a Queued message. Attributes of this object type define message properties. Links with other system objects are defined via references. The keyword MESSAGE is used for this object type. Messages concept is described in section "Communication". The syntax of a message object definition is as follows:

```
MESSAGE <name of MESSAGE> {
  MESSAGEPROPERTY = <SEND_STATIC_INTERNAL/
RECEIVE_UNQUEUED_INTERNAL/RECEIVE_QUEUED_INTERNAL>{
    SENDINGMESSAGE = <name of MESSAGE>;
    QUEUESIZE = <integer>;
    CDATATYPE = <string>;
    INITIALVALUE = <integer>;
  };
  NOTIFICATION = <ACTIVATETASK / SETEVENT / COMCALLBACK / FLAG /
NONE> {
    TASK = <name of TASK>;
    EVENT = <name of EVENT>;
    CALLBACKROUTINENAME = <name of function>;
    MESSAGE = <name of MESSAGE>;
    FLAGNAME = <string>;
  };
  ACCESSING_APPLICATION = <name of Application>;
};
```

Attributes

The object has the following standard attributes:

- ***MESSAGEPROPERTY*** (ENUM)
  [OsMessage/OsMessageProperty]
  The attribute specifies type of the message. In accordance with the *OSEK/VDX Communication, v.3.0.3, 20 July 2004* specification there are three types of internal messages:

  - *SEND_STATIC_INTERNAL,*
  - *RECEIVE_UNQUEUED_INTERNAL,*
  - *RECEIVE_QUEUED_INTERNAL*

  Queued message data is buffered and consumed by receive operations. Unqueued message data is not consumed and is overwritten by each call to *SendMessage*.

- ***SENDINGMESSAGE*** (reference)
  [OsMessage/OsMessageProperty/
  {OsMsgReceiveQueuedInternal
  |OsMsgReceiveUnqueuedInternal }/SendingMessage]
  specifies which message transfers data to this one. This parameter shall be defined only if *MESSAGEPROPERTY* value is *RECEIVE_UNQUEUED_INTERNAL* or *RECEIVE_QUEUED_INTERNAL*.

- ***QUEUESIZE*** (UINT64)
  [OsMessage/OsMessageProperty/
  OsMsgReceiveQueuedInternal/QueueSize]
  The attribute specifies number of items in the message queue. The parameter can be defined only if the *MESSAGEPROPERTY* attribute is *RECEIVE_QUEUED_INTERNAL*.

- ***CDATATYPE*** (STRING)
  [OsMessage/OsMessageProperty/OsMsgSendStaticInternal/
  CDataType]
  The attribute defines the data type of the message item. The message item can have the own type which shall be any C data type. Any ANSI-C type specifier is allowed. It is the standard C type identifier - *char, int, float, double* with any type modifiers (*signed, unsigned, short, long*) and also structure or union specifier (starting *struct or union*), enum specifier (starting *enum*), *typedef* name (any valid C-language identifier) enclosed in the double quotas. To use an array of standard C-language type the user must define the new type via *typedef* operator. In case of user's defined data types or enumerations such definitions must be in

the user's code before using files produced by SG. This parameter shall be defined only if *MESSAGEPROPERTY* value is *SEND_STATIC_INTERNAL*.

- ***INITIALVALUE*** (UINT64)
  [OsMessage/OsMessageProperty/OsMsgReceiveUnqueuedInternal/InitialValue]
  initial value of the message item. Only the values that are representable in the boundaries of UINT64 type may be set. It is the user responsibility to provide the initialization values with appropriate type qualifiers or type suffixes so the assignment of values to the message objects can be compiled. The values of this attributes shall be equal for receiving messages that have the same *SENDINGMESSAGE* value.
  This parameter may be defined only if *MESSAGEPROPERTY* value is *RECEIVE_UNQUEUED_INTERNAL*.
- ***NOTIFICATION*** (ENUM)
  [OsMessage/OsMsgNotification]
  The attribute defines which type of notification is used when the message arrives. One action per each receiving message is allowed. COMCALLBACK notification is allowed only in SC1.
- **TASK** (reference)
  [OsMessage/OsMsgNotification/OsMsgActivateTask/TaskRef]
  The reference specifies the task which shall be notified when the message arrives. This reference shall be defined only if the value of the *NOTIFICATION* attribute is *ACTIVATETASK* or *SETEVENT*.
- ***EVENT*** (reference)
  [OsMessage/OsMsgNotification/OsMsgSetEvent/EventRef]
  The reference specifies the event which is to be set when the message arrives. The event is considered as an inseparable pair of the task and the event belonging to this task, so the reference to the task which owns the events shall be also defined for this message. This reference shall be defined only if the value of the *NOTIFICATION* attribute is *SETEVENT*.
- ***CALLBACKROUTINENAME*** (STRING)
  [OsMessage/OsMsgNotification/OsMsgComCallback/CallbackRoutineName]
  The attribute defines the name of function to call as an action when the message arrives. It shall be defined only if the value of the *NOTIFICATION* attribute is *COMCALLBACK*.
- ***MESSAGE*** (STRING)
  [OsMessage/OsMsgNotification/OsMsgComCallback/

MessageRef]
The reference specifies the message to be sent or received by the given callback function. It may be defined only if the value of the *NOTIFICATION* attribute is *COMCALLBACK*.

- ***FLAGNAME*** (STRING)
  [OsMessage/OsMsgNotification/OsMsgFlag/FlagName]
  The attribute defines the name of the flag that is set when the message is sent. It shall be defined only if the value of the *NOTIFICATION* attribute is *FLAG*. It's value shall be an unique C identificator.
- ***ACCESSING_APPLICATION*** (ENUM)
  [OsMessage/OsMessageAccessingApplication]
  This attribute defines which application(s) may access this object via OS services. It is applicable only in SC3 and SC4 classes.

Application Modes Definition

It is possible to introduce different application modes inside one CPU container by means of objects named *APPMODE*. Each *APPMODE* object defines a set of autostarted Tasks and Alarms for the AUTOSAR OS application mode.

No standard attributes are defined for the APPMODE object. At least one APPMODE object has to be defined in a CPU.

The syntax of an application mode object definition is as follows:

```
APPMODE <name of mode>;
```

Freescale AUTOSAR OS supports up to 8 application modes.

COM Definition

The COM object represents OSEK communication subsystem properties on CPU. Only one COM object must be defined on the local CPU.

The syntax scheme of COM object is as follows:

```
COM <name of COM> {
     COMERRORHOOK = TRUE/FALSE;
```

```
    COMUSEGETSERVICEID = TRUE/FALSE;
    COMUSEPARAMETERACCESS = TRUE/FALSE;
    COMSTARTCOMEXTENSION = TRUE/FALSE;
    COMAPPMODE = "<string>";
    COMSTATUS = COMEXTENDED/COMSTANDARD;
};
```

Attributes

The object has the following standard attributes:

- ***COM*** (string)
  [OsCom]
  The attribute specifies the abstract name of COM object.
- ***COMERRORHOOK*** (BOOLEAN)
  [OsCom/ComErrorHook]
  The attribute specifies if the COM ErrorHook mechanism is used. It is recommended to set this attribute to the same value as OS attribute *ERRORHOOK*
- ***COMUSEGETSERVICEID*** (BOOLEAN)
  [OsCom/ComUseGetServiceId]
  The attribute defines whether the macro to get service ID is provided for COMErrorHook. It is recommended to set this attribute to the same value as OS attribute *USEGETSERVICEID*
- ***COMUSEPARAMETERACCESS*** (BOOLEAN)
  [OsCom/ComUseParameterAccess]
  The attribute defines whether the macros to get the first parameter of the service is provided for COMErrorHook. It is recommended to set this attribute to the same value as OS attribute *USEPARAMETRACCESS*
- ***COMSTARTCOMEXTENSION*** (BOOLEAN)
  [OsCom/ComStartComExtension]
  The attribute specifies if the COM startup extention mechanism is used If this attribute is set TRUE then *StartCOM* service calls the function StartCOMExtension which shall be provided by the User.
- ***COMAPPMODE*** (string)
  [OsCom/ComAppMode]
  The attribute specifies the COM mode name(s).
- ***COMSTATUS*** (ENUM)
  [OsCom/OsComStatus]
  The attribute specifies the COM error checking status. It is

recommended to set this attribute to the same value as OS attribute *STATUS*

13

Building of Application

This chapter contains information on how to build a user's application using Freescale AUTOSAR OS without the AUTOSAR framework. It may be useful for the first experiments with the OS. The Sample Application may be used as an example.

This chapter consists of the following sections:

- Application Structure
- Action Sequence to Build Application

Application Structure

The application developed using the AUTOSAR Operating System basis has a defined structure. An application consists of the Operating System kernel and several user's tasks and ISRs, which interact with the kernel by means of system services and internal mechanisms. ISRs receive control from hardware interrupt sources via the vector table. Tasks are controlled by the scheduler. They may use all means for intertask communications granted by AUTOSAR OS to pass data and synchronize each other.

Tasks and ISRs are considered as system objects. Resources, messages, counters, and alarms are also considered as system objects, because they are controlled by the Operating System. An application typically also has configuration tables for different system objects, task stacks and other entities. To create an application, the user should develop the desired application structure with all necessary objects and define interactions between them.

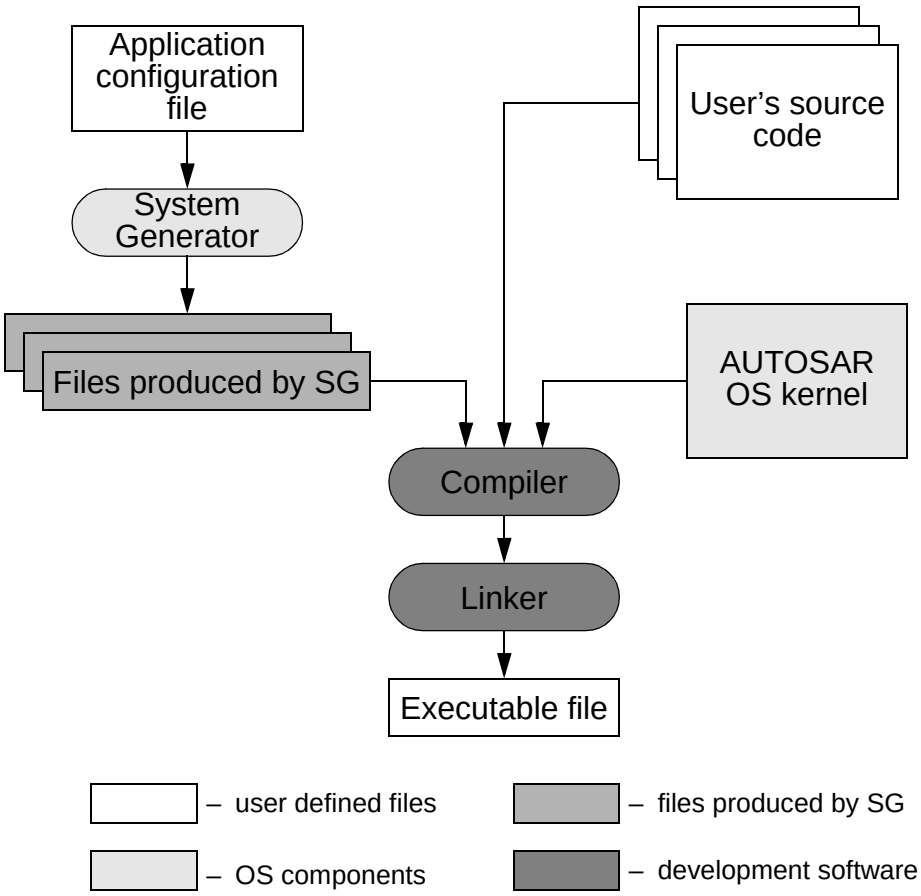
All global Operating System properties, system objects and their parameters are defined by the user statically and cannot be redefined at run time. Special application configuration file is designed to perform such definition and the special tool that processes this file. See "System Configuration". After processing,

files with system object descriptors are created automatically. These files provide the code for all required ROM and RAM structures, arrays, tables, variables, etc. for all system objects defined in the configuration file. Memory allocation is performed during start-up procedure.

Action Sequence to Build Application

To build an application using the AUTOSAR Operating System the user should perform a set of actions. These actions are relatively simple since the most important requirement is a clear understanding of the application algorithm. The actions include creating the application configuration file, processing this file by the System Generator, writing the user's source code, compiling all files and linking the application files together with the OS code. This process is shown on Figure 13.1.

Figure 13.1 Application Building Process



Application Configuration

Applications built using AUTOSAR OS are configured statically via the special configuration file written in OIL. "System Configuration" describes the structure of such file and "System Objects Definition" describes all possible statements in detail. This configuration file defines system specific parameters as well as system objects. Such a file can have any extension and the extension ".oil" is suggested by default.

The configuration file has to be processed by the special utility named System Generator (SG / SysGen). This utility is delivered as one of the parts of the Freescale AUTOSAR OS. This tool runs as a

32-bit console application for Windows XP / Vista and produces header and source files.

The following command is used to run SG:

```
sg [-options] oil_file
```

The following command line options are intended to control SG:

Table 13.1 System Generator Command Line Options

Option	Description	Default value
-c<name>	Defines <name> as the output c-data file name	“os_cfg.c” in the source OIL file directory
-h<name>	Defines <name> as the output header file name	“os_cfg.h” in the source OIL file directory
-i<path>	Defines <path> as the path for include files	N/A
-p<name>	Defines <name> as the OS property file name	“osprop.h” in the source OIL file directory
-o<name>	Defines <name> as the ORTI information file name	Input OIL file name with “.ort” extension
-t	Only verification for OIL input file. No output files shall be generated	No
-v	When this option is defined, SysGen shall output versions of all its components	No
-m<name>	Specifies the directory for generated osmemmap.h file	N/A
-L<name>	Specifies name of input link configuration file	N/A
-M<name>	Specifies name of output link configuration file	N/A
-T	When this option is defined, SysGen shall output information about system timers	No
-C	When this option is defined, SysGen shall output the name of the license file and lists the features to be considered by Sysgen	No

In addition to command line option the OSB_INCLUDE_DIR environment variable can be used to specify the set of directories to search for include files.

The SG utility produces three types of standard C-language files which are to be compiled and linked together with OS kernel code and user's source code:

1. The header file which describes the current properties of the operating system. This file contains the preprocessor directives `#define` and `#undef`. This file is used at compile time to build the OS kernel with the specified properties. The default filename is "osprop.h" but the user can assign another name (see "Source Files").
2. The application configuration header file which contains definitions of all system objects, data types, constants and external declarations of variables which describe system objects. This file is used to compile application files. The default filename is "os_cfg.h" but the user can assign another name (see Table 13.1).
3. The source file which contains initialized data and memory allocation for system objects. This file is compiled with "osprop.h" and another header files and then linked together with another application and OS files. The default filename is "os_cfg.c" but the user can assign another name (see Table 13.1).

For SC3 and SC4 System Generator generates a memory configuration file "osmemmap.h" which shall be included in the User application (MemMap.h file) in accordance with *AUTOSAR Specification of Memory Mapping v.1.1.0, 24 January 2007*. The file defines OS-Applications data sections.

To insure consistency between osmemmap.h and linker command file the SG inserts data sections into the "template" linker command file provided by User.

NOTE As a rule, the user is not allowed to edit files produced by the System Generator. It may lead to data inconsistency, compilation errors or unpredictable application behavior.

Source Files

Freescale AUTOSAR OS is delivered to the user as a set of source files. Header and source files of the Operating System are located in

the predefined directories. Paths to these directories have to be provided by the user.

The OS source code is compiled and linked together with other application's files. The header file "osprop.h" describing system properties defines which functionality will have the OS kernel in run time. Generally, changes in OIL file result in "osprop.h" modification and require recompilation of OS files. However some of object attributes do not affect "osprop.h" contents, see "OS Object Files Dependency" for details.

This file is included in all user's and OS' source files from the OS file "os.h" . Since the user can specify another name for property file the special macro *OSPROPH* is designed to substitute the name. The main header file "os.h shall be included in all C files to make OS services visible to the user application. Then OS configuration file (os_cfg.h), which contains declarations of all OS objects is also included from 'os.h'. Macro *OSCFGH* may be used to substitute a name of configuration header file.
The following code could be used in all user's files:

```
#include   "os.h"
```

The compiler command line (see "Compiling and Linking") in this case should have the options like this:

```
-dOSPROPH="<property_name>".
-dOSCFGH="<config_name>".
```

Where <property_name> is the name of the file with system properties definitions and *<config_name>* is the name of the file with system configuration definitions

Among other files SG generates configuration C-file containing definitions and initialization of OS configuration data declared in corresponding configuration header file.
Configuration C-file is a separate module, however, in some particular OS configurations, it could contain references to user defined data and structures (e.g. user's message structure types). This requires a method to provide SG generated configuration C-file with such user defined types and data declarations. Thus SG generates the following code in configuration C-file:

```
#if defined(APPTYPESH)
#include APPTYPESH /* user's header file */
#endif /* defined(APPTYPESH) */
```

The compiler command line (see "Compiling and Linking") in this case should have the option like this:

```
-dAPPTYPESH="<filename>".
```

`<filename>` is the name of the file with user defined structures and data declarations.

In the example below one data type and variable are defined by the user which are referenced in files generated by SG. Variable are defined in the user's file "user.c" and referenced in the produced file "os_cfg.c". The data type is defined in the user's file "user.h" and referenced in the produced file "os_cfg.c". The user's code should be the following:

USER.H file:

```
typedef struct tagMSG MSGTYPE;
struct tagMSG
{
  TickType timeStamp;
  int x;
};
extern MSGTYPE MsgA;
```

USER.C file:

```
#include "user.h"   /* include user defined data type */
...
MSGTYPE MsgA;        /* user defined variables */
...
```

OS_CFG.C file (generated by SG):

```
...
#if defined(APPTYPESH)
#include APPTYPESH /* user's header file */
```

```
#endif /* defined(APPTYPESH) */
...
/* SG generated code referring to user's type and data */
...
```

The compiler command line has the following option:

```
-dAPPTYPESH="USER.H".
```

Other variants are also possible.

The code of user's tasks and functions should be developed according to common rules of the C language. But some exceptions exist:

- The keyword *TASK* and *ISR* should be used to define a task and ISR correspondingly;
- For objects controlled by the Operating System the data types defined by the system must be used. The data types are described at the end of previous sections and in "System Services".

Compiling and Linking

When all needed header and source files are created or produced by the System Generator an application can be compiled and linked (for details see "Power Architecture Platform-Specific Features").

Linking process is controlled by the typical linker directive file.

OS Object Files Dependency

The OS object files are recompiled when content of the OS property file is changed. The OS configuration depends on many parameters defined in OIL file but there exists a set of parameters which can be changed without necessity to recompile OS files (or rebuild OS library). Note that number of objects affects constants defined in OS property file, so adding or deleting an object will cause OS recompiling.

The configuration attributes which do not require recompiling of OS when their values are changed listed below.

TASK attributes:

- ***PRIORITY***
- ***AUTOSTART***
- ***RESOURCE***
- ***STACKSIZE***

ISR attributes:

- ***RESOURCE***
- ***STACKSIZE***

COUNTER attributes:

- ***MINCYCLE***
- ***MAXALLOWEDVALUE***
- ***TICKSPERBASE***

ALARM attributes:

- ***COUNTER***
- The *ACTION* subattributes ***TASK***, ***EVENT,*** ***ALARMCALLBACKNAME***

MESSAGE attributes:

- ***CDATATYPE***
- The following *ACTION* subattributes:
  - ***TASK***
  - ***EVENT***
  - ***CALLBACKNAME***

In "Sample Application" the code of an AUTOSAR OS based application is provided. This code is a simple demonstration of Operating System mechanisms. It also demonstrates how to write the configuration file and source code.

14

Power Architecture Platform-Specific Features

This chapter discusses special Freescale AUTOSAR OS features for different MCU types and issues connected with porting applications to these MCUs.

This chapter consists of the following sections:

Compiler-Specific Features

The versions of tools that should be used with Freescale AUTOSAR OS are listed in the OS readme.txt file.

Used Library Functions

Freescale AUTOSAR OS itself does not use any functions from compiler run-time libraries except *memcpy* and *memset*.

Compiler Issues

Installation procedure defines environment variable values in sample makefiles and in common.mak. If they were not set during installation the user should do it manually to compile sample. These variables are the following:

- in sample\standard\common.mak

CYGWINDIR=[path] - path to the CygWin “bin” directory

DIABDIR = [path] – path to the WindRiver compiler

GHSDIR = [path] – path to the GreenHills compiler

CWDIR = [path] – path to the CodeWarrior compiler

- in each sample\standard\sc<n>\makefile

SSC_ROOT = [path] – path to the AUTOSAR directory

See the makefiles in the SAMPLE\STANDARD directory for additional information.

User should use the same compiler options as was used for developing and testing for compiling Freescale AUTOSAR OS modules.

NOTE It is recommended to use compiler/linker options list as it is set in the sample makefiles (except the debugging options). Using of the other compiler options may lead to possible compiling, linking or run-time problems. Therefore, in order to escape such problems and be sure that proper code is to be generated please refer to makefiles for exact option list.

Options for Freescale CodeWarrior

```
-proc Zen     # common target for all 55xx
-vle          # generate VLE code
-fp none      # floating-point not used
-msgstyle gcc # use GCC-like message style
-nostdinc      # do not use standard system include
              # paths
-nosyspath     # always search both user and system
              # path lists
-c            # compile only, do not link
-DOSCWPPC     # define CW compiler for OS
-ansi off      # thus enable inline assembler
-g            # include debug information
-pragma "read_only_switch_tables on" # required
             # for memory protection
-sym dwarf-2,full   # generate DWARF 2.x
-Cpp_exceptions off # disable C++ exceptions
```

```
-opt all,nopeep     # specify optimization
-func_align 4       # specify function alignment
-pragma "read_only_switch_tables on" # places
switch tables in a read-only section
-r                  # require prototypes
```

Next 2 options are required only for SC3, SC4:

```
-sdata2 0     # disables placement into small
              # constant data section
-sdata 0      # disables placement into small
              # data section
Sample for SC1 shall be compiled with:
-DFLASH_START  # defines full MCU initialization
by sample code
```

Linker options:

```
-fp none      # floating-point not used
-nostdlib     # do not use default library
-g            # include debug information
-sym dwarf-2,full   # generate DWARF 2.x
-map                # generate link map file
```

Options for GreenHills MULTI

Compiler options:

```
-cpu=ppc563xm  # target MPC5634M is used or
-cpu=ppc5646a  # target MPC5644A is used or
-cpu=ppc5674m  # target MPC5674F is used
-c             # compile only, do not link
-g             # generate debug information
-dual_debug    # enable the generation of DWARF
debugging information
-DOSGHSPPC     # define OS preprocessor macro
-O2            # optimization level
-vle           # generate VLE code
-noSPE         # disable use of SPE instructions
-fnone         # disable floating-point
--no_exceptions # disable exception handling
```

Next option is required only for SC3, SC4:

```
-sda=0         # size limit for "small data"
variables, disables small data.
```

```
Sample for SC1 shall be compiled with:
-DFLASH_START  # defines full MCU initialization
by sample code
```

Linker options:

```
-nocpp         # prevents the creation of C++
constructor/destructor tables
-uvfd          # delete C++ virtual functions
-Mn            # generate detailed map file
-delete       # remove functions that are not used
-lstartup # run-time environment startup routines
-lsys  # run-time environment system routines
-larch  # target specific runtime support
-lansi # standard C library
```

Options for Windriver

```
-tPPCE200Z4VEN:simple # target for MPC5644A or
-tPPCE200Z3VEN:simple # target for MPC5634M or
-tPPCE200Z7VEG:simple # target for MPC5674F
-c                # stop after assembly
-Wa,-Xisa-vle     # pass -Xisa-vle option to
assembler
-DOSDIABPPC       # define OS macro OSDIABPPC
-XO               # enable all standard
optimizations
-Xdialect-ansi    # follow ANSI C standard with
some addditions (default)
-g                  # generate information
-Xdebug-dwarf2    # generate symbolic debug
information in dwarf2 format
-Xdebug-inline-on # generate debug info for inline
functions
-Xdebug-local-all # force generation of type
information for all local variables
-Xdebug-local-cie     # create common information
entry per module
-Xdebug-struct-all   # force generation of type
information for all types
-Xnested-interrupts  # allow nested interrupts
-Xforce-declarations # require prototypes
```

Next option is required only for SC3, SC4:

```
-Xsmall-data=0      # size limit for "small data"
variables, disables small data
Sample for SC1 shall be compiled with:
-DFLASH_START  # defines full MCU initialization
by sample code
```

Linker options:

```
-tPPCE200Z4VEN:simple   # target processor
-Xelf              # use ELF format for output file
-m6                 # generate detailed link map
```

Linker command file

Strict discipline of data placement is required to provide memory protection in AUTOSAR OS. It implies usage of System Generator to insert code and data sections, as they defined by User in the OIL file, into the linker command file. In order to support this insertion the linker file shall have special comments in predifined places. For classes with memory protection the input link file shall be specified to SysGen with switch “-L” and output link file with switch “-M”. Please note that SysGen generates the data for MPU initialization and the data for linker from OIL, thus insures the consistency of memory definition. The same mechanism may be used for SC1 and SC2 classes, but in this case the User may also use his own linker command file without SysGen pre-processing, providing that OS sections - .ostext, .osrodata, .oshook, .osbss and .vects has appropriate placement.

The user application code (Tasks and ISRs bodies) is placed into the section .appcode. Please refer to the sample linker command files for example.

NOTE It is not possible to use the small data sections for non-trusted OS-Applications due to HW restrictions.

General and Special Purpose Registers Usage

The Special purpose registers 1 and 4 (SPRG1, SPRG4) are used by the Freescale AUTOSAR OS / MPC56xxAM.

Stack Size

Generally, the recommended minimal task stack size is:

- 200 bytes for simplest preemptive tasks
- 300 bytes for more complex preemptive tasks (with task, event, message manipulation)
- 400 bytes for complex preemptive tasks (task, event, message manipulation, interrupts)

Stack Usage for OS Services

The task stack usage depends on services called in the task and usage of ISRs in the application. Data from the tables below can be used for the rough estimation of task and ISR stack size.

The values (in bytes) given in the table were measured for the typical application configurations for Freescale AUTOSAR OS / MPC56xxAM compiled with CodeWarror, GreenHills and WindRiver compilers for 4 Scalability Classes. AUTOSAR is a highly scalable RTOS and most of the values depend on configuration, so the numbers in the table should be taken as an approximate values. The data are given for all scalability classes, for SC3 and SC4 data are measured for trusted OS-Application (Tr) and for non-Trusted (non).

Table 14.1 OS Stack Usage for CodeWarrior

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	CC			Tr	non	Tr	non
Empty BASIC task	BCC1	64	112	112	120	128	152
TerminateTask	BCC1	0	80	80	88	96	120
ChainTask	BCC1	0	96	80	88	112	136
Schedule	BCC1	48	16	16	40	16	40
GetTaskID	BCC1	0	0	32	56	32	56

Table 14.1 OS Stack Usage for CodeWarrior

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
GetTaskState	BCC1	0	0	32	56	32	56
DisableAllInterrupts	BCC1	0	0	0	24	0	24
EnableAllInterrupts	BCC1	0	0	0	24	0	24
SuspendAllInterrupts	BCC1	0	0	0	24	0	24
ResumeAllInterrupts	BCC1	0	0	0	24	0	24
SuspendOSInterrupts	BCC1	0	0	0	0	0	24
ResumeOSInterrupts	BCC1	0	0	0	0	0	24
ActivateTask	ECC1	48	112	96	104	128	152
TerminateTask	ECC1	48	80	80	88	96	120
ChainTask	ECC1	48	96	80	88	112	136
Schedule	ECC1	48	96	80	88	112	136
SetEvent	ECC1	48	112	96	104	128	152
ClearEvent	ECC1	0	0	0	24	0	24
GetEvent	ECC1	0	0	32	56	32	56
WaitEvent	ECC1	48	112	96	104	128	152
GetResource	BCC1	48	80	48	72	80	104
ReleaseResource	BCC1	32	80	32	56	80	104
SendMessage	BCC1	16	32	32	56	32	56
ReceiveMessage	BCC1	16	32	32	56	32	56
InitCounter	BCC1	0	0	0	24	0	24
IncrementCounter	BCC1	80	96	80	104	96	120
GetCounterValue	BCC1	0	0	32	56	32	56
GetCounterInfo	BCC1	0	0	32	56	32	56
GetAlarmBase	BCC1	0	0	32	56	32	56
SetRelAlarm	BCC1	80	80	80	104	80	104
SetAbsAlarm	BCC1	80	80	80	104	80	104
CancelAlarm	BCC1	80	80	80	104	80	104
GetAlarm	BCC1	0	0	32	56	32	56
StartSchedule-TableRel	BCC1	80	80	80	104	80	104
StopScheduleTable	BCC1	80	80	80	104	80	104

Table 14.1 OS Stack Usage for CodeWarrior

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
StartSchedule-TableAbs	BCC1	80	80	80	104	80	104
GetScheduleTa-bleStatus	BCC1	0	0	32	56	32	56
SetSchedule-TableAsync	BCC1	0	0	0	24	0	24
SyncScheduleTable	BCC1	32	32	32	56	32	56
NextScheduleTable	BCC1	0	0	0	24	0	24
EXTENDED task with WaitEvent inside end-less loop	ECC1	64	128	112	136	128	152
ISR category 2, task stack	ECC1	112	144	128	184	160	216
Empty ISR category 2, ISR stack	ECC1	4	4	0	64	0	64
SystemTimer ISR, Alarm activated (ISR stack)	ECC1	124	156	132	132	164	164
Empty ISR category 1	BCC1, ECC1	112	112	112	N/A	128	N/A

Table 14.2 OS Stack Usage for WindRiver

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
Empty BASIC task	BCC1	48	112	112	136	160	184
TerminateTask	BCC1	0	96	96	120	144	168
ChainTask	BCC1	0	112	96	120	160	184
Schedule	BCC1	16	16	32	56	32	56
GetTaskID	BCC1	0	0	64	88	64	88
GetTaskState	BCC1	0	32	80	104	80	104
DisableAllInterrupts	BCC1	16	16	48	72	48	72
EnableAllInterrupts	BCC1	0	0	32	56	32	56
SuspendAllInterrupts	BCC1	32	32	48	72	48	72

Table 14.2 OS Stack Usage for WindRiver

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
ResumeAllInterrupts	BCC1	16	16	32	56	32	56
SuspendOSInterrupts	BCC1	0	16	0	0	32	56
ResumeOSInterrupts	BCC1	0	16	0	0	32	56
ActivateTask	ECC1	48	112	112	136	160	184
TerminateTask	ECC1	32	96	96	120	144	168
ChainTask	ECC1	32	112	96	120	160	184
Schedule	ECC1	32	112	96	120	160	184
SetEvent	ECC1	32	112	112	136	160	184
ClearEvent	ECC1	0	32	16	40	48	72
GetEvent	ECC1	0	0	32	56	32	56
WaitEvent	ECC1	32	96	96	120	144	168
GetResource	BCC1	16	80	48	72	96	120
ReleaseResource	BCC1	16	80	48	72	96	120
SendMessage	BCC1	32	48	80	104	80	104
ReceiveMessage	BCC1	32	48	64	88	80	104
InitCounter	BCC1	0	0	48	72	48	72
IncrementCounter	BCC1	64	64	80	104	80	104
GetCounterValue	BCC1	0	0	80	104	80	104
GetCounterInfo	BCC1	0	0	64	88	64	88
GetAlarmBase	BCC1	0	0	64	88	64	88
SetRelAlarm	BCC1	80	80	112	136	112	136
SetAbsAlarm	BCC1	80	80	112	136	112	136
CancelAlarm	BCC1	64	64	80	104	80	104
GetAlarm	BCC1	32	32	80	104	80	104
StartSchedule-TableRel	BCC1	80	80	96	120	96	120
StopScheduleTable	BCC1	80	80	96	120	96	120
StartSchedule-TableAbs	BCC1	80	80	96	120	96	120
GetScheduleTa-bleStatus	BCC1	0	0	32	56	32	56

Table 14.2 OS Stack Usage for WindRiver

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
SetSchedule-TableAsync	BCC1	32	32	48	72	48	72
SyncScheduleTable	BCC1	48	48	64	88	64	88
NextScheduleTable	BCC1	32	32	48	72	48	72
EXTENDED task with WaitEvent inside end-less loop	ECC1	48	112	128	152	160	184
ISR category 2, task stack	ECC1	128	144	128	200	160	232
Empty ISR category 2, ISR stack	ECC1	4	60	24	112	72	112
SystemTimer ISR, Alarm activated (ISR stack)	ECC1	92	124	100	100	132	132
Empty ISR category 1	BCC1, ECC1	96	112	96	N/A	112	N/A

Table 14.3 OS Stack Usage for GreenHills

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
Empty BASIC task	BCC1	48	96	64	88	96	120
TerminateTask	BCC1	0	64	48	72	64	88
ChainTask	BCC1	0	64	48	72	64	88
Schedule	BCC1	16	16	16	40	16	40
GetTaskID	BCC1	0	0	16	40	16	40
GetTaskState	BCC1	0	0	32	56	32	56
DisableAllInterrupts	BCC1	0	0	0	24	0	24
EnableAllInterrupts	BCC1	0	0	0	24	0	24
SuspendAllInterrupts	BCC1	0	0	0	24	0	24
ResumeAllInterrupts	BCC1	0	0	0	24	0	24
SuspendOSInterrupts	BCC1	0	0	0	0	0	24
ResumeOSInterrupts	BCC1	0	0	0	0	0	24

Table 14.3 OS Stack Usage for GreenHills

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
ActivateTask	ECC1	48	96	64	88	96	120
TerminateTask	ECC1	32	64	48	72	64	88
ChainTask	ECC1	32	80	48	72	80	104
Schedule	ECC1	32	80	48	72	80	104
SetEvent	ECC1	32	96	64	88	96	120
ClearEvent	ECC1	0	0	0	24	0	24
GetEvent	ECC1	0	0	16	40	16	40
WaitEvent	ECC1	32	80	64	88	80	104
GetResource	BCC1	0	64	0	24	64	88
ReleaseResource	BCC1	16	32	16	40	32	56
SendMessage	BCC1	16	16	16	40	32	56
ReceiveMessage	BCC1	16	16	16	40	32	56
InitCounter	BCC1	0	0	0	24	0	24
IncrementCounter	BCC1	64	64	64	88	64	88
GetCounterValue	BCC1	0	0	32	56	32	56
GetCounterInfo	BCC1	0	0	16	40	16	40
GetAlarmBase	BCC1	0	0	16	40	16	40
SetRelAlarm	BCC1	48	48	48	72	48	72
SetAbsAlarm	BCC1	48	48	48	72	48	72
CancelAlarm	BCC1	32	32	32	56	32	56
GetAlarm	BCC1	0	0	32	56	32	56
StartSchedule-TableRel	BCC1	32	32	32	56	32	56
StopScheduleTable	BCC1	48	48	48	72	48	72
StartSchedule-TableAbs	BCC1	32	32	32	56	32	56
GetScheduleTa-bleStatus	BCC1	0	0	16	40	16	40
SetSchedule-TableAsync	BCC1	0	0	0	24	0	24
SyncScheduleTable	BCC1	32	32	32	56	32	56
NextScheduleTable	BCC1	0	0	0	24	0	24

Table 14.3 OS Stack Usage for GreenHills

Scalability class:		SC1	SC2	SC3	SC3	SC4	SC4
System service	**CC**			**Tr**	**non**	**Tr**	**non**
EXTENDED task with WaitEvent inside endless loop	ECC1	32	96	80	104	96	136
ISR category 2, task stack	ECC1	112	160	128	184	160	216
Empty ISR category 2, ISR stack	ECC1	4	4	0	64	0	64
SystemTimer ISR, Alarm activated (ISR stack)	ECC1	76	124	84	84	132	132
Empty ISR category 1	BCC1, ECC1	96	112	96	N/A	112	N/A

MPC56xx specific Features

Programming Model

Freescale AUTOSAR OS / MPC56xxAM is developed to work with Variable instruction Length Encoding (VLE) only. OS always works in the big-endian byte ordering mode.

MPC can operate at either of two privilege levels. Supervisor level is more privileged than the User level.

In SC3 and SC4 classes Freescale AUTOSAR OS uses Supervisor mode for OS itself and Trusted Applications while non-trusted OS-Applications are executed in User mode. The OS services are called via ‘sc’ command in SC3,4 classes. In SC1 and SC2 everything runs in Supervisor mode.

Target Hardware Initialization

Freescale AUTOSAR OS initializes MPU and used system timers. The TimeBase is cleared when Timing Protection is configured. All other target hardware (memory used, CPU clock frequency, etc.) shall be initialized before OS startup. The Freescale AUTOSAR OS /

MPC56xxAM package includes the files containing some initialization code located in the HWSPEC directory:

These files are the example of the HW initialization for work with MCUs supported by the OS.

MPU Descriptors usage

The table below shows the usage of MPU descriptors by Freescale AUTOSAR OS / MPC56xxAM in Scalability Classes with memory protection - SC3, SC4.

Table 14.4 Usage of MPU Descriptors by OS

Number	Memory area	Process ID - PID	Access rights for OS-Application	
			Trusted	non-Trusted
0	Region1	any	FULL	none
1	Stacks	OS	(accessed only by OS)	
2	Region2	any	FULL	none
3	Code & Const	any	Read/Execute	Read/Execute
4	Appl. Data	any	Read/Write	Read/Write
5	Active Stack	any	Read/Write	Read/Write
8	Appl. Data1	any	Read/Write	Read/Write
9	Appl. Data2	any	Read/Write	Read/Write

Region 1 includes all memory before Stacks area, Region2 - all memory after Stacks, including peripheral registers. Descriptors 0 .. 3 are static, while 4, 5, 8 and 9 are reloaded at run-time.

The address and size for MPU entries are calculated from the linker constants for the correspondent sections. In SC1 and SC2 classes OS does not handles the MPU.

Interrupt handling and Vector Table

The interrupt vector table is defined in the file “vector.c”. The file is delivered with the Freescale AUTOSAR OS and located in the ssc\hwspec directory. This file contains entries for external

interrupts dispatcher, System service dispatcher (called via 'sc' command) and for exception handlers.

OS/MPC56xxAM provides the external interrupt (sharing IVOR4 interrupt vector) distinction and the appropriate interrupt handler call. In this case OS/MPC56xxAM snaps up the external interrupt exception, makes required save/restore operations and calls appropriated interrupt handler.

The user should copy vector.c file into the project directory and then modify it if needed. The modification may be required only if the User wishes to use own ISR(s) of category 1 for DEC and/or FIT interrupts without redirection thru OS interrupt dispatcher. In this case the user has to place his interrupt handler(s) on the appropriate position in the "vector.c" file.

MSR Bits Manipulation

The user shall not manipulate with MSR general-purpose bits like FP, ME, FE0, FE1, IP during OS run-time. If it is necessary to set MSR ME bit (for example), then it should be set before OS start-up.

Implementation Background

In classes SC3, SC4 the OS code and trusted OS-Applications code is executed in supervisor CPU mode, nontrusted OS-Application code is executed in the user CPU mode. In SC1, SC2 everything is executed in supervisor mode and OS does not provides an initialization of MPU.

The OS does not allows usage of Critical Interrupt input as an OS ISR, it is treated as an exception. The OS does not change the state of CE bit in MSR, except it is cleared on OS shutdown.

All external interrupts, including ISRs of category 1, are redirected to the OSInterruptDispatcher in order to create a correct ISR execution context.

Using Floating Point

Power Architecture Book E floating-point instructions are not supported in MPC56xx hardware therefore the Freescale AUTOSAR

OS/MPC56xxAM does not supports HW floating point operations. Software floating point (compiler support and libraries) may be used as usual as they do not require special support from the OS. It shall be taken in account that the OS saves only 32-bit registers content on Task switch and when interrupt occurs. 64-bit registers are not supported by the OS.

Nested Interrupts

According to the OSEK OS v.2.2 specification there are no any services which enables interrupts directly. Therefore nested interrupts with the same hardware levels can not occur. Application must not enable interrupts in ISR using direct manipulation of CPU registers – it may cause an unpredictable application behavior. Interrupts of different levels are processed in usual way and may be nested.

Interrupt Dispatcher

Freescale AUTOSAR OS/MPC56xxAM supports multilevel interrupts; Interrupt Controller is used in Software mode. OS Interrupt Dispatcher processes interrupts in accordance with theirs priority, so interrupts of higher levels are enabled while interrupts of the same and lower levels are disabled when the ISR code is executed.

NOTE The Timing Protection system uses software setable interrupt 5 in order to signal internal TP event.

OS and Application Stacks

The Freescale AUTOSAR OS uses different approaches to stack definition and usage depending on configuration.

In BCC1 within SC1 class the OS uses the “Single Stack” for all Tasks and ISR, i.e. OS never switches the stack and everithing is executed on the stack where StartOS was called.

In ECC1 within SC1 class the OS uses the “Single Stack” for all Basic Tasks, common ISR stack for all ISRs of category 2 and OS Timers ISRs and separate stacks for Extended Tasks.

In SC2 the separate stack is used for Time Protection ISR, common ISR stack is used for all ISRs of category 2 and OS Timers ISRs, and separate (own) stacks are used for all Tasks - it is nesessary because any Task may be killed and then restarted.

In SC3 and SC4 classes all Tasks and ISRs has own stacks, but the ISRs of the same *PRIORITY* shares the stack memory.

The ISRs of category 1 are allways executed on the stack of the preemted Task / ISR.

Timer Hardware

The special OS attributes are introduced to define hardware interrupt source and desired parameters for counters assigned to the System and / or Second Timer. Timing Protection Timer is also defined in OIL, but only STM timer can be used for TP; the priority of the TP interrupt is calculated by SysGen so it is higher then the priority of any ISR category 2 and / or System / Second Timers.

Note that counter assigned to System (Second) Timer uses interrupts from corresponding timer channel. At run time Freescale AUTOSAR OS enables CPU interrupts and timers interrupts corresponding to System, Second and TP Timers. User shall not directly manipulate with System / Second / TP timer hardware when the OS is running. However the timers hardware may be initialized prior to calling to StartOS function. In this case the *Prescaler* attribute may be set to *USER* thus disabling timer prescaler reinitialization at OS startup. If *Prescaler* is set to *OS*, its *Value* is written to the prescaler bits of the correspondent timer. Note that the *Prescaler/ Value* attribute value is not equal to divide factor of timer hardware. Timers configuration example

The system definition statements for the Freescale AUTOSAR OS / MPC56xxAM has the following form:

```
OS <name> {
...
     SysTimer = <SWCOUNTER / HWCOUNTER>{
       COUNTER = <CounterName>;
       ISRPRIORITY = <Priority>;
       TimerHardware = <HardwareType> {
         TimerModuloValue = <HardwareModulo>;
```

```
        Freeze = TRUE / FALSE;
      };
    };
...
};
```

This Freescale AUTOSAR OS / MPC56xxAM contains system timers code for supported derivativies. The following hardware sources can be used for System and / or Second Timers:

- DEC (Decrementer Counter);
- FIT (Fixed Interval Timer)
- PIT (Periodic Interrupt Timer)
- PIT_RTI (Periodic Interrupt Timer - Real Time Interrupt)
- eMIOS (timers of Enhanced Modular Input / Output Subsystem)
- STM (System Timer Module)

The *ISRPRIORITY* attribute specifies the timer interrupt request priority. This attribute shall have the value from the range 1..15.

15

Application Troubleshooting

In this chapter some advice is given which may be useful for developers working with the Freescale AUTOSAR OS.

This chapter consists of the following sections:

System Generation

The System Generator is used to generate the code for the Freescale AUTOSAR OS kernel and all application objects (tasks, messages, etc.). This tool processed the configuration file created by the user and reports about inconsistencies and errors in it. Most of possible mistakes in application configuration process can be eliminated with the help of SG. See "System Configuration" and "Building of Application" about system generation process.

If an undocumented problem arises please provide us with the detailed description of it and we will help to resolve the problem. See "Technical Support Information" for contact information.

Known Problems

Troubleshooting

Problem A: Development software does not work under MS Windows.

Reason 1: Inappropriate development hardware or software is used.

Reason 2: The OS software is installed into the directory with spaces (like "`C:\Program Files`").

Workaround 1: Install the software into the directory without spaces.

Reason 3: Environment variables `GHSDIR and/or DIABDIR, CWDIR` in the sample\standard\common.mak are not correct.

*Workaround 1:*Set correct environment variables.

16

System Services

This chapter provides a detailed description for all AUTOSAR Operating System run-time services, with appropriate examples.

This chapter consists of the following sections:

General

This chapter provides detailed description of all AUTOSAR OS run-time services including hook routines. Also declarations of system objects – the constructional elements – are described here. The services are arranged in logical groups – for the task management, the interrupt management, etc.

Examples of code are also provided for every logical group. These examples have no practical meaning, they only show how it is possible to use OS calls in an application.

The following scheme is used for service description:

Declaration element:

Syntax: Operating System interface in ANSI-C syntax.

Input: List of all input parameters.

Description: Explanation of the constructional element.

Particularities: Explanation of restrictions relating to the utilization.

Conformance: Specifies the Conformance Classes where the declaration element is provided.

Service description:

Syntax: Operating System interface in ANSI-C syntax.

Input: List of all input parameters.

Output: List of all output parameters. Transfers via the memory use the memory reference as input parameter and the memory contents as output parameter. To clarify the description, the reference is already specified among the output parameters.

Description: Explanation of the functionality of the operating system service.

Particularities: Explanations of restrictions relating to the utilization of the service.

Status: List of possible return values if service returns status of *StatusType* type.

- Standard:

  List of return values provided in the operating system's standard version. Special case – service does not return status.

- Extended:

  List of additional return values in the operating system's extended version.

Conformance: Specifies the Conformance Classes where the service is provided.

AUTOSAR OS-Application Services

This section describes services that are related to OS-Application handling and memory protection

Data Types

The AUTOSAR OS establishes the following data types for the OS-Applications and memory management:

- ApplicationType – the abstract data type for OS-Application identification
- RestartType - argument type for TerminateApplication() service
- MemoryStartAddressType - data type for pointer which is able to point to any location in the MCU address space
- MemorySizeType - data type for the size of a memory region;
- TrustedFunctionParameterRefType - reference type for arguments of Trusted Function (it is void*).

System macros for memory access

The following macros return non-zero value if the corresponded type of access is available:

- OSMEMORY_IS_READABLE(x)
- OSMEMORY_IS_WRITEABLE(x)
- OSMEMORY_IS_EXECUTABLE(x)
- OSMEMORY_IS_STACKSPACE(x)

This macros are applicable only to the values of type AccessType.

Constants

- *INVALID_OSAPPLICATION* – constant of data type *ApplicationType* for undefined OS-Application

GetApplicationID

Syntax: `ApplicationType GetApplicationID (void);`

Input: None.

Output: None.

Description: Returns the current OS-Application Id.

Particularities: If no OS-Application is running, GetApplicationID returns INVALID_OSAPPLICATION.

Scalability: SC3, SC4

CallTrustedFunction

Syntax:
```
StatusType CallTrustedFunction
(TrustedFunctionIndexType
FunctionIndex,TrustedFunctionParameterRefType
FunctionParams);
```

Input: <FunctionIndex> - Index of the function to be called. It is genearted by SysGen in the form <service_name>ID.

<FunctionParams> - pointer to the parameters for the function, it is specified by the function to be called. If no parameters are provided, a NULL pointer has to be passed.

Output: depends on FunctionIndex

Description: switches the processor into privileged mode and calls the function <FunctionIndex> out of a list of implementation specific trusted functions, returns E_OK after completion.

Status:
- E_OK - no error.
- E_OS_SERVICEID - no function defined for this index.

Particularities: The called trusted function must conform to the following C prototype:
void TRUSTED_<service_name>(TrustedFunctionIndexType FunctionIndex, TrustedFunctionParameterRefType);

When function <FunctionIndex> is called it gets the same access rights as the associated trusted OS-application.

Scalbility: SC3, SC4

CheckISRMemoryAccess

Syntax:
```
AccessType CheckISRMemoryAccess (ISRType ISRID,
MemoryStartAddressType Address, MemorySizeType
Size);
```

Input: <ISRID> - ISR reference

<Address> - start of memory area

<Size> - size of memory area

Output: none

Description: checks the access rights for memory area specified.

Status:
- ACCESS - access allowed.
- NO_ACCESS - access from this OS-Application is not allowed or ISRID is invalid.

Particularities:

Scalbility: SC3, SC4

CheckTaskMemoryAccess

Syntax:
```
AccessType CheckTaskMemoryAccess (TaskType TaskID,
MemoryStartAddressType Address, MemorySizeType
Size);
```

Input: <TaskID> - Task reference

<Address> - start of memory area

<Size> - size of memory area

Output: none

Description: checks the access rights for memory area specified.

Status:
- ACCESS - access allowed.
- NO_ACCESS - access from this OS-Application is not allowed or TaskID is invalid.

Particularities:

Scalbility: SC3, SC4

CheckObjectAccess

Syntax:
```
ObjectAccessType CheckObjectAccess
(ApplicationType ApplID, ObjectTypeType
ObjectType,OSObjectType objectId);
```

Input: `<ApplID> - OS-Application Id, <ObjectType> - type of the object, <objectId> - Id of the object to be checked.`

Output: none

Description: Checks is the object is accessible from the given OS-Application.

Status:
- ACCESS - access allowed.
- NO_ACCESS - access from this OS-Application is not allowed.

Scalability: SC3, SC4

CheckObjectOwnership

Syntax:
```
ApplicationType CheckObjectOwnership
(ObjectTypeType ObjectType, OSObjectType
objectId);
```

Input: `<ObjectType> - type of the next parameter`

<objectId> - refernce to the object

Output: none

Description: If the specified object exists, CheckObjectOwnership returns the identifier of the OS-Application to which the object belongs.

Particularities: If the object to be examined is the RES_SCHEDULER the service returns INVALID_OSAPPLICATION**.**

Status:
- The service returns the OS-Application to which the object ObjectType belongs.
- INVALID_OSAPPLICATION - if the objectId is invalid.

Scalability: SC3, SC4

TerminateApplication

Syntax:
```
StatusType TerminateApplication(RestartType
RestartOption);
```

Input: none

Output: none

Description: Terminates the calling OS-Application - kills all Tasks and ISR, frees all other OS resources associated with the application.

If RestartOption equals RESTART, the configured RESTARTTASK of the terminated OS-Application is activated..

Particularities: Allowed on Task level, ISR level and in the Application-specific ErrorHook

The OS does not clear the interrupt request flags for killed ISRs.

Does not returns if there are no error.

Status:
- E_OS_CALLEVEL – called from wrong context.
- E_OS_VALUE - restart option is neither RESTART no NO_RESTART.

Scalability: SC3, SC4

Task Management Services

Data Types

The AUTOSAR OS establishes the following data types for the task management:

- *TaskType* – the abstract data type for task identification
- *TaskRefType* – the data type to refer variables of the *TaskType* data type. Reference or pointer to *TaskType* variable can be used instead of *TaskRefType* variable
- *TaskStateType* – the data type for variables to store the state of a task
- *TaskStateRefType* – the data type to refer variables of the *TaskStateType* data type. Reference or pointer to *TaskStateType* variable can be used instead of *TaskStateRefType* variable

Constants

The following constants are used within the AUTOSAR Operating System to indicate task states:

- *RUNNING* – constant of data type *TaskStateType* for task state *running*
- *WAITING* – constant of data type *TaskStateType* for task state *waiting*
- *READY* – constant of data type *TaskStateType* for task state *ready*
- *SUSPENDED* – constant of data type *TaskStateType* for task state *suspended*

The following constant is used within the AUTOSAR OS to indicate task:

- *INVALID_TASK* – constant of data type *TaskType* for undefined task

Conventions

Within an application of the AUTOSAR OS a task should be defined according to the following pattern:

```
TASK ( <name of task> )
{
...
}
```

The name of the task function will be generated from <*name of task*> by macro *TASK*.

Task Declaration

The constructional statement *DeclareTask* may be used for compatibility with OSEK versions. It may be omitted in application code.

Syntax: `DeclareTask( <name of task> );`

Input: <*name of task*> – reference to the task.

Description: This is a dummy declaration.

Particularities: There is no need in this declaration because all system objects are defined at system generation phase.

ActivateTask

Syntax: `StatusType ActivateTask( TaskType <TaskID> );`

Input: <*TaskID*> – a reference to the task.

Output: None.

Description: The specified task <*TaskID*> is transferred from the *suspended* state into the *ready* state.

Particularities: The service may be called both on the task level (from a task) and the interrupt level (from ISR).

In the case of calling from ISR, the operating system will reschedule tasks only after the ISR completion.

Status:
- Standard:
  - E_OK – no error.
  - E_OS_LIMIT – too many task activations of the specified task.
- Extended:
  - E_OS_ID – the task identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

TerminateTask

Syntax: `StatusType TerminateTask( void );`

Input: None.

Output: None.

Description: This service causes the termination of the calling task. The calling task is transferred from the *running* state into the *suspended* state.

Particularities: The resources occupied by the task shall be released before the call to *TerminateTask* service. If the call was successful, *TerminateTask* does not return to the call level and enforces a rescheduling. Ending

a task function without calling *TerminateTask* or *ChainTask* service is strictly forbidden.

If the system with extended status is used, the service returns in case of error, and provides a status which can be evaluated in the application.

There are the following limitations for BCC1 class if *FastTerminate* is set to TRUE: *TerminateTask* service shall be called in task function body from the function level; in STANDARD status this service does not return a status and can not be used in expressions.

The service call is allowed on task level only.

Status:
- Standard:
  - No return to call level.
- Extended:
  - E_OS_RESOURCE – the task still occupies resources.
  -
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

ChainTask

Syntax: `StatusType ChainTask( TaskType <TaskID> );`

Input: <*TaskID*> – a reference to the sequential succeeding task to be activated.

Output: None.

Description: This service causes the termination of the calling task. After termination of the calling task a succeeding task <*TaskID*> is transferred from the *suspended* state into the *ready* state. Using this service ensures that the succeeding task only starts to run after the calling task has been terminated.

Particularities: The resources occupied by the calling task shall be released before the call to *ChainTask* service. If the call was successful, *ChainTask* does not return to the call level and enforces a rescheduling. Ending

a task function without calling *TerminateTask* or *ChainTask* service is strictly forbidden.

If the succeeding task is identical to the current task, this does not result in multiple requests.

The service returns in case of error and provides a status which can be evaluated by the application.

There are the following limitations for BCC1 class if *FastTerminate* is set to TRUE: *ChainTask* service shall be called in task function body from the function level; in STANDARD status this service does not return a status and can not be used in expressions.

The service call is allowed on task level only.

Status:
- Standard:
  - No return to call level.
  - E_OS_LIMIT – too many activations of <*TaskID*>.
- Extended:
  - E_OS_ID – the task identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_RESOURCE – the calling task still occupies resources.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

Schedule

Syntax: `StatusType Schedule( void );`

Input: None.

Output: None.

Description: If there is a task in *ready* state with priority higher than assigned priority of calling task, the internal resource (if any) of the task is released, the calling task is put into the *ready* state and the higher-

priority task is transferred into the *running* state. Otherwise the calling task is continued.

Particularities: Rescheduling can only take place if an internal resource is assigned to the calling task during system generation (non-preemtable tasks are considered as a tasks with internal resource of highest priority). For these tasks, Schedule enables a processor assignment to other tasks with assigned priority not higher than the ceiling priority of the internal resource and higher than the assigned priority of the calling task. When returning from Schedule, the internal resource is taken again. This service has no influence on tasks with no internal resource assigned (preemptable tasks).

The service call is allowed on task level only.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_RESOURCE - calling task occupies resources.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

GetTaskID

Syntax: `StatusType GetTaskID( TaskRefType <TaskIDRef> );`

Input: None.

Output: <*TaskIDRef*> – a pointer to the variable contained reference to the task which is currently running. The service saves the task reference into the variable, that is addressed by pointer <*TaskIDRef*>. Reference to *TaskType* variable can be used instead of *TaskRefType* variable.

Description: This service returns reference to the task which is currently running. If there is no task in the *running* state, the service returns INVALID_TASK into the variable.

Particularities: The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <TaskIDRef> (only for SC3, SC4).

Conformance: BCC1, ECC1

GetTaskState

Syntax:
```
StatusType GetTaskState( TaskType <TaskID>,
                         TaskStateRefType <StateRef> );
```

Input: <*TaskID*> – a reference to the task.

Output: <*StateRef*> – a pointer to the state of task. The service saves the task state into the variable, that is addressed by pointer <*StateRef*>. Reference to *TaskStateType* variable can be used instead of *TaskStateRefType* variable.

Description: The service returns the state of the specified task <*TaskID*> (*running*, *ready*, *waiting*, *suspended*) at the time of calling *GetTaskState*.

Particularities: The service may be called both on the task level (from a task) and the interrupt level (from ISR). This service may be called from *ErrorHook*, *PreTaskHook*, *PostTaskHook* hook routines.

Within a full-preemptive system, calling this operating system service only provides a meaningful result if the task runs in an interrupt disabling state at the time of calling. When a call is made from a task in a full-preemptive system, the result may already be incorrect at the time of evaluation.

When the service is called for a task, which is multiply activated, the state is set to *running* if any instance of the task is running.

Status:
- Standard:
  - E_OK – no error.
- Extended:

- E_OS_ID – the task identifier is invalid.
- E_OS_ILLEGAL_ADDRESS - illegal <StateRef> (only for SC3, SC4).
- E_OS_ACCESS - insufficient access rights (for SC3, SC4).
- E_OS_CALLEVEL – call at not allowed context.
- E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

ISR Management Services

Data Types

ISRType - the abstract data type for ISR identification.

Constants

INVALID_ISR – constant of type *ISRType* for undefined ISR

For each defined ISR the constant with name <IsrName>PRIORITY equal to this ISR *PRIORITY* is defined.
This is an Freescale OS extension of the AUTOSAR OS

Conventions

Within an application an Interrupt Service Routine should be defined according to the following pattern:

```
ISR( <name of ISR>)
{
...
}
```

The keyword *ISR* is the macro for OS and compiler specific interrupt function modifier, which is used to generate valid code to enter and exit ISR.

ISR Declaration

The constructional statement *DeclareISR*[1] may be used for compatibility with OSEK versions.

Syntax: `DeclareISR( <name of ISR> );`

Input: `<name of ISR>` – reference to the ISR.

Description: This statement declares ISR function.

EnableAllInterrupts

Syntax: `void EnableAllInterrupts ( void );`

Input: None.

Output: None.

Description: This service restores the interrupts state saved by *DisableAllInterrupts* service. It can be called after *DisableAllInterrupts* only. This service is a counterpart of *DisableAllInterrupts* service, and its aim is the completion of the critical section of code. No API service calls are allowed within this critical section.

Particularities: The service may be called from an ISR and from the task level, but not from hook routines.

This service does not support nesting.

Status:

- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

DisableAllInterrupts

Syntax: `void DisableAllInterrupts ( void );`

[1] This declaration is not defined by *OSEK/VDX Operating System, v.2.2.2, 5 July 2004* specification. This is Freescale OS extension of the AUTOSAR OS.

Input: None.

Output: None.

Description: This service saves the current interrupts state and disables all hardware interrupts. This service is intended to start a critical section of the code. This section must be finished by calling the *EnableAllInterrupts* service. No API service calls are allowed within this critical section.

Particularities: The service may be called from an ISR and from the task level, but not from hook routines.

This service does not support nesting.

If EnableAllInterrupts service was not called after this service and before point of rescheduling then Freescale AUTOSAR OS dispatcher calls ErrorHook, if it is defined, with parameter E_OS_MISSINGEND.

Status:
- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

ResumeAllInterrupts

Syntax: `void ResumeAllInterrupts ( void );`

Input: None.

Output: None.

Description: This service restores the recognition status of all interrupts saved by *SuspendAllInterrupts* service.

Particularities: The service may be called from an ISR category 1 and category 2, from the alarm-callbacks and from the task level, from *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

This service is the counterpart of the *SuspendAllInterrupts* service, which must have been called before, and its aim is the completion of the critical section of code. No API service calls beside

SuspendAllInterrupts/ResumeAllInterrupts pairs and *SuspendOSInterrupts/ResumeOSInterrupts* pairs are allowed within this critical section.

SuspendAllInterrupts/ResumeAllInterrupts can be nested. In case of nesting pairs of the calls *SuspendAllInterrupts* and *ResumeAllInterrupts* the interrupt recognition status saved by the first call of *SuspendAllInterrupts* is restored by the last call of the *ResumeAllInterrupts* service.

If *STATUS* is set to EXTENDED the service call is ignored in case of a wrong sequence of interrupt management functions calls. This check is limited to 32 levels of nesting pairs of Suspend / Resume functions.

Status:
- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

SuspendAllInterrupts

Syntax: `void SuspendAllInterrupts ( void );`

Input: None.

Output: None.

Description: This service saves the recognition status of all interrupts and disables all interrupts for which the hardware supports disabling.

Particularities: The service may be called from an ISR category 1 and category 2, from alarm-callbacks and from the task level, from *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

This service is intended to protect a critical section of code from interruptions of any kind. This section must be finished by calling the *ResumeAllInterrupts* service. No API service calls beside *SuspendAllInterrupts/ResumeAllInterrupts* pairs and *SuspendOSInterrupts/ResumeOSInterrupts* pairs are allowed within this critical section.

If ResumeAllInterrupts service was not called after this service and before point of rescheduling then Freescale AUTOSAR OS dispatcher calls ErrorHook, if it is defined, with parameter E_OS_MISSINGEND.

Status:
- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

ResumeOSInterrupts

Syntax: `void ResumeOSInterrupts ( void );`

Input: None.

Output: None.

Description: This service restores the interrupts state saved by *SuspendOSInterrupts* service. It can be called after *SuspendOSInterrupts* only. This service is the counterpart of *SuspendOSInterrupts* service, and its aim is the completion of the critical section of code. No API service calls beside *SuspendAllInterrupts/ResumeAllInterrupts* pairs and *SuspendOSInterrupts/ResumeOSInterrupts* pairs are allowed within this critical section.

Particularities: The service may be called from an ISR and from the task level, but not from hook routines.

In case of nesting pairs of the calls *SuspendOSInterrupts* and *ResumeOSInterrupts* the interrupt recognition status saved by the first call of *SuspendOSInterrupts* is restored by the last call of the *ResumeOSInterrupts* service.

If *STATUS* is set to EXTENDED the service call is ignored in case of a wrong sequence of interrupt management functions calls. This check is limited to 32 levels of nesting pairs of Suspend / Resume functions.

If no ISRs of category 2 are defined, then this service does nothing.

Status:
- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

SuspendOSInterrupts

Syntax:
```
void SuspendOSInterrupts ( void );
```

Input: None.

Output: None.

Description: This service saves current interrupt state and disables all interrupts of category 2. Interrupts category 1 which priority is not higher than priority of any ISR category 2 are disabled also. This service is intended to start a critical section of the code. This section must be finished by calling the *ResumeOSInterrupts* service. No API service calls beside *SuspendAllInterrupts/ResumeAllInterrupts* pairs and *SuspendOSInterrupts/ResumeOSInterrupts* pairs are allowed within this critical section.

Particularities: The service may be called from an ISR and from the task level, but not from hook routines.

In case of nesting pairs of the calls *SuspendOSInterrupts* and *ResumeOSInterrupts* the interrupt status saved by the first call of *SuspendOSInterrupts* is restored by the last call of the *ResumeOSInterrupts* service.

If ResumeOSInterrupts service was not called after this service and before point of rescheduling then Freescale AUTOSAR OS dispatcher calls ErrorHook, if it is defined, with parameter E_OS_MISSINGEND.

If no ISRs of category 2 are defined, then this service does nothing.

Status:
- Standard:
  - None.
- Extended:
  - None.

Conformance: BCC1, ECC1

GetISRID

Syntax: `ISRType GetISRID (void);`

Input: None.

Output: None.

Description: Returns Id of active ISR.

Particularities: In the case of calling not from ISR returns INVALID_ISR.

Scalability: SC2 and SC4, but in the Freescale AUTOSAR OS the service is available in all classes.

DisableInterruptSource

Syntax:
```
StatusType DisableInterruptSource (ISRType
DisableISR);
```

Input: `DisableISR - identifier of ISR`

Output: none

Description: Disables the interrupt source for this ISR.

Particularities:

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the ISR identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Scalability: SC2, SC4

EnableInterruptSource

Syntax: `StatusType EnableInterruptSource (ISRType EnableISR);`

Input: `<EnableISR> - ISR to be enabled by the service`

Output: none

Description: If the arrival rate of ISR <EnableISR> is not reached, enables the source for this ISR. If the arrival rate of ISR <EnableISR> is reached then the interrupt source will be enabled at the start of the next timeframe.

Particularities:

Status:

- Standard:
  - E_OK – no error.
  - E_OS_NOFUNC - arrival rate of ISR is reached, it will be enabled by the OS at the start of the next timeframe..
- Extended:
  - E_OS_ID – the ISR identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Scalability: SC2, SC4

Resource Management Services

Data Types

The AUTOSAR OS establishes the following data type for the resource management:

- *ResourceType* – the abstract data type for referencing a resource

The only data type must be used for operations with resources.

Constants

- *RES_SCHEDULER* – constant of data type *ResourceType* corresponded to Scheduler Resource (see "Scheduler as a Resource")

Resource Declaration

The declaration statement *DeclareResource* may be used for compatibility with OSEK versions. It may be omitted in application code.

Syntax: `DeclareResource( <name of resource> );`

Input: <*name of resource*> – a reference to the resource.

Description: This is a dummy declaration.

Particularities: There is no need in this declaration because all system objects are defined at system generation phase.

GetResource

Syntax: `StatusType GetResource( ResourceType <ResID> );`

Input: <*ResID*> – a reference to the resource.

Output: None.

Description: This service changes current priority of the calling task or ISR according to ceiling priority protocol for resource management. *GetResource* serves to enter critical section in the code and blocks execution of any task or ISR which can get the resource <*ResID*>. A critical section must always be left using *ReleaseResource* within the same task or ISR.

Particularities: This function is fully supported in all Conformance Classes. It is Freescale OS extension of the AUTOSAR OS because OSEK/VDX specifies full support only beginning from BCC2.

Nested resource occupation is only allowed if the inner critical sections are completely executed within the surrounding critical section. Nested occupation of one and the same resource is forbidden.

The service call is allowed on task level and ISR level, but not in hook routines.

This service is not implemented if no *standard* resources are defined in the configuration file.

Regarding Extended Tasks, please note that *WaitEvent* within a critical section is prohibited.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the resource identifier is invalid.
  - E_OS_ACCESS – attempt to get resource which is already occupied by any task or ISR, or the assigned in OIL priority of the calling task or interrupt routine is higher than the calculated ceiling priority; OR insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

ReleaseResource

Syntax: `StatusType ReleaseResource( ResourceType <ResID> );`

Input: <*ResID*> – a reference to the resource.

Output: None.

Description: This call serves to leave the critical sections in the code that are assigned to the resources referenced by <*ResID*>. A *ReleaseResource* call is a counterpart of a *GetResource* service call. This service returns task or ISR priority to the level saved by corresponded *GetResource* service.

Particularities: This function is fully supported in all Conformance Classes. It is Freescale OS extension of the AUTOSAR OS because OSEK / VDX specifies full support only begining from BCC2.

Nested resource occupation is allowed only if the inner critical sections are completely executed within the surrounding critical section. Nested occupation of one and the same resource is forbidden.

The service call is allowed on task level and ISR level, but not in hook routines.

This service is not implemented if no *standard* resources are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the resource identifier is invalid.
  - E_OS_NOFUNC – attempt to release a resource which is not occupied by any task or ISR, or another resource has to be released before.
  - E_OS_ACCESS – attempt to release a resource which has a lower ceiling priority than the assigned in OIL priority of the calling task or interrupt routine. This error code is returned only if E_OS_NOFUNC was not returned; OR insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

Event Management Services

Data Types

The AUTOSAR Operating System establishes the following data types for the event management:

- *EventMaskType* – the data type of the event mask
- *EventMaskRefType* – the data type of the pointer to an event mask

Event Declaration

DeclareEvent declaration statement may be used for compatibility with OSEK versions. It may be omitted in application code.

Syntax: `DeclareEvent( <name of event> );`

Input: <*name of event*> – event name.

Description: This is a dummy declaration.

Particularities: There is no need in this declaration because all system objects are defined at system generation phase.

SetEvent

Syntax:
```
StatusType SetEvent( TaskType <TaskID>,
          EventMaskType <Mask> );
```

Input: <*TaskID*> – a reference to the task for which one or several events are to be set.

<*Mask*> – an event mask to be set.

Output: None.

Description: This service is used to set one or several events of the desired task according to the event mask. If the task was *waiting* for at least one of the specified events, then it is transferred into the *ready* state. The events not specified by the mask remain unchanged. Only an extended task which is not suspended may be referenced to set an event.

Particularities: It is possible to set events for the running task (task-caller).

The service call is allowed on task level and ISR level, but not in hook routines.

This service is not implemented if no events are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the task identifier is invalid.

- E_OS_ACCESS – the referenced task is not an Extended Task; OR insufficient access rights (for SC3, SC4).
- E_OS_STATE – the referenced task is in the suspended state.
- E_OS_CALLEVEL – call at not allowed context.
- E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: ECC1

ClearEvent

Syntax:
```
StatusType ClearEvent( EventMaskType <Mask> );
```

Input: <*Mask*> – an event mask to be cleared.

Output: None.

Description: The task which calls this service defines the event which has to be cleared.

Particularities: The system service *ClearEvent* can be called from extended tasks which own an event only.

This service is not implemented if no events are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ACCESS – the calling task is not an Extended Task.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: ECC1

GetEvent

Syntax:
```
StatusType GetEvent( TaskType <TaskID>,
                     EventMaskRefType <Event> );
```

Input: <*TaskID*> – a reference to the task whose event mask is to be returned.

Output: <*Event*> – a pointer to the variable of the return state of events.

Description: The event mask which is referenced to in the call is filled according to the state of the events of the desired task. Current state of events is returned but not the mask of events that task is waiting for.

It is possible to get event mask of the running task (task-caller).

Particularities: The referenced task must be an extended task and it can not be in *suspended* state.

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

This service is not implemented if no events are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the task identifier is invalid.
  - E_OS_ACCESS – the referenced task is not an Extended Task; OR insufficient access rights (for SC3, SC4).
  - E_OS_STATE – the referenced task is in the suspended state.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: ECC1

WaitEvent

Syntax: `StatusType WaitEvent( EventMaskType <Mask> );`

Input: <*Mask*> – an event mask to wait for.

Output: None.

Description: The calling task is transferred into the *waiting* state until at least one of the events specified by the mask is set. The task is kept the *running* state if any of the specified events is set at the time of the service call.

Particularities: This call enforces the rescheduling, if the wait condition occurs.

All resources occupied by the task must be released before *WaitEvent* service call.

The service can be called from extended tasks which own an event only.

This service is not implemented if no events are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ACCESS – the calling task is not an Extended Task.
  - E_OS_RESOURCE – the calling task occupies resources.
  - E_OS_CALLEVEL – a call at the interrupt level is not allowed.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: ECC1

Counter Management Services

Data Types and Identifiers

The following data types are established by AUTOSAR OS to operate with counters:

- *TickType*– the data type represent count value in ticks
- *TickRefType*– the data type of a pointer to the variable of data type *TickType*
- *CounterType*– the data type references a counter

- *CtrInfoRefType* – the data type of a pointer to the structure of data type *CtrInfoType*
- *CtrInfoType*– the data type represents a structure for storage of counter characteristics. This structure has the following elements:
  - *maxallowedvalue* – maximum possible allowed counter value in ticks
  - *ticksperbase* – number of ticks required to reach a counter-specific significant unit (it is a user constant, OS does not use it)
  - *mincycle* – minimum allowed number of ticks for the cycle parameter of *SetRelAlarm* and *SetAbsAlarm* services (only for system with Extended Status)

All elements of *CtrInfoType* structure have the data type *TickType*, and the structure looks like the following:

```
/* for EXTENDED status */
typedef CtrInfoType tagCIT;
struct tagCIT
{
    TickType maxallowedvalue;
    TickType ticksperbase;
    TickType mincycle;
};

/* for STANDARD status */
typedef CtrInfoType tagCIT;
struct tagCIT
{
    TickType maxallowedvalue;
    TickType ticksperbase;
};
```

NOTE CtrInfoType and CtrInfoRefType data types are not defined in the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. These are Freescale OS extension of the AUTOSAR OS.

System macros for time conversion

The conversation macros to convert counter ticks into real time are defined for Counters attached to System(Second)Timer. The format of the macros is *OS_TICKS2<Unit>_<Counter>(ticks)* whereas <Unit> is one of *NS* (nanoseconds), *US* (microseconds), *MS* (milliseconds) or *SEC* (seconds) and <*Counter*> is the name of the counter.

Constants

For all counters, the following constants are defined:

- *OSMAXALLOWEDVALUE_cname*
  Maximum possible allowed value of counter <cname> in ticks.
- *OSTICKSPERBASE_cname*
  Number of ticks required to reach a specific unit of counter <cname> (it is a user constant, OS does not use it).
- *OSMINCYCLE_cname*
  Minimum allowed number of ticks for a cyclic alarm of counter <cname>. This constant is not defined in *STANDARD* status

For system counters, which are always time counters, the special constants are provided by the operating system:

- *OSMAXALLOWEDVALUE / OSMAXALLOWEDVALUE2*
  maximum possible allowed value of the system / second timer in ticks (see also "Counter Definition")
- *OSTICKSPERBASE* / *OSTICKSPERBASE*2
  number of ticks required to reach a counter-specific value in the system / second counter (it is a user constants, not used by OS) (see also "Counter Definition")
- *OSTICKDURATION / OSTICKDURATION2*
  duration of a tick of the system / second counter in nanoseconds (defined automatically by *System Generator utility* (see also "CPU Related Attributes")
- *OSMINCYCLE / OSMINCYCLE2*
  minimum allowed number of ticks for a cyclic alarm attached to the system / second counter (only for system with Extended Status, see also "Alarm Definition")

NOTE *OSMAXALLOWEDVALUE2*, OSTICKSPERBASE2, *OSTICKDURATION2*, and OSMINCYCLE2 constants are not

defined in the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. These are Freescale OS extension of the AUTOSAR OS.

Counter Declaration

DeclareCounter declaration statement may be used for compatibility with previous OSEK versions. It may be omitted in application code.

Syntax:
```
DeclareCounter( <name of counter> );
```

Input: <*name of counter*> – reference to the counter.

Description: This is a dummy declaration.

Particularities: There is no need in this declaration because all system objects are defined at system generation phase.

InitCounter

Syntax:
```
StatusType InitCounter( CounterType <CounterID>,
                        TickType <Ticks> );
```

Input: <*CounterID*> – a reference to the counter.

<*Ticks*> – a counter initialization value in ticks.

Output: None.

Description: Sets the initial value of the counter with the value <*Ticks*>. After this call the counter will advance this initial value by one via the following call of *IncrementCounter*. If there are running attached alarms, then their state stays unchanged, but the expiration time becomes indeterminate.

Particularities: The service call is allowed on task level only.

This service is not implemented if no counters are defined in the configuration file.

The *InitCounter* service is not defined in the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003*

(AUTOSAR_SWS_OS) specification. This is Freescale OS extension of the AUTOSAR OS.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE – the counter initialization value exceeds the maximum admissible value.
  - E_OS_CALLEVEL – a call at interrupt level (not allowed).
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

IncrementCounter

Syntax:
```
StatusType IncrementCounter( CounterType <CounterID> );
```

Input: <*CounterID*> – a reference to the counter.

Output: None.

Description: The service increments the current value of the counter. If the counter value was equal to *maxallowedvalue* (see "Data Types and Identifiers") it is reset to zero.

If alarms are linked to the counter, the system checks whether they expired after this tick and performs appropriate actions (task activation and / or event setting).

Particularities: The service call is allowed on task level and ISR level, but not in hook routines.

The service shall not be used for counters assigned to System and Second timers.

This service is not implemented if no counters are defined in the configuration file.

Status:
- Standard:

  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid or belongs to hardware counter.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

GetCounterValue

Syntax:
```
StatusType GetCounterValue( CounterType
<CounterID>, TickRefType <TicksRef> );
```

Input: <*CounterID*> – a reference to the counter.

Output: <*TicksRef*> – a pointer to counter value in ticks. Reference to *TickType* variable can be used instead of *TickRefType* variable.

Description: The service provides the current value of the counter <*CounterID*> in ticks and saves it in the variable referenced by <*TicksRef*>.

Particularities: The service call is allowed on task and ISR level.

This service is not implemented if no counters are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal reference to <`TicksRef`> (only for SC3, SC4)

Conformance: BCC1, ECC1

GetElapsedCounterValue

Syntax: `StatusType GetElapsedCounterValue ( CounterType <CounterID>, TickRefType <PreviousValue>, TickRefType <Value> );`

Input: <*CounterID*> – a reference to the counter.
<*PreviousValue*> - a reference to the previously read value

Output: <*PreviousValue*> - updated with current read value.
<Value> - reference to the difference with previous read value in ticks.

Description: The service returns in the variable referensed by <Value> the number of elapsed ticks since the given <PreviousValue> value and updates <Value> with new counter value.

Particularities: The service call is allowed on task and ISR level.

If the timer already passed the <PreviousTick> value a second time, the returned result is invalid.

This service is not implemented if no counters are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE - given PreviousValue is not valid
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal reference to <PreviousValue> or <Value> (only for SC3, SC4)

Conformance: BCC1, ECC1

GetCounterInfo

Syntax:
```
StatusType GetCounterInfo( CounterType
<CounterID>,
CtrInfoRefType <InfoRef> );
```

Input: <*CounterID*> – a reference to the counter.

Output: <*InfoRef*> – a pointer to the structure of *CtrInfoType* data type. Reference to the *CtrInfoType* variable can be used instead of the *CtrInfoRefType* variable.

Description: The service provides the counter characteristics into the structure referenced by <*InfoRef*>. For a system counter special constants may be used instead of this service.

Particularities: The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines. The structure referenced by <*InfoRef*> consists of two elements in case of the "Standard Status", and of three elements in case of the "Extended Status".

This service is not implemented if no counters are defined in the configuration file.

The *GetCounterInfo* service is not defined in the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. This is Freescale OS extension of the AUTOSAR OS.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <InfoRef> (only for SC3, SC4).

Conformance: BCC1, ECC1

Alarm Management Services

Data Types and Identifiers

The following data types are established by AUTOSAR OS to operate with alarms:

- *TickType* – data type represents count values in ticks
- *TickRefType* – the data type of a pointer to the variable of data type *TickType*
- *AlarmBaseType* – the data type represents a structure for storage of counter characteristics. The elements of the structure are:
  - *maxallowedvalue* – maximum possible allowed counter value in ticks
  - *ticksperbase* – number of ticks required to reach a counter-specific significant unit
  - *mincycle* – minimum allowed number of ticks for the cycle parameter of *SetRelAlarm* and *SetAbsAlarm* services (only for system with Extended Status)

  All elements of the structure are of data type *TickType.*
- *AlarmBaseRefType*– the data type references data corresponding to the data type *AlarmBaseType*
- *AlarmType*– the data type represents an alarm object

Constants

OSMINCYCLE – minimum allowed number of ticks for a cyclic alarm (only for system with Extended Status)

Alarm Declaration

The declaration statement *DeclareAlarm* may be used for compatibility with previous OSEK versions. It may be omitted in application code.

Syntax: `DeclareAlarm( <name of alarm> );`

Input: <*name of alarm*> – reference to the alarm.

Description: This is a dummy declaration.

Particularities: There is no need in this declaration because all system objects are defined at system generation phase.

GetAlarmBase

Syntax:
```
StatusType GetAlarmBase( AlarmType <AlarmID>,
                         AlarmBaseRefType <InfoRef> );
```

Input: <*AlarmID*> – a reference to the alarm.

Output: <*InfoRef*> – a pointer to the structure <*InfoRef*> with returned values of the alarm base. Reference to *AlarmBaseType* variable can be used instead of *AlarmBaseRefType* variable.

Description: The service returns the alarm base characteristics into the structure pointed by <*InfoRef*>. The return value is a structure in which the information of data type *AlarmBaseType* is stored.

Particularities: The structure consists of two elements in case of the “Standard Status”, and of three elements in case of the “Extended Status”.

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

This service is not implemented if no alarms are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the alarm identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <InfoRef> (only for SC3, SC4).

Conformance: BCC1, ECC1

GetAlarm

Syntax:
```
StatusType GetAlarm( AlarmType <AlarmID>,
                     TickRefType <TicksRef> );
```

Input: <*AlarmID*> – a reference to the alarm.

Output: <*TicksRef*> – a pointer to a variable which gets a relative value in ticks before the alarm expires. Reference to *TickType* variable can be used instead of *TickRefType* variable.

Description: This service calculates the time in ticks before the alarm expires. If the alarm is not in use, then returned value is not defined.

Particularities: It is up to the application to decide whether for example an alarm may still be useful or not.

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

This service is not implemented if no alarms are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
  - E_OS_NOFUNC – the alarm is not in use.
- Extended:
  - E_OS_ID – the alarm identifier is invalid.
  - E_OS_ACCESS – insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <TicksRef> (only for SC3, SC4).

Conformance: BCC1, ECC1

SetRelAlarm

Syntax:
```
StatusType SetRelAlarm( AlarmType <AlarmID>,
     TickType <Increment>, TickType <Cycle> );
```

Input: <*AlarmID*> – a reference to the alarm;

<*Increment*> – an alarm initialization value in ticks;

<*Cycle*> – an alarm cycle value in ticks in case of cyclic alarm. In case of single alarms, the value cycle has to be equal zero.

Output: None.

Description: The system service occupies the alarm <*AlarmID*> element. After <*Increment*> counter ticks have elapsed, the task assigned to the alarm <*AlarmID*> is activated or the assigned event (only for Extended Tasks) is set.

If <*Cycle*> is unequal to 0, the alarm element is logged on again immediately after expiry with the relative value <*Cycle*>. Otherwise, the alarm triggers only once.

If relative value<*Increment*> equals 0, the alarm expires immediately and assigned task becomes *ready* before the system service returns to the calling task or ISR.

Particularities: Allowed on task level and ISR level, but not in hook routines.

If alarm is already in use, the service call is ignored. To change values of alarms already in use the alarm has to be cancelled first.

This service is not implemented if no alarms are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
  - E_OS_STATE – the alarm is already in use.
  - E_OS_VALUE - Value of <Increment> is equal to 0.
- Extended:
  - E_OS_ID – the alarm identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE - an alarm initialization value is outside of the admissible limits (lower than zero or greater than the maximum allowed value of the counter), or alarm cycle value is unequal to 0 and outside of the admissible counter limits (less than the minimum cycle value of the

counter or greater than the maximum allowed value of the counter).

- E_OS_CALLEVEL – call at not allowed context.
- E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: • BCC1, ECC1.

SetAbsAlarm

Syntax:
```
StatusType SetAbsAlarm( AlarmType <AlarmID>,
           TickType <Start>, TickType <Cycle> );
```

Input: <*AlarmID*> – a reference to the alarm;

<*Start*> – an absolute value in ticks;

<*Cycle*> – an alarm cycle value in ticks in case of cyclic alarm. In case of single alarms, cycle has to be equal zero.

Output: None.

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.

If <*Cycle*> is unequal to 0, the alarm element is logged on again immediately after expiry with the relative value <*Cycle*>. Otherwise, the alarm triggers only once.

If the absolute value <Start> is very close to the current counter value, the alarm may expire and assigned task may become *ready* before the system service returns to the calling task or ISR.

If the absolute value <Start> have been reached before the service call, the alarm will only expire when <Start> value will be reached again.

Particularities: Allowed on task level and ISR level, but not in hook routines.

If alarm is already in use, the service call is ignored. To change values of alarms already in use the alarm has to be cancelled first.

This service is not implemented if no alarms are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error;
  - E_OS_STATE – the alarm is already in use.
- Extended:
  - E_OS_ID – the alarm identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE - an alarm absolute value is outside of the admissible limits (lower than zero or greater than the maximum allowed value of the counter), or alarm cycle value is unequal to 0 and outside of the admissible counter limits (less than the minimum cycle value of the counter or greater than the maximum allowed value of the counter).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance:
- BCC1, ECC1.

CancelAlarm

Syntax: `StatusType CancelAlarm( AlarmType <AlarmID> );`

Input: <*AlarmID*> – a reference to the alarm.

Output: None.

Description: The service cancels the alarm (transfers it into the stop state).

Particularities: The service is allowed on task level and in ISR, but not in hook routines.

This service is not implemented if no alarms are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
  - E_OS_NOFUNC – the alarm is not in use.

- Extended:
  - E_OS_ID – the alarm identifier is invalid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

ScheduleTable Management Services

Data Types

- ScheduleTableType - identifier of ScheduleTable.
- ScheduleTableStatusType - the status of a schedule table;
- ScheduleTableStatusRefType - reference to a variable of the data type ScheduleTableStatusType;
- GlobalTimeTickType - the global time source type.

Constants

The following constants of ScheduleTableStatusType type are defined:

- SCHEDULETABLE_STOPPED
- SCHEDULETABLE_NEXT
- SCHEDULETABLE_WAITING
- SCHEDULETABLE_RUNNING
- SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS

For meaning of this constants please refer to GetScheduleTableStatus description.

StartScheduleTableRel

Syntax:
```
StatusType StartScheduleTableRel(ScheduleTableType ScheduleTableID, TickType Offset);
```

Input: <ScheduleTableID> - schedule table to be started.

<Offset> - relative tick value between now and the first action point.

Output: None.

Description: If the input parameters are valid, the service starts the processing schedule table <ScheduleTableID> at its first expiry point after offset <Offset> ticks have elapsed and returns E_OK.

Particularities: Allowed on Task and ISR level.

Status:
- Standard:
  - E_OK.
  - E_OS_STATE - Schedule table is already started.
- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid or implicitly synchronized.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE - <Offset> is greater than MAXALLOWEDVALUE.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

StartScheduleTableAbs

Syntax:
```
StatusType StartScheduleTableAbs(ScheduleTableType ScheduleTableID, TickType TickValue);
```

Input: <ScheduleTableID> - schedule table to be started.

<TickValue> - absolute tick value of the first action point.

Output: None.

Description: If the input parameters are valid, the service starts the processing schedule table <ScheduleTableID> at its first expiry point after underlaying counter reaches <TickValue> and returns E_OK.

Particularities: Allowed on Task and ISR level.

Status:
- Standard:
  - E_OK.

  - E_OS_STATE - Schedule table is already started.
- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid or implicitly synchronized.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_VALUE - <Offset> is greater than MAXALLOWEDVALUE.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

StartScheduleTableSynchron

Syntax:
```
StatusType
StartScheduleTableSynchron(ScheduleTableType
ScheduleTableID);
```

Input: <ScheduleTableID> - schedule table to be started.

<TickValue> - absolute tick value of the first action point.

Output: None.

Description: If its input parameters are valid, the service sets the state of <ScheduleTableID> to SCHEDULETABLE_WAITING and start the processing of schedule table <ScheduleTableID> after the synchronization count of the schedule table is set via SyncScheduleTable().

Particularities: Allowed on Task and ISR level.
Available in all Scalability Classes - it is Freescale OS extension of the AUTOSAR OS specification, which requires this service only in SC2, SC4.

Status:
- Standard:
  - E_OK.
  - E_OS_STATE - Schedule table is already started.
- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).

- E_OS_CALLEVEL – call at not allowed context.
- E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

StopScheduleTable

Syntax: `StatusType StopScheduleTable(ScheduleTableType ScheduleTableID);`

Input: <ScheduleTableID> - schedule table to be stopped.

Output: None.

Description: The service stops a *Schedule Table* processing immediately.

Particularities: Allowed on Task and ISR level.

Status:

- Standard:
  - E_OK.
  - E_OS_NOFUNC - Schedule table is already started.
- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

NextScheduleTable

Syntax: `StatusType NextScheduleTable(ScheduleTableType ScheduleTableIDCurrent, ScheduleTableType ScheduleTableIDNext);`

Input: <ScheduleTableIDCurrent> - schedule table.

<ScheduleTableIDNext> - schedule table that provides its series of expiry points.

Output: None.

Description: If the input parameters are valid and the <ScheduleTableIDCurrent> is running, the service plans the start of

the schedule table <ScheduleTableIDNext> after <ScheduleTableCurrent> reaches its period and returns E_OK. If its input parameters are valid and the <ScheduleTableIDCurrent> is running and NextScheduleTable() was previously successsfuly called, the new <ScheduletableIDNext> replace the previous next value. The current ScheduleTable is stopped when it's period is reached.

Particularities: Allowed on Task and ISR level.

Status:
- Standard:
  - E_OK.
  - E_OS_NOFUNC - <ScheduleTableIDCurrent> was not started.
- Extended:
  - E_OS_ID - <ScheduleTableIDCurrent> or <ScheduleTableIDNext> is not valid or belongs to different counters.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_STATE - <ScheduleTableIDNext> is started
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

SyncScheduleTable

Syntax:
```
StatusType SyncScheduleTable(ScheduleTableType
ScheduleTableID,  GlobalTimeTickType GlobalTime);
```

Input: <ScheduleTableID> - the schedule table identifier.

<GlobalTime> - the current value of global time.

Output: None.

Description: This service provides the OS with current global time. It is used to synchronize the processing of schedule table to global time.

Particularities: The service is applicable for not started ScheduleTable.
Allowed on Task and ISR level.

Available in all Scalability Classes - it is Freescale OS extension of the AUTOSAR OS specification, which requires this service only in SC2, SC4.

The service has no effect for non-perioric ScheduleTable with SYNCSTRATEGY = IMPLICIT.

Status:
- Standard:
  - E_OK.
- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_STATE - the ScheduleTable is not started (its state is SCHEDULETABLE_STOPPED or SCHEDULETABLE_NEXT).
  - E_OS_VALUE – <Value> is more or equal to ScheduleTable period.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

SetScheduleTableAsync

Syntax: `StatusType SetScheduleTableAsync(ScheduleTableType ScheduleID);`

Input: <ScheduleTableID> - the schedule table identifier.

Output: None.

Description: This service set the synchronization status of the <ScheduleTableID> to asynchronous. If this service is called for a running schedule table the OS contiunue it's processing.

Particularities: Allowed on Task and ISR level.

Available in all Scalability Classes - it is Freescale OS extension of the AUTOSAR OS specification, which requires this service only in SC2, SC4.

Status:
- Standard:
  - E_OK.

- Extended:
  - E_OS_ID - <ScheduleTableID> is not valid.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

GetScheduleTableStatus

Syntax:
```
StatusType GetScheduleTableStatus
(ScheduleTableType ScheduleID,
ScheduleTableStatusRefType ScheduleStatus);
```

Input: <ScheduleTableID> - the schedule table identifier.

Output: <ScheduleStatus> - reference to the variable of ScheduleStatusType.

Description: If schedule table <ScheduleTableID> is not yet started, this service passes back SCHEDULETABLE_STOPPED via the reference parameter <ScheduleStatus> and return E_OK. If the schedule table <ScheduleTableID> is configured with explicit synchronization strategy and no synchronization count was provided to the OS, the service returns SCHEDULETABLE_WAITING via the reference parameter <ScheduleStatus> and returns E_OK. If the schedule table <ScheduleTableID> was used in a NextScheduleTable () call and waits for the end of the current schedule table, this service SCHEDULETABLE_NEXT via the reference parameter <ScheduleStatus> and return E_OK. If schedule table <ScheduleTableID> started and synchronous, the service passes back SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS via the reference parameter <ScheduleStatus> and returns E_OK. If schedule table <ScheduleTableID> is started, but is not synchronous (deviation is not within the precision interval or the global time is not available), the service passes back SCHEDULETABLE_RUNNUING via the reference parameter <ScheduleStatus> and returns E_OK.

Particularities: none

Status:
- Standard:
  - E_OK.
- Extended:

– E_OS_ID - <ScheduleTableID> is not valid.
– E_OS_ACCESS - insufficient access rights (for SC3, SC4).
– E_OS_CALLEVEL – call at not allowed context.
– E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
– E_OS_ILLEGAL_ADDRESS - illegal <ScheduleStatus> (only for SC3, SC4).

Communication Management Services

OSEK COM is not part of AUTOSAR OS specification. It is supported in the Freescale AUTOSAR OS for compatibility with OSEKturbo implementations.

All communication services should be used after call to the *StartCOM* service, but it's behaviour is identical before and after call to *StartCOM*, except *GetCOMApplicationMode.*

Data Types and Identifiers

The following types are used in the OSEK COM to operate with internal messages:

- *MessageIdentifier* - type for OSEK COM message object identifier.
- *ApplicationDataRef* - type for reference to the application message data:
- *FlagValue* - type for the current state of a message flag.
- *COMApplicationModeType* - type for COM application mode.
- *COMShutdownModeType* - type for COM shutdown mode.
- *COMServiceIdType* - type for identifier of an OSEK COM service.

Constants

COM_SHUTDOWN_IMMEDIATE – the allowed COM shutdown mode.

SendMessage

Syntax:
```
StatusType SendMessage ( MessageIdentifier
          <Message>, ApplicationDataRef <Data> );
```

Input: <*Message*> – message identifier.

<*Data*> – reference to the message data to be sent.

Output: None.

Description: This service updates the message objects of receiving messages for which this Message is sending with the Data given by <Data>. It activates a notification mechanism of all receiving messages, if configured. The service updates also the status information of the message object accordingly.

Particularities: If the queue of receiving message is full and a new message data arrives, this message data will be lost, but notification will be performed.

OS (not COM) ErrorHook is called (if it is configured in OS) when attempt to activate not suspended task or set event for suspended task is performed as a result of message notification, but no error code returned.

The service is allowed in Tasks, ISRs and in COMcallback functions.

This service is not implemented if no messages are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_COM_ID – the message name <Message> is invalid or belongs to the receiving message.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <Data> (only for SC3, SC4).

Conformance: BCC1, ECC1

ReceiveMessage

Syntax:
```
StatusType ReceiveMessage ( MessageIdentifier
          <Message>, ApplicationDataRef <Data>);
```

Input: <*Message*> – message identifier.

Output: <*Data*> – reference to the application data area for received data.

Description: This service delivers the message data associated with the message object <Message> to the applications message copy referenced by <Data>. It resets all flags associated with <Message>.

Particularities: Unqueued messages:

- The service returns the current message.
- If no new message has been received since the last call to ReceiveMessage the current message is returned.

The service is allowed in Tasks, ISRs and in COMcallback functions.

This service is not implemented if no messages are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
  - E_COM_NOMSG - the queued message identified by <*Message*> is empty.
  - E_COM_LIMIT - an overflow of the FIFO of the queued message occurred since the last call to ReceiveMessage for that particular <*Message*>.
- Extended:
  - E_COM_ID – the message name <Message> is invalid or belongs to the sending message.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <Data> (only for SC3, SC4).

Conformance: BCC1, ECC1

GetMessageStatus

Syntax:
```
StatusType GetMessageStatus (
                MessageIdentifier <Message> )
```

Input: <*Message*> – message identifier.

Output: None.

Description: The service return the current status of the message object <Message>.

Particularities: The service is allowed in Tasks, ISRs and in COMcallback functions.

This service calls ErrorHook only if it returns E_COM_ID.

This service is not implemented if no messages are defined in the configuration file.

Status:

- Standard:
  - E_COM_NOMSG - the FIFO of the queued message identified by <*Message*> is empty.
  - E_COM_LIMIT - an overflow of the queued message identified by <Message> occurred since the last call to the ReciveMessage service for that particular message.
  - E_OK - none of conditions specified above fullfiled.
- Extended:
  - E_COM_ID - the message name <Message> is invalid, i.e. it does not belongs to the queued message.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

ReadFlag

Syntax:
```
FlagValue ReadFlag_<FlagName>()
```

Input: None.

Output: None.

Description: The service returns COM_TRUE if the flag <FlagName> is set, otherwise it returns COM_FALSE.

Particularities: The service is implemented as a macro generated by Sysgen.

The service is allowed on task level, ISR level and in COMcallback function.

This service is implemented if there are at least one message with NOTIFICATION = FLAG.

Return value:
- FlagValue – state of the flag <FlagName>

Conformance: BCC1, ECC1

ResetFlag

Syntax: `StatusType ResetFlag_<FlagName>()`

Input: None.

Output: None.

Description: The service cleares the state of flag <FlagName>.

Particularities: The service is implemented as a macro generated by Sysgen.

The service is allowed on task level, ISR level and in COMcallback function.

This service is implemented if there are at least one message with NOTIFICATION = FLAG.

Status:
- Standard and Extended:
  - E_OK - no error.

Conformance: BCC1, ECC1

StartCOM

Syntax: `StatusType StartCOM( COMApplicationModeType Mode )`

Input: Mode - COM application mode.

Output: None.

Description: The StartCOM service starts the OS communication module. This routine performs the initialization of OSEK COM implementation specific internal data and then calls user-provided StartCOMExtension function, if configured.

Particularities: The service returns E_OK or value returned by StartCOMExtension; it does not calls ErrorHook despite of returned value.

StartCOMExtension is called with imterrutps of category 2 disabled.

The service is allowed only on task level.

This service is not implemented if no messages are defined in the configuration file.

Status:
- Standard:
  - E_OK – the initialization completed successfully.
  - Value returned by StartCOMExtension, if it is configured.
- Extended:
  - E_COM_ID - the parameter <Mode> is invalid.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

StopCOM

Syntax:
```
StatusType StopCOM( COMShutdownModeType
                                ShutdownMode);
```

Input: ShutdownMode;
the only allowed value is *COM_SHUTDOWN_IMMEDIATE.*

Output: None.

Description: This services causes all OSEK COM activity to cease and all resources used by COM to be left in an inactive state.

Particularities: The service is allowed in Tasks and ISRs.

This service is not implemented if no messages are defined in the configuration file.

Status:
- Standard:
  - E_OK – no error.
- Extended:
  - E_COM_ID - the parameter <ShutdownMode> is invalid.
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.

Conformance: BCC1, ECC1

GetCOMApplicationMode

Syntax:
```
COMApplicationModeType GetCOMApplicationMode(
                                              void)
```

Input: None

Output: None.

Description: This services returns the value used as argument for the last call to StartCOM.

Particularities: The service is allowed in Tasks, ISRs and in COMcallback functions.

This service is not implemented if no messages are defined in the configuration file.

Return value:
- Current COM application mode.

Conformance: BCC1, ECC1

InitMessage

Syntax:
```
StatusType  InitMessage( MessageIdentifier
             Message, ApplicationDataRef DataRef )
```

Input: <Message> - message identifier.

<DataRef> - reference to the message data to be used as initial message value.

Output: None.

Description: Unqueued messages:

- data pointed by <DataRef> is copied to message object.

Queued messages:

- The number of messages in queue is set to 0, parameter <DataRef> is not used.

Particularities: All unqueued messages belonging to the same sending message share one message object.

The service is allowed in Tasks, ISRs and in COMcallback functions.

This service is not implemented if no messages are defined in the configuration file.

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_COM_ID - the message name <Message> is invalid, i.e. it does not belongs to the receiving message.
  - E_OS_ACCESS - insufficient access rights (for SC3, SC4).
  - E_OS_CALLEVEL – call at not allowed context.
  - E_OS_DISABLEDINT - call when interrupts are disabled by OS services.
  - E_OS_ILLEGAL_ADDRESS - illegal <DataRef> (only for SC3, SC4).

Conformance: BCC1, ECC1

Debugging Services

These services are not defined by *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. This is Freescale OS extension of the AUTOSAR OS.

GetRunningStackUsage

Syntax:

```
unsigned short GetRunningStackUsage ( void );
```

Input: None.

Output:
- amount of stack used by running task in bytes.
- 0xFFFF if there is not any running task or the task uses “single stack”.

Description: The service returns amount of stack used by running task in bytes. The service returns 0xFFFF for basic task in SC1, when single stack is used.

Particularities: The service is implemented if the value of the *STACKMONITORING* attribute is *TRUE* or the value of the *DEBUG_LEVEL* attribute is greater than 0.

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

Conformance: BCC1, ECC1

GetStackUsage

Syntax: `unsigned short GetStackUsage ( TaskType <TaskID> );`

Input: <*TaskID*> - a reference to the task.

Output:
- amount of stack used by task <TaskID> in bytes.
- 0xFFFF in SC1 if the task is basic (uses “single stack”).

Description: The service returns stack usage by task <*TaskID*> in bytes.

Particularities: The service is implemented if the value of the *STACKMONITORING* attribute is set *TRUE* or the value of the *DEBUG_LEVEL* attribute is greater than 0.

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

Conformance: BCC1, ECC1

GetTimeStamp

Syntax: `unsigned short GetTimeStamp (void );`

Input: None.

Output: System Counter current value.

Description: The service returns current value of the System Counter (the counter which is attached to the System Timer).

Particularities: The service is implemented if the value of the *DEBUG_LEVEL* attribute is greater than 0 and the System Timer is defined in the application.

The service is implemented as macro, so it can not be used in non-Trusted Applications

The service call is allowed on task level, ISR level and in *ErrorHook*, *PreTaskHook* and *PostTaskHook* hook routines.

Conformance: BCC1, ECC1

Operating System Execution Control

Data Types

The AUTOSAR OS establishes the following data type for operation mode representation:

- *StatusType* – the data type represent variable for saving system status
- *AppModeType* – the data type represents the operating mode

Constants

The following constant is used within the AUTOSAR OS to indicate default application mode:

- *OSDEFAULTAPPMODE* – constant of data type *AppModeType*. The constant is assigned to one of the application modes defined in the OIL file. This constant is always a valid parameter for *StartOS* service.

The following constants are used within the AUTOSAR Operating System to indicate system status. All of them have type *StatusType*. Status meaning is specified in service descriptions:

- E_OK
- E_OS_ACCESS
- E_OS_CALLEVEL

- E_OS_ID
- E_OS_LIMIT
- E_OS_NOFUNC
- E_OS_RESOURCE
- E_OS_STATE
- E_OS_VALUE
- E_OS_SERVICEID
- E_OS_RATE
- E_OS_ILLEGAL_ADDRESS
- E_OS_MISSINGEND
- E_OS_DISABLEDINT
- E_OS_STACKFAULT
- E_OS_PROTECTION_MEMORY
- E_OS_PROTECTION_TIME
- E_OS_PROTECTION_LOCKED
- E_OS_PROTECTION_EXCEPTION
- E_OS_PROTECTION_ARRIVAL
- E_OS_SYS_FATAL
- E_COM_ID
- E_COM_NOMSG
- E_COM_LIMIT
- E_OS_SYS_ORDER[1]

GetActiveApplicationMode

Syntax: `AppModeType GetActiveApplicationMode( void );`

Input: None.

Output: Current application mode.

[1] E_OS_SYS_xxx are not defined in the *AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)* specification. This is Freescale OS extension of the AUTOSAR OS.

Description: This service returns the current application mode.

Particularities: Allowed on task level, on ISR level and in all hook routines.

Conformance: BCC1, ECC1

StartOS

Syntax: `void StartOS( AppModeType <Mode> );`

Input: <*Mode*> – an operating mode.

Output: None.

Description: This service starts the operation system in a specified mode. If a *StartupHook* is configured, the hook routine is always called before starting the application.

Particularities: Allowed outside of the operating system only.
In Freescale AUTOSAR OS this service returns to the caller only if it is called with the wrong parameter *Mode*.

All interrupts are disabled inside the service.

Conformance: BCC1, ECC1

ShutdownOS

Syntax: `void ShutdownOS( StatusType <Error> );`

Input: <*Error*> – a code of the error occurred.

Output: None.

Description: The service aborts the overall operating system.

If a task is in the *running* state, *PostTaskHook* is not called.

If a *ShutdownHook* is configured, the hook routine is always called (with <*Error*> as argument) before shutting down the operating system. If *ShutdownHook* returns, the operating system enters endless loop. (see "System Shutdown").

Particularities: *ShutdownOS* is ignored if called from non-Trusted OS-Application.

ShutdownOS runs in connection with the currently active context, which may be unknown to the user. Thus only

GetActiveApplicationMode and *GetApplicationID* services are allowed within the *ShutdownHook* routine.

All interrupts are disabled inside the service.

Allowed on task level, on ISR level and in *StartupHook* and *ErrorHook* hook routines.

Conformance: BCC1, ECC1

Hook Routines

The hook routines is called by the OS, but shall be implemented by the User, if configured.

Interrupts of category 2 and TP interrupt are disabled in all hook routines.

See "Hook Routines" for general description of hook routines.

In the classes with memory protection (SC3, SC4) the Application specific hooks may be defined. This hooks are called in the same cases as System-wide hooks, but in the context and with access rigths of the given OS-Application. The Application specific hooks with following names may be defined by User:

- ErrorHook_<AppId>()
- StartupHook_<AppId>()
- ShutdownHook_<AppId>()

where <AppId> is the name of the OS-Application

ProtectionHook

Syntax:
```
ProtectionReturnType ProtectionHook( StatusType error );
```

Input: <*Error*> – a code of the error occurred.

Output: None.

Description: To implement *ProtectionHook* the routine with the name 'ProtectionHook' shall be defined in the user's code. This hook is called if serious error occurs. E.g. exceeding the worst case execution time or violating against the memory protection.

Return:
- PRO_TERMINATETASKISR - the OS shall terminate running Task or ISR and perform an appropriate cleanup.
- PRO_TERMINATEAPPL - the OS shall kill active OS-Application.
- PRO_TERMINATEAPPL_RESTART - the OS shall kill active OS-Application and then restart RESTARTTASK of this OS-Application.
- PRO_SHUTDOWN - the OS shall perform shutdown.
- PRO_IGNORE - the OS shall do nothing, allowed only in case of *E_OS_PROTECTION_ARRIVAL*

Particularities: This hook is not called if the system configuration option *PROTECTIONHOOK* is set to *FALSE*. In this case the shutdown is performed in case of any violation.

Conformance: BCC1, ECC1 within SC2, SC4

ErrorHook

Syntax:
```
void ErrorHook( StatusType <Error> );
```

Input: <*Error*> – a code of the error occurred.

Output: None.

Description: To implement *ErrorHook* the routine with the name 'ErrorHook' shall be defined in the user's code. This routine is called by the operating system at the end of a system service which has a return value not equal to E_OK if not specified other (see also "COMErrorHook"). It is called before returning from the service. This hook is also called from OS dispatcher when an error is detected during task activation or event setting as a result of an alarm expiry or a message arrival, or if a rescheduling occurs after *Suspend(OS/All)Interrupts* before call to appropriate Resume function.

Particularities: *ErrorHook* can not be nested. Therefore the error hook is not called, if a system service called from *ErrorHook* and does not return E_OK as a status value. OS *ErrorHook* also can not be nested with *COMErrorHook*.

If ErrorHook is called from inside the service, called from other hook then it inherits the context of enclosing hook and the restrictions for that hook are applied.

This hook is not called if the system configuration option *ERRORHOOK* is set to *FALSE*.

There is a set of special macros to get the ID of service where error has occured and it's first argument - see "Macros for ErrorHook".

Conformance: BCC1, ECC1

COMErrorHook

Syntax:
```
void COMErrorHook( StatusType <Error> );
```

Input: <*Error*> – a code of the error occurred.

Output: None.

Description: To implement *COMErrorHook* the routine with the name 'COMErrorHook' shall be defined in the user's code. This routine is called by the operating system at the end of a COM service which has a return value E_COM_xxx. If the return value is E_OS_xxx then the *ErrorHook* is called if not specified other.

Particularities: *COMErrorHook* can not be nested. Therefore the error hook is not called, if a service is called from *COMErrorHook* and does not return E_OK as a status value. *COMErrorHook* also can not be nested with OS *ErrorHook*.

This hook is not called if the COM configuration option *COMERRORHOOK* is set to *FALSE*.

If COMErrorHook is called from inside the service, called from other hook then it inherits the context of enclosing hook and the restrictions for that hook are applied.

There is a set of special macros to get the ID of service where error has occured and it's first argument - see "Macros for ErrorHook".

Conformance: BCC1, ECC1

PreTaskHook

Syntax:
```
void PreTaskHook( void );
```

Input: None.

Output: None.

Description: To implement *PreTaskHook* the routine with name 'PreTaskHook' shall be defined in user's code. This hook routine is called by the operating system before executing a new task, but after the transition of the task to the running state (to allow evaluation of the task ID by *GetTaskID* services). This hook is called from the scheduler when it passes control to the given task. It may be used by the application to trace the sequences and timing of tasks' execution.

Particularities: This hook is not called if the system configuration option *PRETASKHOOK* is set to FALSE.

Conformance: BCC1, ECC1

PostTaskHook

Syntax:
```
void PostTaskHook( void );
```

Input: None.

Output: None.

Description: To implement *PostTaskHook* the routine with name 'PostTaskHook' shall be defined in user's code. This hook routine is called by the operating system after executing the current task, but before leaving the task's running state (to allow evaluation of the task ID by *GetTaskID*).This hook is called from the scheduler when it switches from the current task to another. It may be used by the application to trace the sequences and timing of tasks' execution.

Particularities: *PostTaskHook* is not called if running task is exist and OS is aborted by *ShutdownOS* service.

This hook is not called if the system configuration option *POSTTASKHOOK* is set to *FALSE*.

Status: None.

Conformance: BCC1, ECC1

PreIsrHook

Syntax:
```
void PreIsrHook( void );
```

Input: None.

Output: None.

Description: To implement *PreIsrHook* the routine with name 'PreIsrHook' shall be defined in user's code. This hook routine is called by the operating system before executing a ISR category 2 or System Timer ISR with all interrupts disabled. It may be used for debugging purposes only. This hook is not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.

Particularities: This hook is not called if the system configuration option *IsrHooks* is set to FALSE.

Conformance: BCC1, ECC1

PostIsrHook

Syntax: `void PostIsrHook( void );`

Input: None.

Output: None.

Description: To implement *PostIsrHook* the routine with name 'PostIsrHook' shall be defined in user's code. This hook routine is called by the operating system after executing a ISR category 2 or System Timer ISR with all interrupts disabled. It may be used for debugging purposes only. This hook is not defined in AUTOSAR OS v.3.0 specification, it is Freescale OS extension of the AUTOSAR OS.

Particularities: This hook is not called if the system configuration option *IsrHooks* is set to *FALSE*.

Status: None.

Conformance: BCC1, ECC1

StartupHook

Syntax: `void StartupHook( void );`

Input: None.

Output: None.

Description: To implement *StartupHook* the routine with name 'StartupHook' shall be defined in user's code. This hook is called by the operating system at the end of the operating system initialization and before the initialization of Interrupt Sources, System and Second timers

and before scheduler starts running. At this point in time the application can initialize hardware, etc.

Particularities: All interrupts are disabled in *StartupHook*.

This hook is not called if the system configuration option *STARTUPHOOK* is set to *FALSE*.

Status: None.

Conformance: BCC1, ECC1

ShutdownHook

Syntax:
```
void ShutdownHook( StatusType <Error> );
```

Input: <*Error*> – a code of the error occurred.

Output: None.

Description: To implement *ShutdownHook* the routine with name 'ShutdownHook' shall be defined in user's code. This hook is called by the operating system when the *ShutdownOS* service has been called. This routine is called during the operation system shut down. User can avoid return from the hook to calling level. For example reset signal can be generated in the hook.

Particularities: All interrupts are disabled in *ShutdownHook*.

This hook is not called if the system configuration option *ShutdownHook* is turned off in the configuration file.

Status: None.

Conformance: BCC1, ECC1

This hook is not called if the system configuration option *SHUTDOWNHOOK* is set to *FALSE*.

Conformance: BCC1, ECC1

17

Debugging Application

This chapter provides information about Freescale AUTOSAR OS feature intended for debugging a user application with OS.

This chapter consists of the following sections:

General

AUTOSAR OS contains several mechanisms which help user to debug application.

Extended Status

Extended status allows to check most of errors caused by improper use of AUTOSAR services.

ORTI

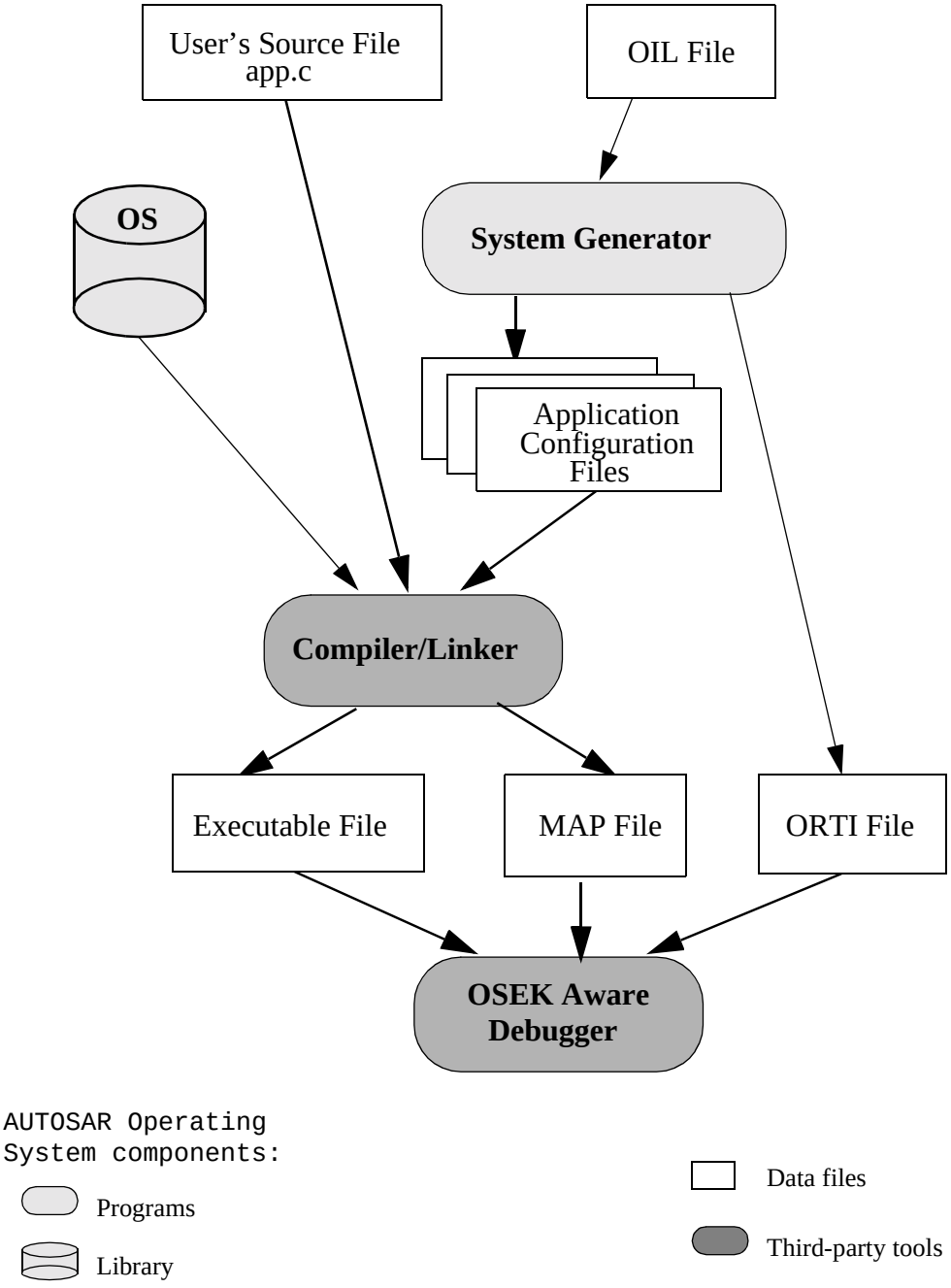
The purpose of OSEK Run Time Interface (ORTI) implementation is giving the user extended opportunities in debugging embedded OSEK and AUTOSAR applications. The OSEK Run Time Interface implementation confirms *OSEK/VDX OSEK Run Time Interface (ORTI), Part A: Language Specification, v. 2.1.1, 4 March 2002* and *OSEK/VDX OSEK Run Time Interface (ORTI), Part B: OSEK Objects and Attributes, v. 2.1, 17 April 2002*. The current version of Freescale AUTOSAR OS supports ORTI version 2.1

The ORTI shall be supported from both sides: an AUTOSAR OS and a debugger. The debugger able to display information in terms of OSEK system objects is “OSEK aware” debugger. The internal OS data is to be made available to the debugger. For this purpose

special ORTI file is generated at configuration time by a System Generator. As a result, more information will be available to the user during application debugging session.

System Generator (SysGen) uses OIL file (App.oil) as an input file. Option **-o** of the SysGen defines output ORTI file name. The SysGen utility generates static information in the ORTI format. This utility analyzes the application configuration and generates ORTI file. The same OIL file is used for configuring OS. After application is compiled and linked and executable and map files are created then they are loaded by the debugger. If the debugger is OSEK aware then it can load also the ORTI file for this application. The information from ORTI file provides the debugger with possibility to display information about system objects of current implementation of the OS. This process is depicted on Figure 17.1.

Figure 17.1 Application Building Process with ORTI Support



Using OS Extended Status for Debugging

It is recommended to use Operating System *Extended Status* when developing an application to analyze return codes of system services. Such OS configuration is more memory and time consuming but it allows the user to save time for errors eliminating. Error codes returned by the AUTOSAR OS services covers most of possible errors that can arise during development. Therefore it is useful to check these codes after a service call to avoid error that can lead to the system crash. For example, a task can perform the *TerminateTask* service while it is still occupying a resource. This service will not be performed and the task will remain active (*running*). In case of Extended Status the *E_OS_RESOURCE* error code is returned and it is possible to detect this situation. But in the system without Extended Status there is no additional check and this error is not indicated and the application behavior will be unpredictable!

After all errors in the application are eliminated the *Extended Status* may be turned off to remove additional status checks from the application and get the reliable application of the smaller size.

Context Switch Monitoring

"ORTI Trace Interfaces" provides the user with the ability to trace application execution, including context switching. But there is an ability to monitor context switching without using ORTI. Breakpoints, traces and time stamps can be integrated individually into application software with the help of context switch hook routines *PreTaskHook* and *PostTaskHook*.

Additionally, the user can set time stamps enabling him to trace the program execution, for example, at the following locations before calling operating system services:

- When activating or terminating tasks;
- When setting or clearing events in the case of Extended Tasks;
- At explicit points of the schedule (*ChainTask*, *Schedule*);
- At the beginning or the end of ISR;
- When occupying and releasing resources or at critical locations.

ORTI Features

ORTI provides two kinds of interfaces to the OS data:

- Trace interface, which means getting an access to the data on a running target when it is essential to trace data changes in a real time;
- Breakpoint interface, which provides an access to desirable data on a stopped target.

Note that ORTI Trace interface is designed to provide access to requested data in accomplished form that is not requiring an additional processing.

ORTI Trace Interfaces

There are trace ORTI interfaces in run-time: running task identification, ISR identification, AUTOSAR OS system calls identification. This interface is turned on if DEBUG_LEVEL = 2. If DEBUG_LEVEL = 3 then only running Task and ISR trace is supported.

The special attributes are generated for the OS object in ORTI file to support trace interfaces, the names of these attributes corresponds to ORTI specification supported. The following trace interfaces are available to the user:

- running task

  This attribute specifies the name of the currently running task within the OS.

  The value of this attribute presents a one byte memory block which contains either byte numeric identifier of a task which is currently in the *running* state or the special byte value in case of none tasks are in the *running* state. The certain values of task identifiers are statically determined and enumerated in the type field of running task attribute of an OS object as well as the special value (*NO_TASK*) representing none of tasks running.

  The value of the running task attribute is updated each time CPU switches to another task or to the code corresponding to none of task running.

- running ISR

This attribute specifies the identifier of currently executed ISR of category 2. It occupies a one byte memory block.

There are the following special values of the attribute:

- NO_ISR - OS works on task or dispatcher level.
- SYSTEM_TIMER - System Timer ISR is activate.
- SECOND_TIMER - Second Timer ISR is activate.

Other values correspond to ISRs of category 2 defined in the OIL file.

The next scheme is used for ISR tracing: running ISR attribute has TASK_LEVEL value when OS works at task level. When one of ISRs category 2 starts execution, the value of running ISR is changed to ISR identifier. When SystemTimer ISR gets CPU, the value of running ISR is changed to SYSTEM_TIMER. When ISR terminates and OS returns to task level, running ISR gets value TASK_LEVEL.

- current service

  This attribute specifies which operating service, if any, is currently being executed.

  The current service attribute value presents a one byte memory block which represents a service numeric identifier. This value is updated with AUTOSAR OS service (*Service*) identifier with least signicant bit set when CPU enters *Service* code.

  In case of leaving AUTOSAR OS service the attribute value is set to service indentifier with least signicant bit cleared thus indicating end of service. The attribute is updated with such value on the following events occurred:

  - CPU leaves AUTOSAR OS service code and starts to execute non AUTOSAR OS service code - i.e. the application code in the same or another task;
  - CPU leaves AUTOSAR OS service code and starts to execute idle loop.

Typically trace sequence of current service looks like the following:

Table 17.1 ORTI Trace Sequence

Traced Value	Description
...	
SERVICE_1_ID	entering Service 1
SERVICE_1_EXIT	leaving Service 1 (possibly due to task switching)
...	
SERVICE_2_ID	entering Service 2
SERVICE_3_ID	entering nested Service 3 call (e.g. hook routine)
SERVICE_3_EXIT	leaving Service 3 and resuming Service 2
SERVICE_2_EXIT	leaving Service 2 (possibly due to task switching)
...	

ORTI Breakpoint Interface

There is the ORTI Breakpoint interface intended to facilitate debugger access to task related data. The interface is turned on, if DEBUG_LEVEL OIL attribute does not equal to 0.

The following static (at breakpoints) ORTI services are supported for a debugger on breakpoints: access to tasks, stacks, counters, alarms, resources and messages information.

Information needed to display the current status of objects is available for the debugger whenever the debugger is stopped (i.e. this information is not required in real-time and hence can be read from the TCB or similar structure).

Information in the ORTI file allows a debugger to display task information using values that the user sets in the OIL file.

The following information about OS is available to the user:

- current application mode

  This attribute specifies the current application mode.

  The current application mode value presents a one byte memory block which represents a mode identifier.
- running task priority

This attribute specifies the current priority of the task reffered by the running task attribute.

The value of this attribute is provided with taking into account possible task priority changes due to a dynamic resource allocation.

- last error

  This attribute specifies the code of the last error detected. The last error attribute value presents a one byte memory block which represents an error code identifier. This value is updated with error identifier when error occurs. At start up the error code is initialized with E_OK. It is never set back to E_OK after first error

The following task information is available to the user:

- property

  The task property indicates static properties of the task.

- priority

  The task priority value is provided with taking into account possible task priority changes due to a dynamic resource allocation.

- task state

  The task state indicates a standard AUTOSAR state the task is currently in.

- task stack

  The task stack shows which stack area is used by the task.

- event states

  The event states can be used to determine events which are currently set or cleared.

- wait events

  Wait events value contains bit set to one for event which task is waiting for.

- event masks

  The event masks define correspondence between event name and bit mask.

The following stack information is available to the user:

- size

The size specifies the size (in bytes) of a memory area allocated for stack.

- base address

  The base address specifies the lowest address of stack memory area.

- stack direction

  The stack direction specifies the direction of stack growth and may have “UP” or “DOWN” as its value. “UP” means growing from lower to higher addresses. “DOWN” means growing from higher addressed to lower addresses.

- fill pattern

  The fill pattern attribute specifies with which pattern the stack area is initialized. This allows the debugger to evaluate the maximum stack usage.

The following counter information is available to the user:

- maxallowed value
- ticksperbase
- mincycle
- current counter value
- indicator if alarms attached to counter is activated
- property, which indicates OSTICKDURATION, the type of the counters (SWCOUNTER or HWCOUNTER) and timer hardware for system timers or indicates if it is an application counter

The following alarm information is available to the user:

- alarm status
- assigned counter
- action on alarm expiration
- time left to expire
- cycle value for cyclic alarm

The following resource information is available to the user:

- priority of the resource
- resource state

Depending on ORTI specification supported either message or message container information is generated. The following message information is available to the user:

- message type
- task to be notified and event
- callback function name
- notification flag

The message container describes each existing combination of messages and ACCESSORRECEIVED. The following message container information is available to the user:

- message name
- message type
- size of the queue
- number of valid messages in th equeue
- first valid message
- message state
- action to be performed when message arrives
- message receiver
- message sender

Stack Debugging Support

Stack Overflow Checking

Stack overflow check can be used in all classes; in SC3 and SC4 the stack errors might be catched also by Memory Protection subsystem.

Optional stack overflow checking can be used during run time to check tasks, ISR and main stacks usage. Main and task stacks are checked during task switch. If stack overflow is detected the

ProtectionHook (only in SC3,4) or *ShutdownOS* with E_OS_STACKFAULT status is called. OS stack overflow control is turned on if *STACKMONITORING* attribute value is TRUE.

System checks task's and main stacks for overflow when tasks state is changed from *RUNNING* and before entering ISR of category 2. ISR stack (or main stack in case of BCC1, SC1) is checked before leaving ISR of category 2.

A

Sample Application

This appendix contains the text and listing of the sample customer application developed using Freescale AUTOSAR OS.

This appendix consists of the following sections:

Description

The Sample applications delivered with the Freescale AUTOSAR OS should help to learn how to use AUTOSAR OS. The Sample's source files are located in the `sample\standard` directory – it contains all files needed to create an executable files.

The samples might be configured with help of ElectroBit Tresos Studio 2009a.sr1. The 'os_ts' directory contains example of Tresos configuration project for OS module based on OS samples.

The sample applications are configured to run on the EVB from internal RAM. The sample for SC1 is configured to run from internal flash with or without debugger. Please see the sample readme file for details.

The are 4 samples: one for each of SC1..4 classes with similar algorithms.There are scripts for Lauterbach debbuger (sample\standard\56xxAM.cmm) that allows to load the sample onto XPC56xx EVB board.

The Sample is not a real application and it does not perform any useful work. But it has a certain algorithm so it is possible to track the execution. It uses most of AUTOSAR OS mechanisms and allows the user to have the first look inside the AUTOSAR OS.

Each Sample consists of six tasks. It uses three counters (HW and SW are on the System Timer and Second Timer and one "user

counter"), two alarms, ScheduleTable, two resources and two messages.

Generally, Sample tasks are divided into two parts. *TASKSND1, TASKSND2* and *TASKCNT* compose the first part (`samplets.c` file) and *TASKRCV1, TASKRCV2* and *TASKSTOP* are the second part (`samplerv.c` file). These two parts interact with the help of the messages and alarm mechanism.

The Extended task *TASKRCV1* is activated by OS (autostarted). It performs the following initializations:

- init *TSCOUNTER* counter with value 0,
- set absolute *STOPALARM* alarm to value 20 (when *STOPALARM* expired it activates *STOPTASK* task),
- starts *ScheduleTable*,
- clears *MSGAEVENT* and *TIMLIMITEVENT* events,
- set relative *TIMLIMITALARM* alarm to value 20 (when *TIMLIMITALARM* expired it sets *TIMLIMITEVENT* event for this (*TASKRCV1*) task).

Then it enters *waiting* state - waiting *MSGAEVENT* and / or *TIMLIMITEVENT* events. When event occurred, *TASKRCV1* checks which event occurs.

If *MSGAEVENT* event occurred (*MsaA* message arrived) then *TASKRCV1* task:

- cancels *TIMLIMITEVENT* alarm,
- gets *MSGAACCESS* resource to prevent access to *MsgA* message,
- receives *MsgAReceived* message,
- releases *MSGAACCESS* resource,
- clears event and goes to waiting state again.

If *TIMLIMITEVENT* event occurred (time limit was exceeded) then *TASKRCV1* task goes to the very beginning and repeats initialization, restarting the application (but OS is not restarted, it continue running).

ScheduleTable has three steps (action points):

1. On the first step *TASKSND1* task is activated. It does the following:

- gets *MSGAACCESS* resource to prevent access to *MsgA* message,
- increments *MsgA* message value,
- send *MsgA* message
  (*MsgA* message sets *MSGAEVENT* event for *TASKRCV1* task),
- releases *MSGAACCESS* resource,
- terminates itself.

2. On the second step *TASKSND2* task is activated.
   Task *TASKSND2*:

- adds 30 to “ind” variable value,
- if “ind” variable value greater or equal 50 then subtracts 51 from the “ind” value,
- stores *SYSTEMTIMER* value to body of *MsgB* message copy,
- gets *MSGBACCESS* resource to prevent access to *MsgB* message,
- send *MsgB* message
  (message *MsgB* activates *TASKRCV2* task),
- releases *MSGBACCESS* resource,
- terminates itself.

3. On third step *TASKCNT* task is activated.
   TASKCNT task:

- increments *TSCOUNTER* counter
  (when *TSCOUNTER* counter value reaches 20 the alarm ‘*STOPALARM’* expires),
- terminates itself.

TASKRCV2 task:

- gets *MSGBACCESS* resource to prevent access to *MsgB* message,
- receives *MsgBReceived* message without copy,
- releases *MSGBACCESS* resource,
- terminates itself.

TASKSTOP task:

- stops ScheduleTable (after a while *TIMLIMITALARM* expires)

- terminates itself.

The user can watch “ind” variable, messages content and so on.

Sample with Timing Protection

In the Sample for SC2 the timing protection is set for the following tasks:

- task *TASKSND1* has execution budget and locking time for resource *MSGAACCESS* defined;
- task *TASKSND2* has task arrival rate defined;
- task *TASKRCV1* has execution budget and locking time set for resource *MSGAACCESS*.

The variable ‘repeat’ is added for SC2, it is incremented in external cycle of TaskRcv1. When the value of ‘repeat’ becames 5 the TaskRcv1 task forces Timing Protection vioaltion - tries to perform endless cycle. After expiration of the TaskRcv1 task execution budget the OS automatically performs shutdown.

Sample with Memory Protection

In the Sample for SC3 the memory violation is forced in the *TASKRCV1* task. The OS calls ProtectionHook which returns PRO_TERMINATEAPPL_RESTART thus restarting the RCV_APPL Application.

Sample with Timing and Memory Protection

In the Sample for SC4 the timing protection is set for the *TASKSND2*: When *TASKSND2* variable reaches value 22 the timing violation is forced and the Task is killed, but it does not disturbs the Sample algorithm as *TASKSND2* in any case terminates. After *TASKSND2* reaches 72 the memory violation is forced and the OS goes to shutdown because ProtectionHook returns PRO_SHUTDOWN.

Source Files

Source files for the Sample application are the following:

- `samplets(n).c` – the application code (*TASKSND1*, *TASKSND2* and *TASKCNT*)
- `samplerv(n).c` – the application code (*TASKRCV1*, *TASKRCV2* and *TASKSTOP*)
- `sample.h` – header file for the application code
- `memmap.h` – header file for memory sections control
- `sample(n).oil` – OSEK Implementation Language file
- sample(n).epc - sample configuraqtion in AUTOSAR XML
- `makefile` – make file for compiling sample using GNU make utility
- `sample.`(ld, dld) - linker command files

The directory structure of the Sample application is described in the readme.txt file located in the `sample\standard` directory.

To the description of sample build please refer to the sample readme file.

B

System Service Timing

This appendix provides information about OS services execution time. All measurements were done on MPC5644A MCU.

This appendix consists of the following sections:

General Notes

Results in tables below were got on the basis of the certain OS configuration. The list of system properties is shown below, and this configuration is called in the table as “Initial”. Properties that are not listed have their default values. In the column “Configuration” the differences from the given list (“Initial“) are indicated. For each configuration the corresponded numbers are provided in the table. In the column “Conditions” the specifics details for service execution are indicated.

```
CC = BCC1;
STATUS = STANDARD (for SC1,2) EXTENDED (for SC3,4);
STARTUPHOOK = FALSE;
SHUTDOWNHOOK = FALSE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
ERRORHOOK = FALSE;
USEGETSERVICEID = FALSE;
USEPARAMETERACCESS = FALSE;
FastTerminate = TRUE;
USERESSCHEDULER = TRUE;
ACTIVATION = 1; (for all TASK objects)
SCHEDULE = FULL; (for all TASK objects)
ISR category 2 defined
```

Four tasks are present in the application

In SC2, SC4 Execution Budget and Arrival Rate are defined for each Task / ISR

In SC3, SC4 classes 3 OS-Applications are defined. For column marked “Trust” 2 trusted and one none-trusted OS-Applications are defined, the services are called from trusted OS-Application. For column “NonTr” 1 trusted and 2 non-trusted OS-Applications are defined, the services are called from non-trusted OS-Application.

Data Sheets for CodeWarrior Compiler

The table below contains OS services timings in CPU cycles for Freescale AUTOSAR OS / MPC56xxAM compiled with CodeWarrior compiler.

The data for all Scalability classes are presented, for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown.

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		42179	43579	42830	42931	44020	44114
ActivateTask							
Initial	Task activated, no rescheduling	224	234	385	630	385	630
	Task activated and rescheduled	462	870	921	1136	1221	1425
Internal resource defined	Task activated and rescheduled	588	970	1021	1226	1331	1572
SCHEDULE = NON for all tasks	Task activated, no rescheduling	195	175	325	570	325	570

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
TerminateTask							
Initial	Task terminated, return to lower prio task	275	809	731	1020	971	1269
Initial	Task terminated, new task started	221	655	680	881	910	1101
Internal resource defined	Task terminated, return to lower prio task	453	939	851	1170	1131	1416
Internal resource defined	Task terminated, new task started	369	805	810	1011	1060	1261
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	195	185	665	882	665	882
SCHEDULE = NON for all tasks	Task terminated, new task started	210	635	681	882	911	1102
ChainTask							
Initial	Task terminated, next task started	182	736	871	1062	1101	1282
Internal resource defined	Task terminated, next task started	340	886	1001	1212	1251	1462
SCHEDULE = NON for all tasks	Task terminated, next task started	201	754	881	1082	1111	1302
Schedule							
Initial	No rescheduling, all tasks are preemptable	315	165	235	480	245	490
Internal resource defined	No rescheduling, all tasks are preemptable	267	295	375	610	375	610
SCHEDULE = NON for all tasks	No rescheduling	325	165	235	480	245	490
	Rescheduling, other task becomes running	281	746	681	892	1021	1222
GetTaskId							
Initial		69	69	459	666	459	666

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetTaskState							
Initial	For running task	125	125	614	841	614	841
	For ready task	205	185	655	882	655	882
	For suspended task	195	185	665	882	665	882
EnableAllInterrupts							
Initial		14	14	132	377	132	377
DisableAllInterrupts							
Initial		29	29	149	394	149	394
ResumeAllInterrupts							
Initial		92	92	142	397	142	397
SuspendAllInterrupts							
Initial		109	109	139	394	139	394
ResumeOSInterrupts							
Initial		112	132	172	427	182	437
SuspendOSInterrupts							
Initial		132	142	172	427	182	437
GetResource							
a resource defined	Task occupies resource	315	725	555	800	975	1220
call from ISR level	ISR occupies resource	302	734	612	857	1085	1280
ReleaseResource							
a resource defined	Task release resource, no rescheduling	341	545	601	836	844	1049
	Task release resource, other task becomes running	562	1236	1168	1339	1791	1906
call from ISR level	ISR release resource	362	574	642	887	885	1120

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	175	184	355	620	355	620
	Event set, task becomes ready	225	235	414	669	414	669
CC = ECC1 an event defined	Event set, but task was not waiting it	264	255	465	690	465	690
	Event set, task becomes ready	315	305	515	750	515	750
	Event set, task becomes running	702	1164	1118	1393	1531	1836
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		115	105	214	459	214	459
CC = ECC1 an event defined		105	115	195	450	195	450
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		59	68	618	845	618	835
CC = ECC1 an event defined		68	59	619	836	619	846
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	134	325	235	490	435	670
	Task becomes waiting	572	923	841	1133	1138	1446
CC = ECC1 an event defined	Event was already set, task remains running	135	324	245	490	425	690
	Task becomes waiting	592	935	868	1163	1181	1506

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SendMessage							
message defined	Message size 16 bytes	764	604	1255	1482	1255	1482
ReceiveMessage							
message defined	Message size 16 bytes	745	755	1205	1272	1205	1272
InitCounter							
a counter defined		69	69	265	510	265	510
IncrementCounter							
a counter defined		191	191	331	576	331	576
1 alarm is set to a counter	Alarm is not expired	421	481	661	916	661	916
	Notified task becomes ready	501	571	750	995	750	995
	Notified task becomes running	737	1212	1287	1478	1597	1798
10 tasks, 10 alarms are set to a counter	Alarm is not expired	581	751	920	1165	920	1165
	Notified tasks become ready	1251	1501	1761	1916	1761	1916
	High priority notified task becomes running	1517	2201	2400	2481	2700	2791
GetCounterValue							
a counter defined		69	69	548	775	548	775
GetCounterInfo							
a counter defined		91	91	623	840	623	840
GetAlarmBase							
an alarm defined		113	113	589	806	589	806

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SetRelAlarm							
an alarm defined	Alarm set	441	461	707	942	707	942
	Alarm expires immediately, notified task becomes ready	585	645	921	1136	921	1136
	Alarm expires immediately, notified task becomes running	937	1392	1639	2105	1949	2425
10 alarms are defined, 9 alarms are set on a counter	Alarm set	351	390	676	931	676	931
	Alarm expires immediately, notified task becomes ready	761	931	1183	1458	1183	1458
	Alarm expires immediately, notified task becomes running	997	1551	1742	1963	2042	2273
SetAbsAlarm							
an alarm defined	Alarm set	335	355	627	862	627	862
	Alarm expires immediately, notified task becomes ready	601	660	922	1187	922	1187
	Alarm expires immediately, notified task becomes running	827	1282	1459	1660	1769	1980
10 alarms are defined, 9 alarms are set on a counter	Alarm set	284	295	607	862	607	862
	Alarm expires immediately, notified task becomes ready	751	931	1193	1458	1193	1458
	Alarm expires immediately, notified task becomes running	987	1550	1732	1963	2032	2273
CancelAlarm							
1 alarm is set on a counter		224	255	405	650	405	650
10 tasks, 10 alarms are set on a counter		234	255	375	640	375	640

Table B.1 Run-time Services Timing in CPU cycles (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetAlarm							
1 alarm is set on a counter		195	195	685	902	685	902
10 tasks, 10 alarms are set on a counter		195	194	705	912	705	912
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	1479	1508	1715	1960	1715	1960
	Task activated, and rescheduled	1724	2132	2251	2436	2561	2736
StopScheduleTable							
ScheduleTable defined		518	508	658	903	658	903
GetApplicationID							
Initial		N/A	N/A	0	374	0	374
CheckObjectAccess							
Initial		N/A	N/A	129	504	129	504
CheckObjectOwnerShip							
Initial		N/A	N/A	69	424	69	424
CheckTaskMemoryAccess							
Initial		N/A	N/A	259	626	259	626
CheckISRMemoryAccess							
Initial		N/A	N/A	179	666	179	666
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	387	738	387	738
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	427	2600	427	2982

The table below contains OS services timings in microseconds for Freescale AUTOSAR OS / MPC56xxAM compiled with CodeWarrior. The data is calculated from Table B.5 for 64 MHz CPU clock frequency.

The data for all Scalability classes are presented, for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown.

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		659.0	680.9	669.2	670.8	687.8	689.3
ActivateTask							
Initial	Task activated, no rescheduling	3.50	3.65	6.01	9.84	6.01	9.84
	Task activated and rescheduled	7.21	13.59	14.39	17.75	19.08	22.27
Internal resource defined	Task activated and rescheduled	9.18	15.16	15.95	19.16	20.80	24.56
SCHEDULE = NON for all tasks	Task activated, no rescheduling	3.04	2.73	5.07	8.90	5.07	8.90
TerminateTask							
Initial	Task terminated, return to lower prio task	4.29	12.64	11.42	15.94	15.17	19.83
Initial	Task terminated, new task started	3.45	10.23	10.63	13.77	14.22	17.20
Internal resource defined	Task terminated, return to lower prio task	7.07	14.67	13.30	18.28	17.67	22.13
Internal resource defined	Task terminated, new task started	5.76	12.58	12.66	15.80	16.56	19.70
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	3.04	2.89	10.39	13.78	10.39	13.78
SCHEDULE = NON for all tasks	Task terminated, new task started	3.28	9.92	10.64	13.78	14.23	17.22
ChainTask							
Initial	Task terminated, next task started	2.84	11.50	13.61	16.59	17.20	20.03
Internal resource defined	Task terminated, next task started	5.31	13.84	15.64	18.94	19.55	22.84

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SCHEDULE = NON for all tasks	Task terminated, next task started	3.14	11.78	13.77	16.91	17.36	20.34
Schedule							
Initial	No rescheduling, all tasks are preemptable	4.92	2.57	3.67	7.50	3.82	7.65
Internal resource defined	No rescheduling, all tasks are preemptable	4.17	4.60	5.85	9.53	5.85	9.53
SCHEDULE = NON for all tasks	No rescheduling	5.07	2.57	3.67	7.50	3.82	7.65
	Rescheduling, other task becomes running	4.39	11.66	10.64	13.94	15.95	19.09
GetTaskId							
Initial		1.07	1.07	7.17	10.41	7.17	10.41
GetTaskState							
Initial	For running task	1.95	1.95	9.59	13.14	9.59	13.14
	For ready task	3.20	2.89	10.23	13.78	10.23	13.78
	For suspended task	3.04	2.89	10.39	13.78	10.39	13.78
EnableAllInterrupts							
Initial		0.21	0.21	2.06	5.89	2.06	5.89
DisableAllInterrupts							
Initial		0.45	0.45	2.32	6.15	2.32	6.15
ResumeAllInterrupts							
Initial		1.43	1.43	2.21	6.20	2.21	6.20
SuspendAllInterrupts							
Initial		1.70	1.70	2.17	6.15	2.17	6.15
ResumeOSInterrupts							
Initial		1.75	2.06	2.68	6.67	2.84	6.82
SuspendOSInterrupts							
Initial		2.06	2.21	2.68	6.67	2.84	6.82
GetResource							
a resource defined	Task occupies resource	4.92	11.33	8.67	12.50	15.23	19.06

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
call from ISR level	ISR occupies resource	4.71	11.47	9.56	13.39	16.95	20.00
ReleaseResource							
a resource defined	Task release resource, no rescheduling	5.32	8.51	9.39	13.06	13.19	16.39
	Task release resource, other task becomes running	8.78	19.31	18.25	20.92	27.98	29.78
call from ISR level	ISR release resource	5.65	8.96	10.03	13.86	13.83	17.50
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	2.73	2.87	5.54	9.68	5.54	9.68
	Event set, task becomes ready	3.51	3.67	6.46	10.45	6.46	10.45
CC = ECC1 an event defined	Event set, but task was not waiting it	4.12	3.98	7.26	10.78	7.26	10.78
	Event set, task becomes ready	4.92	4.76	8.04	11.72	8.04	11.72
	Event set, task becomes running	10.97	18.19	17.47	21.77	23.92	28.69
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		1.79	1.64	3.34	7.17	3.34	7.17
CC = ECC1 an event defined		1.64	1.79	3.04	7.03	3.04	7.03
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		0.92	1.06	9.65	13.20	9.65	13.05
CC = ECC1 an event defined		1.06	0.92	9.67	13.06	9.67	13.22

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	2.09	5.07	3.67	7.65	6.79	10.47
	Task becomes waiting	8.93	14.42	13.14	17.70	17.78	22.59
CC = ECC1 an event defined	Event was already set, task remains running	2.10	5.06	3.82	7.65	6.64	10.78
	Task becomes waiting	9.25	14.61	13.56	18.17	18.45	23.53
SendMessage							
message defined	Message size 16 bytes	11.94	9.43	19.61	23.16	19.61	23.16
ReceiveMessage							
message defined	Message size 16 bytes	11.64	11.80	18.83	19.88	18.83	19.88
InitCounter							
a counter defined		1.07	1.07	4.14	7.96	4.14	7.96
IncrementCounter							
a counter defined		2.98	2.98	5.17	9.00	5.17	9.00
1 alarm is set to a counter	Alarm is not expired	6.57	7.51	10.33	14.31	10.33	14.31
	Notified task becomes ready	7.82	8.92	11.72	15.55	11.72	15.55
	Notified task becomes running	11.52	18.94	20.11	23.09	24.95	28.09
10 tasks, 10 alarms are set to a counter	Alarm is not expired	9.07	11.73	14.38	18.20	14.38	18.20
	Notified tasks become ready	19.55	23.45	27.52	29.94	27.52	29.94
	High priority notified task becomes running	23.70	34.39	37.50	38.77	42.19	43.61
GetCounterValue							
a counter defined		1.07	1.07	8.56	12.11	8.56	12.11

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetCounterInfo							
a counter defined		1.42	1.42	9.73	13.13	9.73	13.13
GetAlarmBase							
an alarm defined		1.76	1.76	9.20	12.59	9.20	12.59
SetRelAlarm							
an alarm defined	Alarm set	6.89	7.20	11.05	14.72	11.05	14.72
	Alarm expires immediately, notified task becomes ready	9.14	10.08	14.39	17.75	14.39	17.75
	Alarm expires immediately, notified task becomes running	14.64	21.75	25.61	32.89	30.45	37.89
10 alarms are defined, 9 alarms are set on a counter	Alarm set	5.48	6.09	10.56	14.55	10.56	14.55
	Alarm expires immediately, notified task becomes ready	11.89	14.55	18.48	22.78	18.48	22.78
	Alarm expires immediately, notified task becomes running	15.58	24.23	27.22	30.67	31.91	35.52
SetAbsAlarm							
an alarm defined	Alarm set	5.23	5.54	9.79	13.47	9.79	13.47
	Alarm expires immediately, notified task becomes ready	9.39	10.31	14.41	18.55	14.41	18.55
	Alarm expires immediately, notified task becomes running	12.92	20.03	22.80	25.94	27.64	30.94
10 alarms are defined, 9 alarms are set on a counter	Alarm set	4.43	4.60	9.48	13.47	9.48	13.47
	Alarm expires immediately, notified task becomes ready	11.73	14.55	18.64	22.78	18.64	22.78
	Alarm expires immediately, notified task becomes running	15.42	24.22	27.06	30.67	31.75	35.52

Table B.2 Run-time Services Timing in microseconds (CW)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
CancelAlarm							
1 alarm is set on a counter		3.50	3.98	6.32	10.16	6.32	10.16
10 tasks, 10 alarms are set on a counter		3.65	3.98	5.85	10.00	5.85	10.00
GetAlarm							
1 alarm is set on a counter		3.04	3.04	10.70	14.09	10.70	14.09
10 tasks, 10 alarms are set on a counter		3.04	3.03	11.02	14.25	11.02	14.25
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	23.11	23.56	26.80	30.63	26.80	30.63
	Task activated, and rescheduled	26.94	33.31	35.17	38.06	40.02	42.75
StopScheduleTable							
ScheduleTable defined		8.09	7.93	10.28	14.11	10.28	14.11
GetApplicationID							
Initial		N/A	N/A	0	5.84	0	5.84
CheckObjectAccess							
Initial		N/A	N/A	2.01	7.87	2.01	7.87
CheckObjectOwnerShip							
Initial		N/A	N/A	1.07	6.62	1.07	6.62
CheckTaskMemoryAccess							
Initial		N/A	N/A	4.04	9.78	4.04	9.78
CheckISRMemoryAccess							
Initial		N/A	N/A	2.79	10.41	2.79	10.41
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	6.04	11.53	6.04	11.53
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	6.67	40.63	6.67	46.59

The table below contains OS services latency for Freescale AUTOSAR OS / MPC56xxAM compiled with CodeWarrior. For classes SC3 and SC4 latency was measured only for trusted Application. The data in "µs" columns is calculated for 64 MHz CPU clock.

OS Latency is the time for which the interrupts of category 2 are disabled in the service. Interrupts of category 1 are not disabled inside OS services that caused rescheduling only for SC3 and SC4, except StartOS, which disables all interrupts for the time given in the table. The ISRs of category 1 are disabled for the time needed to re-configure MPU for given OS-Application, it is about 60 cycles in case of switching to non-trusted OS-Application.

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
StartOS									
Initial		41791	653.0	42793	668.6	44403	693.8	43019	672.2
ActivateTask									
Initial	Task activated, no rescheduling	154	2.40	174	2.71	164	2.56	164	2.56
	Task activated and rescheduled	372	5.81	795	12.42	682	10.66	991	15.48
Internal resource defined	Task activated and rescheduled	488	7.62	892	13.94	778	12.16	1088	17.00
SCHEDULE = NON for all tasks	Task activated, no rescheduling	124	1.93	114	1.78	114	1.78	114	1.78
TerminateTask									
Initial	Task terminated, return to lower prio task	204	3.18	754	11.78	584	9.12	843	13.17
Initial	Task terminated, new task started	92	1.43	565	8.82	518	8.09	738	11.53
Internal resource defined	Task terminated, return to lower prio task	362	5.65	884	13.81	750	11.72	980	15.31
Internal resource defined	Task terminated, new task started	220	3.43	732	11.44	658	10.28	878	13.72

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	134	2.09	124	1.93	124	1.93	124	1.93
SCHEDULE = NON for all tasks	Task terminated, new task started	62	0.96	536	8.37	518	8.09	738	11.53
ChainTask									
Initial	Task terminated, next task started	102	1.59	645	10.08	608	9.50	828	12.94
Internal resource defined	Task terminated, next task started	220	3.43	812	12.69	748	11.69	968	15.13
SCHEDULE = NON for all tasks	Task terminated, next task started	82	1.28	645	10.08	608	9.50	828	12.94
Schedule									
Initial	No rescheduling, all tasks are preemptable	62	0.96	82	1.28	82	1.28	82	1.28
Internal resource defined	No rescheduling, all tasks are preemptable	184	2.87	202	3.15	212	3.31	202	3.15
SCHEDULE = NON for all tasks	No rescheduling	52	0.81	82	1.28	82	1.28	82	1.28
	Rescheduling, other task becomes running	112	1.75	689	10.77	522	8.15	851	13.30
GetTaskId									
Initial		0	0.00	0	0.00	0	0.00	0	0.00
GetTaskState									
Initial	For running task	74	1.15	84	1.31	84	1.31	84	1.31
	For ready task	144	2.25	124	1.93	124	1.93	124	1.93
	For suspended task	134	2.09	124	1.93	124	1.93	124	1.93
GetResource									
a resource defined	Task occupies resource	124	1.93	524	8.18	134	2.09	524	8.18
call from ISR level	ISR occupies resource	131	2.04	574	8.96	131	2.04	574	8.96

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
ReleaseResource									
a resource defined	Task release resource, no rescheduling	181	2.82	433	6.76	200	3.12	424	6.62
	Task release resource, other task becomes running	429	6.70	1141	17.83	714	11.16	1348	21.06
call from ISR level	ISR release resource	181	2.82	433	6.76	200	3.12	424	6.62
SetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	74	1.15	64	1.00	74	1.15	74	1.15
	Event set, task becomes ready	134	2.09	124	1.93	134	2.09	134	2.09
CC = ECC1 an event defined	Event set, but task was not waiting it	134	2.09	134	2.09	144	2.25	144	2.25
	Event set, task becomes ready	194	3.03	194	3.03	204	3.18	204	3.18
	Event set, task becomes running	589	9.20	1040	16.25	775	12.11	1218	19.03
ClearEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		54	0.84	54	0.84	54	0.84	54	0.84
CC = ECC1 an event defined		54	0.84	54	0.84	54	0.84	54	0.84
GetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		0	0.00	0	0.00	0	0.00	0	0.00
CC = ECC1 an event defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
WaitEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	54	0.84	54	0.84	54	0.84	54	0.84
	Task becomes waiting	519	8.10	852	13.31	645	10.08	968	15.13
CC = ECC1 an event defined	Event was already set, task remains running	52	0.81	242	3.78	52	0.81	232	3.62
	Task becomes waiting	531	8.29	892	13.94	687	10.73	1030	16.09
SendMessage									
message defined	Message size 16 bytes	664	10.38	504	7.87	664	10.38	664	10.38
ReceiveMessage									
message defined	Message size 16 bytes	644	10.06	644	10.06	644	10.06	644	10.06
InitCounter									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
IncrementCounter									
a counter defined		130	2.03	120	1.87	120	1.87	120	1.87
1 alarm is set to a counter	Alarm is not expired	330	5.15	400	6.25	410	6.40	410	6.40
	Notified task becomes ready	410	6.40	490	7.65	500	7.81	500	7.81
	Notified task becomes running	618	9.65	1131	17.67	1024	16.00	1324	20.69
10 tasks, 10 alarms are set to a counter	Alarm is not expired	520	8.12	670	10.47	670	10.47	670	10.47
	Notified tasks become ready	1200	18.75	1420	22.19	1430	22.34	1430	22.34
	High priority notified task becomes running	1458	22.78	2118	33.09	2004	31.31	2314	36.16
GetCounterValue									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
GetCounterInfo									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
GetAlarmBase									
an alarm defined		0	0.00	0	0.00	0	0.00	0	0.00
SetRelAlarm									
an alarm defined	Alarm set	320	5.00	330	5.15	320	5.00	320	5.00
	Alarm expires immediately, notified task becomes ready	474	7.40	534	8.34	534	8.34	534	8.34
	Alarm expires immediately, notified task becomes running	718	11.22	1161	18.14	1084	16.94	1384	21.63
10 alarms are defined, 9 alarms are set on a counter	Alarm set	239	3.73	260	4.06	300	4.68	300	4.68
	Alarm expires immediately, notified task becomes ready	650	10.16	820	12.81	820	12.81	820	12.81
	Alarm expires immediately, notified task becomes running	878	13.72	1428	22.31	1344	21.00	1654	25.84
SetAbsAlarm									
an alarm defined	Alarm set	224	3.50	244	3.81	244	3.81	244	3.81
	Alarm expires immediately, notified task becomes ready	490	7.65	540	8.43	560	8.75	560	8.75
	Alarm expires immediately, notified task becomes running	718	11.22	1161	18.14	1084	16.94	1384	21.63
10 alarms are defined, 9 alarms are set on a counter	Alarm set	174	2.71	204	3.18	214	3.34	214	3.34
	Alarm expires immediately, notified task becomes ready	650	10.16	820	12.81	820	12.81	820	12.81
	Alarm expires immediately, notified task becomes running	878	13.72	1428	22.31	1344	21.00	1654	25.84

Table B.3 Run-time Services Latency (CW)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
CancelAlarm									
1 alarm is set on a counter		144	2.25	174	2.71	164	2.56	164	2.56
10 tasks, 10 alarms are set on a counter		134	2.09	164	2.56	144	2.25	144	2.25
GetAlarm									
1 alarm is set on a counter		84	1.31	84	1.31	94	1.46	94	1.46
10 tasks, 10 alarms are set on a counter		94	1.46	84	1.31	94	1.46	94	1.46
StartSCheduleTableRel									
ScheduleTable defined	Task activated, no rescheduling	1368	21.38	1418	22.16	1408	22.00	1408	22.00
	Task activated and rescheduled	1596	24.94	2046	31.97	1926	30.09	2236	34.94
StopScheduleTable									
ScheduleTable defined		407	6.35	437	6.82	427	6.67	427	6.67
CallTrustedFunction									
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	N/A	N/A	94	1.46	94	1.46

The table below contains interrupt frame characteristics for Freescale AUTOSAR OS / MPC56xxAM compiled with CodeWarrior

Interrupt frame characteristics are the data about delay from interrupt until start of first ISR command and from end of last ISR command to first instruction of Task for ISR category 2. The timing data for Time Protection interrupt - time from TP interrupt to ProtectionHook and from ProtectionHook to another Task in case current Task is killed are also presented.

Table B.4 Interrupt Frame Characteristics for ISRs (CW)

Conditions	SC1		SC2		SC3		SC4	
	cycles	µs	cycles	µs	cycles	µs	cycles	µs
From interrupt to 1st ISR cat.1 command	230,0	3.59	221,0	3.45	338,0	5.28	329,0	5.14
Return to interrupted task from ISR cat.1	162,0	2.53	173,0	2.70	282,0	4.40	232,0	3.62
From interrupt to 1st ISR cat.2 command	309,0	4.82	772,0	12.06	572,0	8.93	891,0	13.92
Return to interrupted task from ISR cat.2	299,0	4.67	658,0	10.28	618,0	9.65	759,0	11.86
Return with Task rescheduling	490,0	7.65	846,0	13.22	861,0	13.45	1150,0	17.97
Time Protection event occurs	N/A	N/A	490,0	7.65	N/A	N/A	653,0	10.20
Return from TP interrupt	N/A	N/A	918,0	14.34	N/A	N/A	1224,0	19.13

Data Sheets for GreenHills Compiler

The table below contains OS services timings in CPU cycles for Freescale AUTOSAR OS / MPC56xxAM compiled with GreenHills compiler.

The data for all Scalability classes are presented, for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown.

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		32743	38953	44753	45081	36540	36745
ActivateTask							
Initial	Task activated, no rescheduling	186	236	416	651	416	651
	Task activated and rescheduled	302	746	852	1083	1086	1283
Internal resource defined	Task activated and rescheduled	438	826	902	1133	1146	1387
SCHEDULE = NON for all tasks	Task activated, no rescheduling	126	146	346	571	346	571
TerminateTask							
Initial	Task terminated, return to lower prio task	166	606	602	897	752	1047
Initial	Task terminated, new task started	192	516	602	823	752	953
Internal resource defined	Task terminated, return to lower prio task	278	686	662	967	866	1161
Internal resource defined	Task terminated, new task started	294	626	682	893	872	1093
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	156	186	626	893	646	903
SCHEDULE = NON for all tasks	Task terminated, new task started	182	457	612	833	762	953

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
ChainTask							
Initial	Task terminated, next task started	192	586	792	1003	942	1133
Internal resource defined	Task terminated, next task started	314	686	862	1063	1072	1273
SCHEDULE = NON for all tasks	Task terminated, next task started	192	556	782	993	932	1123
Schedule							
Initial	No rescheduling, all tasks are preemptable	136	156	226	481	226	481
Internal resource defined	No rescheduling, all tasks are preemptable	248	256	336	591	336	601
SCHEDULE = NON for all tasks	No rescheduling	136	166	226	491	226	481
	Rescheduling, other task becomes running	222	636	676	907	916	1093
GetTaskId							
Initial		60	60	350	627	390	657
GetTaskState							
Initial	For running task	96	116	556	843	576	853
	For ready task	156	166	606	863	626	873
	For suspended task	156	186	626	893	646	903
EnableAllInterrupts							
Initial		8	8	113	388	123	378
DisableAllInterrupts							
Initial		20	20	130	385	140	375
ResumeAllInterrupts							
Initial		73	73	143	408	133	408
SuspendAllInterrupts							
Initial		80	80	130	415	130	405
ResumeOSInterrupts							
Initial		83	93	143	408	153	428

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SuspendOSInterrupts							
Initial		103	113	143	418	163	438
GetResource							
a resource defined	Task occupies resource	246	666	486	731	906	1141
call from ISR level	ISR occupies resource	243	716	483	758	956	1201
ReleaseResource							
a resource defined	Task release resource, no rescheduling	323	506	533	798	686	961
	Task release resource, other task becomes running	463	1069	1040	1240	1463	1713
call from ISR level	ISR release resource	313	546	563	818	736	991
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	216	226	426	671	426	671
	Event set, task becomes ready	266	286	486	721	506	731
CC = ECC1 an event defined	Event set, but task was not waiting it	276	276	456	701	486	751
	Event set, task becomes ready	336	346	526	761	566	811
	Event set, task becomes running	666	1012	986	1271	1339	1627
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		96	106	226	451	236	451
CC = ECC1 an event defined		106	106	226	461	226	451

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		70	80	520	797	550	807
CC = ECC1 an event defined		80	90	530	807	530	817
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	146	276	246	481	396	631
	Task becomes waiting	556	780	736	1041	969	1277
CC = ECC1 an event defined	Event was already set, task remains running	136	286	236	491	386	651
	Task becomes waiting	546	762	705	1040	959	1307
SendMessage							
message defined	Message size 16 bytes	296	296	706	963	706	983
ReceiveMessage							
message defined	Message size 16 bytes	296	296	686	953	706	953
InitCounter							
a counter defined		60	60	276	501	276	481
IncrementCounter							
a counter defined		152	152	292	547	312	557
1 alarm is set to a counter	Alarm is not expired	362	372	542	797	562	827
	Notified task becomes ready	512	552	722	967	742	997
	Notified task becomes running	608	1036	1188	1449	1428	1659

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
10 tasks, 10 alarms are set to a counter	Alarm is not expired	832	752	1042	1287	1152	1167
	Notified tasks become ready	1932	2092	2432	2657	2452	2537
	High priority notified task becomes running	2058	2662	2952	3183	3182	3273
GetCounterValue							
a counter defined		70	70	490	767	480	747
GetCounterInfo							
a counter defined		92	92	550	827	530	817
GetAlarmBase							
an alarm defined		72	92	518	785	508	795
SetRelAlarm							
an alarm defined	Alarm set	452	502	682	927	682	947
	Alarm expires immediately, notified task becomes ready	646	696	932	1167	932	1177
	Alarm expires immediately, notified task becomes running	818	1276	1570	2076	1810	2306
10 alarms are defined, 9 alarms are set on a counter	Alarm set	422	462	692	937	672	917
	Alarm expires immediately, notified task becomes ready	1142	1102	1484	1729	1564	1619
	Alarm expires immediately, notified task becomes running	1238	1582	1934	2155	2234	2235
SetAbsAlarm							
an alarm defined	Alarm set	386	426	648	913	648	933
	Alarm expires immediately, notified task becomes ready	652	712	944	1199	944	1219
	Alarm expires immediately, notified task becomes running	738	1196	1390	1641	1620	1861

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
10 alarm are defined, 9 alarms are set on a counter	Alarm set	326	376	658	903	648	903
	Alarm expires immediately, notified task becomes ready	1132	1112	1474	1719	1554	1609
	Alarm expires immediately, notified task becomes running	1238	1572	1934	2155	2234	2235
CancelAlarm							
1 alarm is set on a counter		296	316	476	731	486	731
10 tasks, 10 alarms are set on a counter		296	296	476	711	476	721
GetAlarm							
1 alarm is set on a counter		186	196	656	913	646	873
10 tasks, 10 alarms are set on a counter		186	196	616	873	646	923
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	1410	1422	1646	1891	1636	1871
	Task activated, and rescheduled	1526	1922	2155	2366	2375	2543
StopScheduleTable							
ScheduleTable defined		539	549	699	934	689	924
GetApplicationID							
Initial		N/A	N/A	0	345	0	335
CheckObjectAccess							
Initial		N/A	N/A	110	465	110	475
CheckObjectOwnerShip							
Initial		N/A	N/A	80	385	70	395
CheckTaskMemoryAccess							
Initial		N/A	N/A	200	647	210	647

Table B.5 Run-time Services Timing in CPU cycles (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
CheckISRMemoryAccess							
Initial		N/A	N/A	140	647	140	667
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	368	709	358	709
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	348	2421	348	2753

The table below contains OS services timings in microseconds for Freescale AUTOSAR OS / MPC56xxAM compiled with GreenHills. The data is calculated from Table B.5 for 64 MHz CPU clock frequency.

The data for all Scalability classes are presented, for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown.

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		511.6	608.6	699.3	704.4	570.9	574.1
ActivateTask							
Initial	Task activated, no rescheduling	2.90	3.68	6.50	10.17	6.50	10.17
	Task activated and rescheduled	4.71	11.66	13.31	16.92	16.97	20.05
Internal resource defined	Task activated and rescheduled	6.84	12.91	14.09	17.70	17.91	21.67
SCHEDULE = NON for all tasks	Task activated, no rescheduling	1.96	2.28	5.40	8.92	5.40	8.92

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
TerminateTask							
Initial	Task terminated, return to lower prio task	2.59	9.46	9.40	14.02	11.75	16.36
Initial	Task terminated, new task started	3.00	8.06	9.40	12.86	11.75	14.89
Internal resource defined	Task terminated, return to lower prio task	4.34	10.72	10.34	15.11	13.53	18.14
Internal resource defined	Task terminated, new task started	4.59	9.78	10.66	13.95	13.63	17.08
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	2.43	2.90	9.78	13.95	10.09	14.11
SCHEDULE = NON for all tasks	Task terminated, new task started	2.84	7.14	9.56	13.02	11.91	14.89
ChainTask							
Initial	Task terminated, next task started	3.00	9.15	12.38	15.67	14.72	17.70
Internal resource defined	Task terminated, next task started	4.90	10.72	13.47	16.61	16.75	19.89
SCHEDULE = NON for all tasks	Task terminated, next task started	3.00	8.68	12.22	15.52	14.56	17.55
Schedule							
Initial	No rescheduling, all tasks are preemptable	2.12	2.43	3.53	7.51	3.53	7.51
Internal resource defined	No rescheduling, all tasks are preemptable	3.87	4.00	5.25	9.23	5.25	9.39
SCHEDULE = NON for all tasks	No rescheduling	2.12	2.59	3.53	7.67	3.53	7.51
	Rescheduling, other task becomes running	3.46	9.93	10.56	14.17	14.31	17.08
GetTaskId							
Initial		0.93	0.93	5.46	9.79	6.09	10.27

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetTaskState							
Initial	For running task	1.50	1.81	8.68	13.17	9.00	13.33
	For ready task	2.43	2.59	9.46	13.48	9.78	13.64
	For suspended task	2.43	2.90	9.78	13.95	10.09	14.11
EnableAllInterrupts							
Initial		0.12	0.12	1.76	6.06	1.92	5.90
DisableAllInterrupts							
Initial		0.31	0.31	2.03	6.01	2.18	5.85
ResumeAllInterrupts							
Initial		1.14	1.14	2.23	6.37	2.07	6.37
SuspendAllInterrupts							
Initial		1.25	1.25	2.03	6.48	2.03	6.32
ResumeOSInterrupts							
Initial		1.29	1.45	2.23	6.37	2.39	6.68
SuspendOSInterrupts							
Initial		1.60	1.76	2.23	6.53	2.54	6.84
GetResource							
a resource defined	Task occupies resource	3.84	10.41	7.59	11.42	14.16	17.83
call from ISR level	ISR occupies resource	3.79	11.19	7.54	11.84	14.94	18.77
ReleaseResource							
a resource defined	Task release resource, no rescheduling	5.04	7.90	8.32	12.47	10.72	15.02
	Task release resource, other task becomes running	7.23	16.70	16.25	19.38	22.86	26.77
call from ISR level	ISR release resource	4.89	8.53	8.79	12.78	11.50	15.48

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	3.37	3.53	6.65	10.48	6.65	10.48
	Event set, task becomes ready	4.15	4.46	7.59	11.27	7.90	11.42
CC = ECC1 an event defined	Event set, but task was not waiting it	4.31	4.31	7.12	10.95	7.59	11.73
	Event set, task becomes ready	5.25	5.40	8.21	11.89	8.84	12.67
	Event set, task becomes running	10.41	15.81	15.41	19.86	20.92	25.42
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		1.50	1.65	3.53	7.04	3.68	7.04
CC = ECC1 an event defined		1.65	1.65	3.53	7.20	3.53	7.04
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		1.09	1.25	8.12	12.45	8.59	12.61
CC = ECC1 an event defined		1.25	1.40	8.28	12.61	8.28	12.77
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	2.28	4.31	3.84	7.51	6.18	9.85
	Task becomes waiting	8.68	12.19	11.50	16.27	15.14	19.95
CC = ECC1 an event defined	Event was already set, task remains running	2.12	4.46	3.68	7.67	6.03	10.17
	Task becomes waiting	8.53	11.91	11.02	16.25	14.98	20.42

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SendMessage							
message defined	Message size 16 bytes	4.62	4.62	11.03	15.05	11.03	15.36
ReceiveMessage							
message defined	Message size 16 bytes	4.62	4.62	10.72	14.89	11.03	14.89
InitCounter							
a counter defined		0.93	0.93	4.31	7.82	4.31	7.51
IncrementCounter							
a counter defined		2.37	2.37	4.56	8.54	4.87	8.70
1 alarm is set to a counter	Alarm is not expired	5.65	5.81	8.46	12.45	8.78	12.92
	Notified task becomes ready	8.00	8.62	11.28	15.11	11.59	15.58
	Notified task becomes running	9.50	16.19	18.56	22.64	22.31	25.92
10 tasks, 10 alarms are set to a counter	Alarm is not expired	13.00	11.75	16.28	20.11	18.00	18.23
	Notified tasks become ready	30.19	32.69	38.00	41.52	38.31	39.64
	High priority notified task becomes running	32.16	41.59	46.13	49.73	49.72	51.14
GetCounterValue							
a counter defined		1.09	1.09	7.65	11.98	7.50	11.67
GetCounterInfo							
a counter defined		1.43	1.43	8.59	12.92	8.28	12.77
GetAlarmBase							
an alarm defined		1.12	1.43	8.09	12.27	7.93	12.42

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SetRelAlarm							
an alarm defined	Alarm set	7.06	7.84	10.66	14.48	10.66	14.80
	Alarm expires immediately, notified task becomes ready	10.09	10.88	14.56	18.23	14.56	18.39
	Alarm expires immediately, notified task becomes running	12.78	19.94	24.53	32.44	28.28	36.03
10 alarms are defined, 9 alarms are set on a counter	Alarm set	6.59	7.21	10.81	14.64	10.50	14.33
	Alarm expires immediately, notified task becomes ready	17.84	17.22	23.19	27.02	24.44	25.30
	Alarm expires immediately, notified task becomes running	19.34	24.72	30.22	33.67	34.91	34.92
SetAbsAlarm							
an alarm defined	Alarm set	6.03	6.65	10.13	14.27	10.13	14.58
	Alarm expires immediately, notified task becomes ready	10.19	11.13	14.75	18.73	14.75	19.05
	Alarm expires immediately, notified task becomes running	11.53	18.69	21.72	25.64	25.31	29.08
10 alarms are defined, 9 alarms are set on a counter	Alarm set	5.09	5.87	10.28	14.11	10.13	14.11
	Alarm expires immediately, notified task becomes ready	17.69	17.38	23.03	26.86	24.28	25.14
	Alarm expires immediately, notified task becomes running	19.34	24.56	30.22	33.67	34.91	34.92
CancelAlarm							
1 alarm is set on a counter		4.62	4.93	7.43	11.42	7.59	11.42
10 tasks, 10 alarms are set on a counter		4.62	4.62	7.43	11.11	7.43	11.27

Table B.6 Run-time Services Timing in microseconds (GreenHills)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetAlarm							
1 alarm is set on a counter		2.90	3.06	10.25	14.27	10.09	13.64
10 tasks, 10 alarms are set on a counter		2.90	3.06	9.62	13.64	10.09	14.42
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	22.03	22.22	25.72	29.55	25.56	29.23
	Task activated, and rescheduled	23.84	30.03	33.67	36.97	37.11	39.73
StopScheduleTable							
ScheduleTable defined		8.42	8.57	10.92	14.59	10.77	14.44
GetApplicationID							
Initial		N/A	N/A	0	5.39	0	5.23
CheckObjectAccess							
Initial		N/A	N/A	1.71	7.26	1.71	7.42
CheckObjectOwnerShip							
Initial		N/A	N/A	1.25	6.01	1.09	6.17
CheckTaskMemoryAccess							
Initial		N/A	N/A	3.12	10.11	3.28	10.11
CheckISRMemoryAccess							
Initial		N/A	N/A	2.18	10.11	2.18	10.42
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	5.75	11.08	5.59	11.08
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	5.43	37.83	5.43	43.02

The table below contains OS services latency for Freescale AUTOSAR OS / MPC56xxAM compiled with GreenHills. For classes SC3 and SC4 latency was measured only for trusted Application. The data in “µs” columns are calculated for 64 MHz CPU clock.

The Latency is the time for which the interrupts of category 2 are disabled in the service. Interrupts of category 1 are disabled inside OS services that cause rescheduling only for SC3 and SC4, except StartOS, which disables all interrupts for the time given in the table. The ISRs of category 1 are disabled for the time needed to re-configure MPU for given OS-Application, it is about 60 cycles in case of switching to non-trusted OS-Application.

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
StartOS									
Initial		32456	507.1	38376	599.6	34610	540.8	35628	556.7
ActivateTask									
Initial	Task activated, no rescheduling	115	1.79	165	2.57	155	2.42	155	2.42
	Task activated and rescheduled	193	3.01	653	10.20	599	9.35	802	12.53
Internal resource defined	Task activated and rescheduled	349	5.45	736	11.50	642	10.03	892	13.94
SCHEDULE = NON for all tasks	Task activated, no rescheduling	85	1.32	124	1.93	95	1.48	95	1.48
TerminateTask									
Initial	Task terminated, return to lower prio task	115	1.79	538	8.40	454	7.09	614	9.59
Initial	Task terminated, new task started	73	1.14	413	6.45	429	6.70	579	9.04
Internal resource defined	Task terminated, return to lower prio task	207	3.23	648	10.13	531	8.29	734	11.47
Internal resource defined	Task terminated, new task started	175	2.73	543	8.48	499	7.79	709	11.08
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	84	1.31	124	1.93	95	1.48	95	1.48

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
SCHEDULE = NON for all tasks	Task terminated, new task started	53	0.82	374	5.84	419	6.54	559	8.73
ChainTask									
Initial	Task terminated, next task started	83	1.29	463	7.23	499	7.79	649	10.14
Internal resource defined	Task terminated, next task started	195	3.04	583	9.10	569	8.89	779	12.17
SCHEDULE = NON for all tasks	Task terminated, next task started	73	1.14	463	7.23	479	7.48	619	9.67
Schedule									
Initial	No rescheduling, all tasks are preemptable	43	0.67	63	0.98	93	1.45	93	1.45
Internal resource defined	No rescheduling, all tasks are preemptable	175	2.73	173	2.70	193	3.01	193	3.01
SCHEDULE = NON for all tasks	No rescheduling	33	0.51	63	0.98	73	1.14	73	1.14
	Rescheduling, other task becomes running	124	1.93	567	8.85	519	8.10	729	11.39
GetTaskId									
Initial		0	0.00	0	0.00	0	0.00	0	0.00
GetTaskState									
Initial	For running task	44	0.68	54	0.84	55	0.85	55	0.85
	For ready task	84	1.31	104	1.62	85	1.32	85	1.32
	For suspended task	84	1.31	124	1.93	95	1.48	95	1.48
GetResource									
a resource defined	Task occupies resource	95	1.48	465	7.26	115	1.79	475	7.42
call from ISR level	ISR occupies resource	102	1.59	515	8.04	112	1.75	505	7.89

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
ReleaseResource									
a resource defined	Task release resource, no rescheduling	152	2.37	345	5.39	172	2.68	345	5.39
	Task release resource, other task becomes running	300	4.68	886	13.84	646	10.09	1093	17.08
call from ISR level	ISR release resource	162	2.53	375	5.85	172	2.68	365	5.70
SetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	105	1.64	95	1.48	115	1.79	115	1.79
	Event set, task becomes ready	165	2.57	165	2.57	175	2.73	185	2.89
CC = ECC1 an event defined	Event set, but task was not waiting it	145	2.26	165	2.57	165	2.57	165	2.57
	Event set, task becomes ready	205	3.20	235	3.67	225	3.51	235	3.67
	Event set, task becomes running	550	8.59	916	14.31	666	10.41	982	15.34
ClearEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		54	0.84	45	0.70	55	0.85	64	1.00
CC = ECC1 an event defined		45	0.70	54	0.84	55	0.85	55	0.85
GetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		0	0.00	0	0.00	0	0.00	0	0.00
CC = ECC1 an event defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
WaitEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	54	0.84	45	0.70	55	0.85	64	1.00
	Task becomes waiting	460	7.18	727	11.36	553	8.64	769	12.02
CC = ECC1 an event defined	Event was already set, task remains running	43	0.67	183	2.85	53	0.82	193	3.01
	Task becomes waiting	472	7.37	718	11.22	538	8.40	785	12.27
SendMessage									
message defined	Message size 16 bytes	192	3.00	192	3.00	195	3.04	195	3.04
ReceiveMessage									
message defined	Message size 16 bytes	164	2.56	164	2.56	195	3.04	195	3.04
InitCounter									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
IncrementCounter									
a counter defined		91	1.42	91	1.42	91	1.42	91	1.42
1 alarm is set to a counter	Alarm is not expired	271	4.23	281	4.39	311	4.85	311	4.85
	Notified task becomes ready	421	6.57	451	7.04	481	7.51	481	7.51
	Notified task becomes running	489	7.64	939	14.67	938	14.66	1145	17.89
10 tasks, 10 alarms are set to a counter	Alarm is not expired	741	11.58	661	10.33	891	13.92	891	13.92
	Notified tasks become ready	1841	28.77	1991	31.11	2181	34.08	2181	34.08
	High priority notified task becomes running	1939	30.30	2569	40.14	2668	41.69	2888	45.13
GetCounterValue									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
GetCounterInfo									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
GetAlarmBase									
an alarm defined		0	0.00	0	0.00	0	0.00	0	0.00
SetRelAlarm									
an alarm defined	Alarm set	371	5.79	381	5.95	355	5.54	355	5.54
	Alarm expires immediately, notified task becomes ready	565	8.82	595	9.29	615	9.60	615	9.60
	Alarm expires immediately, notified task becomes running	619	9.67	1069	16.70	1048	16.38	1255	19.61
10 alarms are defined, 9 alarms are set on a counter	Alarm set	341	5.32	351	5.48	365	5.70	365	5.70
	Alarm expires immediately, notified task becomes ready	1061	16.58	1001	15.64	1211	18.92	1211	18.92
	Alarm expires immediately, notified task becomes running	1109	17.33	1459	22.80	1628	25.44	1848	28.88
SetAbsAlarm									
an alarm defined	Alarm set	315	4.92	325	5.07	325	5.07	325	5.07
	Alarm expires immediately, notified task becomes ready	571	8.92	611	9.54	621	9.70	621	9.70
	Alarm expires immediately, notified task becomes running	619	9.67	1069	16.70	1048	16.38	1255	19.61
10 alarms are defined, 9 alarms are set on a counter	Alarm set	255	3.98	265	4.14	315	4.92	315	4.92
	Alarm expires immediately, notified task becomes ready	1061	16.58	1001	15.64	1211	18.92	1211	18.92
	Alarm expires immediately, notified task becomes running	1109	17.33	1459	22.80	1628	25.44	1848	28.88

Table B.7 Run-time Services Latency (GreenHills)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
CancelAlarm									
1 alarm is set on a counter		194	3.03	195	3.04	235	3.67	235	3.67
10 tasks, 10 alarms are set on a counter		184	2.87	185	2.89	205	3.20	205	3.20
GetAlarm									
1 alarm is set on a counter		64	1.00	65	1.01	75	1.17	75	1.17
10 tasks, 10 alarms are set on a counter		64	1.00	65	1.01	75	1.17	75	1.17
StartSCheduleTableRel									
ScheduleTable defined	Task activated, no rescheduling	1278	19.97	1310	20.47	1301	20.33	1301	20.33
	Task activated, and rescheduled	1356	21.19	1798	28.09	1755	27.42	1959	30.61
StopScheduleTable									
ScheduleTable defined		417	6.51	438	6.84	418	6.53	418	6.53
CallTrustedFunction									
TRUSTED_FUNCTION defined	the body of user function is empty	N/A	N/A	N/A	N/A	85	1.32	85	1.32

The table below contains interrupt frame characteristics for Freescale AUTOSAR OS / MPC56xxAM compiled with GreenHills.

Interrupt frame characteristics are the data about delay from interrupt until start of first ISR command and from end of last ISR command to first instruction of Task for ISR category 2. The timing data for Time Protection interrupt - time from TP interrupt to ProtectionHook and from ProtectionHook to another Task in case current Task is killed are also presented.

Table B.8 Interrupt Frame Characteristics for ISRs (GreenHills)

Conditions	SC1		SC2		SC3		SC4	
	cycles	µs	cycles	µs	cycles	µs	cycles	µs
From interrupt to 1st ISR cat.1 command	250,0	3.90	240,0	3.75	269,0	4.20	260,0	4.06
Return to interrupted task from ISR cat.1	153,0	2.39	153,0	2.39	203,0	3.17	154,0	2.40
From interrupt to 1st ISR cat.2 command	330,0	5.15	780,0	12.19	533,0	8.32	802,0	12.53
Return to interrupted task from ISR cat.2	279,0	4.35	519,0	8.10	509,0	7.95	669,0	10.45
Return with Task rescheduling	361,0	5.64	742,0	11.59	842,0	13.16	992,0	15.50
Time Protection event occurs	N/A	N/A	516,0	8.06	N/A	N/A	562,0	8.78
Return from TP interrupt	N/A	N/A	709,0	11.08	N/A	N/A	1035,0	16.17

Data Sheets for WindRiver Compiler

The table below contains OS services timings in CPU cycles for Freescale AUTOSAR OS / MPC56xxAM compiled with WindRiver.

The data for all Scalability classes are presented., for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		37464	38958	45004	45191	46129	46413
ActivateTask							
Initial	Task activated, no rescheduling	225	234	446	711	446	711
	Task activated and rescheduled	321	944	886	1083	1315	1513
Internal resource defined	Task activated and rescheduled	436	1048	972	1173	1425	1597
SCHEDULE = NON for all tasks	Task activated, no rescheduling	175	185	426	700	425	691
TerminateTask							
Initial	Task terminated, return to lower prio task	215	684	692	1000	972	1271
Initial	Task terminated, new task started	241	604	632	832	902	1102
Internal resource defined	Task terminated, return to lower prio task	376	888	826	1160	1162	1467
Internal resource defined	Task terminated, new task started	422	805	792	1002	1091	1293
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	215	205	775	983	765	1003
SCHEDULE = NON for all tasks	Task terminated, new task started	211	566	632	853	922	1093
ChainTask							
Initial	Task terminated, next task started	221	665	802	1002	1072	1272

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
Internal resource defined	Task terminated, next task started	383	915	991	1202	1282	1493
SCHEDULE = NON for all tasks	Task terminated, next task started	211	695	831	1053	1112	1292
Schedule							
Initial	No rescheduling, all tasks are preemptable	145	174	295	551	285	541
Internal resource defined	No rescheduling, all tasks are preemptable	336	335	406	691	435	720
SCHEDULE = NON for all tasks	No rescheduling	145	165	285	531	286	550
	Rescheduling, other task becomes running	251	845	734	923	1142	1366
GetTaskId							
Initial		59	69	550	797	549	796
GetTaskState							
Initial	For running task	164	165	746	952	736	972
	For ready task	215	205	786	992	776	1012
	For suspended task	215	205	775	983	765	1003
EnableAllInterrupts							
Initial		27	32	163	438	173	448
DisableAllInterrupts							
Initial		89	109	270	545	270	545
ResumeAllInterrupts							
Initial		112	122	183	458	183	458
SuspendAllInterrupts							
Initial		189	179	270	545	270	545
ResumeOSInterrupts							
Initial		102	132	173	428	223	478
SuspendOSInterrupts							
Initial		112	142	223	478	233	488

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetResource							
a resource defined	Task occupies resource	295	695	627	861	977	1251
call from ISR level	ISR occupies resource	302	755	684	928	1077	1351
ReleaseResource							
a resource defined	Task release resource, no rescheduling	342	725	653	908	1006	1261
	Task release resource, other task becomes running	461	1464	1099	1260	1936	2126
call from ISR level	ISR release resource	352	725	683	948	1046	1311
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	265	265	526	771	526	781
	Event set, task becomes ready	324	334	566	811	586	841
CC = ECC1 an event defined	Event set, but task was not waiting it	305	305	566	821	586	831
	Event set, task becomes ready	365	364	606	861	636	881
	Event set, task becomes running	754	1211	1079	1387	1632	1914
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		125	125	296	541	276	541
CC = ECC1 an event defined		135	125	286	551	296	541
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		89	89	710	947	700	937

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
CC = ECC1 an event defined		89	89	700	967	700	967
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	155	385	316	571	546	801
	Task becomes waiting	582	901	832	1124	1208	1526
CC = ECC1 an event defined	Event was already set, task remains running	154	385	326	581	555	810
	Task becomes waiting	585	911	819	1127	1182	1474
SendMessage							
message defined	Message size 16 bytes	494	485	1026	1262	1036	1242
ReceiveMessage							
message defined	Message size 16 bytes	474	475	996	1263	1006	1243
InitCounter							
a counter defined		59	59	386	661	376	641
IncrementCounter							
a counter defined		181	180	422	697	432	687
1 alarm is set to a counter	Alarm is not expired	360	370	612	867	622	877
	Notified task becomes ready	511	530	752	1017	772	1037
	Notified task becomes running	657	1231	1198	1373	1638	1812
10 tasks, 10 alarms are set to a counter	Alarm is not expired	671	781	922	1167	912	1157
	Notified tasks become ready	2021	2051	2282	2537	2172	2427
	High priority notified task becomes running	2097	2803	2772	2989	3111	3329
GetCounterValue							
a counter defined		69	78	649	867	650	896

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetCounterInfo							
a counter defined		110	101	764	981	750	987
GetAlarmBase							
an alarm defined		123	123	717	955	727	935
SetRelAlarm							
an alarm defined	Alarm set	601	571	908	1172	908	1172
	Alarm expires immediately, notified task becomes ready	705	705	1032	1287	1042	1297
	Alarm expires immediately, notified task becomes running	937	1501	1760	2200	2180	2619
10 alarms are defined, 9 alarms are set on a counter	Alarm set	521	581	878	1153	888	1163
	Alarm expires immediately, notified task becomes ready	1011	1111	1374	1629	1384	1639
	Alarm expires immediately, notified task becomes running	1067	1784	1784	1981	2233	2431
SetAbsAlarm							
an alarm defined	Alarm set	435	445	788	1043	798	1053
	Alarm expires immediately, notified task becomes ready	730	711	1083	1339	1083	1339
	Alarm expires immediately, notified task becomes running	827	1401	1480	1665	1910	2094
10 alarm are defined, 9 alarms are set on a counter	Alarm set	365	365	777	1033	777	1033
	Alarm expires immediately, notified task becomes ready	1011	1121	1374	1629	1384	1639
	Alarm expires immediately, notified task becomes running	1076	1774	1774	1990	2223	2440

Table B.9 Run-time Services Timing in CPU cycles (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
CancelAlarm							
1 alarm is set on a counter		274	325	516	771	526	781
10 tasks, 10 alarms are set on a counter		285	274	515	770	505	760
GetAlarm							
1 alarm is set on a counter		215	215	766	1032	746	982
10 tasks, 10 alarms are set on a counter		215	214	755	993	775	1043
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	1469	1485	1762	2037	1772	2047
	Task activated, and rescheduled	1585	2188	2221	2439	2661	2879
StopScheduleTable							
ScheduleTable defined		578	598	799	1054	789	1044
GetApplicationID							
Initial		N/A	N/A	0	475	0	455
CheckObjectAccess							
Initial		N/A	N/A	190	655	210	645
CheckObjectOwnerShip							
Initial		N/A	N/A	110	505	120	525
CheckTaskMemoryAccess							
Initial		N/A	N/A	400	807	370	797
CheckISRMemoryAccess							
Initial		N/A	N/A	260	847	240	847
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of the function is empty	N/A	N/A	538	779	548	779
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	578	2461	548	3074

The table below contains OS services timings in CPU cycles for Freescale AUTOSAR OS / MPC56xxAM compiled with WindRiver. The data is calculated from Table B.5 for 64 MHz CPU clock frequency.

The data for all Scalability classes are presented, for SC3 and SC4 data for calls from trusted and non-trusted OS-Applications are shown.

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
StartOS							
Initial		585.4	608.7	703.2	706.1	720.8	725.2
ActivateTask							
Initial	Task activated, no rescheduling	3.51	3.65	6.96	11.11	6.96	11.11
	Task activated and rescheduled	5.01	14.75	13.84	16.92	20.55	23.64
Internal resource defined	Task activated and rescheduled	6.81	16.38	15.19	18.33	22.27	24.95
SCHEDULE = NON for all tasks	Task activated, no rescheduling	2.73	2.89	6.65	10.94	6.64	10.80
TerminateTask							
Initial	Task terminated, return to lower prio task	3.35	10.69	10.81	15.63	15.19	19.86
Initial	Task terminated, new task started	3.76	9.43	9.87	13.00	14.09	17.22
Internal resource defined	Task terminated, return to lower prio task	5.87	13.88	12.91	18.13	18.16	22.92
Internal resource defined	Task terminated, new task started	6.59	12.58	12.38	15.66	17.05	20.20
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	3.35	3.20	12.11	15.36	11.95	15.67
SCHEDULE = NON for all tasks	Task terminated, new task started	3.29	8.84	9.87	13.33	14.41	17.08

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
ChainTask							
Initial	Task terminated, next task started	3.45	10.39	12.53	15.66	16.75	19.88
Internal resource defined	Task terminated, next task started	5.98	14.30	15.48	18.78	20.03	23.33
SCHEDULE = NON for all tasks	Task terminated, next task started	3.29	10.86	12.98	16.45	17.38	20.19
Schedule							
Initial	No rescheduling, all tasks are preemptable	2.26	2.71	4.60	8.60	4.45	8.45
Internal resource defined	No rescheduling, all tasks are preemptable	5.25	5.23	6.34	10.80	6.79	11.25
SCHEDULE = NON for all tasks	No rescheduling	2.26	2.57	4.45	8.29	4.46	8.59
	Rescheduling, other task becomes running	3.92	13.20	11.47	14.42	17.84	21.34
GetTaskId							
Initial		0.92	1.07	8.59	12.45	8.57	12.44
GetTaskState							
Initial	For running task	2.56	2.57	11.66	14.88	11.50	15.19
	For ready task	3.35	3.20	12.28	15.50	12.13	15.81
	For suspended task	3.35	3.20	12.11	15.36	11.95	15.67
EnableAllInterrupts							
Initial		0.42	0.50	2.54	6.84	2.70	7.00
DisableAllInterrupts							
Initial		1.39	1.70	4.21	8.51	4.21	8.51
ResumeAllInterrupts							
Initial		1.75	1.90	2.85	7.15	2.85	7.15
SuspendAllInterrupts							
Initial		2.95	2.79	4.21	8.51	4.21	8.51
ResumeOSInterrupts							
Initial		1.59	2.06	2.70	6.68	3.48	7.46

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
SuspendOSInterrupts							
Initial		1.75	2.21	3.48	7.46	3.64	7.62
GetResource							
a resource defined	Task occupies resource	4.60	10.86	9.79	13.45	15.27	19.55
call from ISR level	ISR occupies resource	4.71	11.80	10.69	14.50	16.83	21.11
ReleaseResource							
a resource defined	Task release resource, no rescheduling	5.34	11.33	10.20	14.19	15.72	19.70
	Task release resource, other task becomes running	7.20	22.88	17.17	19.69	30.25	33.22
call from ISR level	ISR release resource	5.50	11.33	10.67	14.81	16.34	20.48
SetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	4.14	4.14	8.21	12.05	8.21	12.20
	Event set, task becomes ready	5.06	5.21	8.84	12.67	9.15	13.14
CC = ECC1 an event defined	Event set, but task was not waiting it	4.76	4.76	8.84	12.83	9.15	12.98
	Event set, task becomes ready	5.70	5.68	9.46	13.45	9.93	13.77
	Event set, task becomes running	11.78	18.92	16.86	21.67	25.50	29.91
ClearEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		1.95	1.95	4.62	8.45	4.31	8.45
CC = ECC1 an event defined		2.10	1.95	4.46	8.60	4.62	8.45

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
GetEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined		1.39	1.39	11.09	14.80	10.94	14.64
CC = ECC1 an event defined		1.39	1.39	10.94	15.11	10.94	15.11
WaitEvent							
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	2.42	6.01	4.93	8.92	8.53	12.52
	Task becomes waiting	9.09	14.08	13.00	17.56	18.88	23.84
CC = ECC1 an event defined	Event was already set, task remains running	2.40	6.01	5.09	9.07	8.67	12.66
	Task becomes waiting	9.14	14.23	12.80	17.61	18.47	23.03
SendMessage							
message defined	Message size 16 bytes	7.71	7.57	16.03	19.72	16.19	19.41
ReceiveMessage							
message defined	Message size 16 bytes	7.40	7.42	15.56	19.73	15.72	19.42
InitCounter							
a counter defined		0.92	0.92	6.03	10.33	5.87	10.02
IncrementCounter							
a counter defined		2.82	2.81	6.59	10.89	6.75	10.73
1 alarm is set to a counter	Alarm is not expired	5.62	5.78	9.56	13.55	9.71	13.70
	Notified task becomes ready	7.98	8.28	11.75	15.89	12.06	16.20
	Notified task becomes running	10.27	19.23	18.72	21.45	25.59	28.31

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
10 tasks, 10 alarms are set to a counter	Alarm is not expired	10.48	12.20	14.41	18.23	14.25	18.08
	Notified tasks become ready	31.58	32.05	35.66	39.64	33.94	37.92
	High priority notified task becomes running	32.77	43.80	43.31	46.70	48.61	52.02
GetCounterValue							
a counter defined		1.07	1.21	10.14	13.55	10.16	14.00
GetCounterInfo							
a counter defined		1.71	1.57	11.94	15.33	11.72	15.42
GetAlarmBase							
an alarm defined		1.92	1.92	11.20	14.92	11.36	14.61
SetRelAlarm							
an alarm defined	Alarm set	9.39	8.92	14.19	18.31	14.19	18.31
	Alarm expires immediately, notified task becomes ready	11.02	11.02	16.13	20.11	16.28	20.27
	Alarm expires immediately, notified task becomes running	14.64	23.45	27.50	34.38	34.06	40.92
10 alarms are defined, 9 alarms are set on a counter	Alarm set	8.14	9.07	13.72	18.02	13.88	18.17
	Alarm expires immediately, notified task becomes ready	15.80	17.36	21.47	25.45	21.63	25.61
	Alarm expires immediately, notified task becomes running	16.67	27.88	27.88	30.95	34.89	37.98
SetAbsAlarm							
an alarm defined	Alarm set	6.79	6.95	12.31	16.30	12.47	16.45
	Alarm expires immediately, notified task becomes ready	11.41	11.11	16.92	20.92	16.92	20.92
	Alarm expires immediately, notified task becomes running	12.92	21.89	23.13	26.02	29.84	32.72

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
10 alarms are defined, 9 alarms are set on a counter	Alarm set	5.70	5.70	12.14	16.14	12.14	16.14
	Alarm expires immediately, notified task becomes ready	15.80	17.52	21.47	25.45	21.63	25.61
	Alarm expires immediately, notified task becomes running	16.81	27.72	27.72	31.09	34.73	38.13
CancelAlarm							
1 alarm is set on a counter		4.28	5.07	8.06	12.05	8.21	12.20
10 tasks, 10 alarms are set on a counter		4.45	4.28	8.04	12.03	7.89	11.88
GetAlarm							
1 alarm is set on a counter		3.35	3.35	11.97	16.13	11.66	15.34
10 tasks, 10 alarms are set on a counter		3.35	3.34	11.80	15.52	12.11	16.30
StartSCheduleTableRel							
ScheduleTable defined	Task activated, no rescheduling	22.95	23.20	27.53	31.83	27.69	31.98
	Task activated, and rescheduled	24.77	34.19	34.70	38.11	41.58	44.98
StopScheduleTable							
ScheduleTable defined		9.03	9.34	12.48	16.47	12.33	16.31
GetApplicationID							
Initial		N/A	N/A	0	7.42	0	7.10
CheckObjectAccess							
Initial		N/A	N/A	2.96	10.23	3.28	10.08
CheckObjectOwnerShip							
Initial		N/A	N/A	1.71	7.89	1.87	8.20
CheckTaskMemoryAccess							
Initial		N/A	N/A	6.25	12.61	5.78	12.45

Table B.10 Run-time Services Timing in microseconds (WindRiver)

Configuration	Conditions	SC1	SC2	SC3		SC4	
				Trust	NonTr	Trust	NonTr
CheckISRMemoryAccess							
Initial		N/A	N/A	4.06	13.23	3.75	13.23
CallTrustedFunction							
TRUSTED_FUNCTION defined	the body of the function is empty	N/A	N/A	8.40	12.17	8.56	12.17
TerminaneApplication							
RESTARTTASK defined	called with RESTART	N/A	N/A	9.03	38.45	8.56	48.03

The table below contains OS services latency for Freescale AUTOSAR OS / MPC56xxAM compiled with WindRiver. For classes SC3 and SC4 latency is measured only for trusted Application. The data in "µs" columns is calculated for 64 MHz CPU clock.

The Latency is the time for which the interrupts of category 2 are disabled in the service. Interrupts of category 1 are not disabled inside OS services that cause rescheduling only for SC3 and SC4, except StartOS, which disables all interrupts for the time given in the table. The ISRs of category 1 are disabled for the time needed to re-configure MPU for given OS-Application, it is about 60 cycles in case of switching to non-trusted OS-Application.

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
StartOS									
Initial		37197	581.2	38317	598.7	39590	618.6	45362	708.8
ActivateTask									
Initial	Task activated, no rescheduling	144	2.25	154	2.40	135	2.10	135	2.10
	Task activated and rescheduled	202	3.15	825	12.89	562	8.78	1003	15.67
Internal resource defined	Task activated and rescheduled	308	4.81	941	14.70	638	9.96	1053	16.45
SCHEDULE = NON for all tasks	Task activated, no rescheduling	113	1.76	123	1.92	85	1.32	85	1.32
TerminateTask									
Initial	Task terminated, return to lower prio task	144	2.25	607	9.48	454	7.09	715	11.17
Initial	Task terminated, new task started	112	1.75	502	7.84	419	6.54	689	10.77
Internal resource defined	Task terminated, return to lower prio task	306	4.78	824	12.88	631	9.85	925	14.45
Internal resource defined	Task terminated, new task started	284	4.43	702	10.97	599	9.35	879	13.73
SCHEDULE = NON for all tasks	Task terminated, return to lower prio task	134	2.09	124	1.93	85	1.32	85	1.32

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
SCHEDULE = NON for all tasks	Task terminated, new task started	72	1.12	463	7.23	419	6.54	689	10.77
ChainTask									
Initial	Task terminated, next task started	112	1.75	542	8.46	489	7.64	759	11.86
Internal resource defined	Task terminated, next task started	274	4.28	792	12.38	659	10.30	948	14.81
SCHEDULE = NON for all tasks	Task terminated, next task started	102	1.59	542	8.46	489	7.64	759	11.86
Schedule									
Initial	No rescheduling, all tasks are preemptable	52	0.81	72	1.12	53	0.82	73	1.14
Internal resource defined	No rescheduling, all tasks are preemptable	233	3.64	241	3.76	203	3.17	213	3.32
SCHEDULE = NON for all tasks	No rescheduling	52	0.81	72	1.12	53	0.82	73	1.14
	Rescheduling, other task becomes running	122	1.90	725	11.33	509	7.95	943	14.73
GetTaskId									
Initial		0	0.00	0	0.00	0	0.00	0	0.00
GetTaskState									
Initial	For running task	74	1.15	84	1.31	55	0.85	55	0.85
	For ready task	144	2.25	134	2.09	85	1.32	85	1.32
	For suspended task	134	2.09	124	1.93	85	1.32	85	1.32
GetResource									
a resource defined	Task occupies resource	144	2.25	514	8.03	135	2.10	525	8.20
call from ISR level	ISR occupies resource	151	2.35	564	8.81	132	2.06	555	8.67

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
ReleaseResource									
a resource defined	Task release resource, no rescheduling	181	2.82	564	8.81	172	2.68	555	8.67
	Task release resource, other task becomes running	279	4.35	1282	20.03	606	9.46	1482	23.16
call from ISR level	ISR release resource	181	2.82	554	8.65	172	2.68	555	8.67
SetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event set, but task was not waiting it	133	2.07	124	1.93	114	1.78	105	1.64
	Event set, task becomes ready	183	2.85	184	2.87	154	2.40	165	2.57
CC = ECC1 an event defined	Event set, but task was not waiting it	173	2.70	164	2.56	154	2.40	155	2.42
	Event set, task becomes ready	223	3.48	214	3.34	194	3.03	205	3.20
	Event set, task becomes running	588	9.18	1058	16.53	672	10.50	1183	18.48
ClearEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		53	0.82	54	0.84	34	0.53	35	0.54
CC = ECC1 an event defined		53	0.82	54	0.84	34	0.53	34	0.53
GetEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined		0	0.00	0	0.00	0	0.00	0	0.00
CC = ECC1 an event defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
WaitEvent									
CC = ECC1 SCHEDULE = NON for all tasks an event defined	Event was already set, task remains running	53	0.82	54	0.84	34	0.53	35	0.54
	Task becomes waiting	502	7.84	798	12.47	543	8.48	933	14.58
CC = ECC1 an event defined	Event was already set, task remains running	62	0.96	282	4.40	43	0.67	263	4.10
	Task becomes waiting	491	7.67	810	12.66	564	8.81	914	14.28
SendMessage									
message defined	Message size 16 bytes	384	6.00	374	5.84	355	5.54	365	5.70
ReceiveMessage									
message defined	Message size 16 bytes	354	5.53	354	5.53	355	5.54	355	5.54
InitCounter									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
IncrementCounter									
a counter defined		119	1.85	119	1.85	91	1.42	81	1.26
1 alarm is set to a counter	Alarm is not expired	279	4.35	289	4.51	280	4.37	271	4.23
	Notified task becomes ready	429	6.70	449	7.01	430	6.71	431	6.73
	Notified task becomes running	527	8.23	1120	17.50	874	13.66	1295	20.23
10 tasks, 10 alarms are set to a counter	Alarm is not expired	590	9.21	690	10.78	571	8.92	580	9.06
	Notified tasks become ready	1940	30.31	1950	30.47	1941	30.33	1850	28.91
	High priority notified task becomes running	1978	30.91	2688	42.00	2438	38.09	2798	43.72
GetCounterValue									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
GetCounterInfo									
a counter defined		0	0.00	0	0.00	0	0.00	0	0.00
GetAlarmBase									
an alarm defined		0	0.00	0	0.00	0	0.00	0	0.00
SetRelAlarm									
an alarm defined	Alarm set	440	6.87	460	7.18	441	6.89	440	6.87
	Alarm expires immediately, notified task becomes ready	574	8.96	594	9.28	585	9.14	604	9.43
	Alarm expires immediately, notified task becomes running	648	10.13	1251	19.55	1015	15.86	1454	22.72
10 alarms are defined, 9 alarms are set on a counter	Alarm set	389	6.07	419	6.54	420	6.56	431	6.73
	Alarm expires immediately, notified task becomes ready	880	13.75	1010	15.78	910	14.22	911	14.23
	Alarm expires immediately, notified task becomes running	928	14.50	1638	25.59	1327	20.73	1769	27.64
SetAbsAlarm									
an alarm defined	Alarm set	314	4.90	334	5.21	315	4.92	334	5.21
	Alarm expires immediately, notified task becomes ready	600	9.37	610	9.53	611	9.54	620	9.68
	Alarm expires immediately, notified task becomes running	648	10.13	1251	19.55	1015	15.86	1454	22.72
10 alarms are defined, 9 alarms are set on a counter	Alarm set	244	3.81	274	4.28	314	4.90	305	4.76
	Alarm expires immediately, notified task becomes ready	880	13.75	1010	15.78	910	14.22	911	14.23
	Alarm expires immediately, notified task becomes running	928	14.50	1638	25.59	1327	20.73	1769	27.64

Table B.11 Run-time Services Latency (WindRiver)

Configuration	Conditions	SC1		SC2		SC3		SC4	
		cycles	µs	cycles	µs	cycles	µs	cycles	µs
CancelAlarm									
1 alarm is set on a counter		174	2.71	194	3.03	175	2.73	195	3.04
10 tasks, 10 alarms are set on a counter		174	2.71	204	3.18	185	2.89	165	2.57
GetAlarm									
1 alarm is set on a counter		83	1.29	83	1.29	75	1.17	65	1.01
10 tasks, 10 alarms are set on a counter		84	1.31	84	1.31	65	1.01	75	1.17
StartSCheduleTableRel									
ScheduleTable defined	Task activated, no rescheduling	1358	21.22	1373	21.45	1375	21.48	1375	21.48
	Task activated, and rescheduled	1426	22.28	2051	32.05	1822	28.47	2272	35.50
StopScheduleTable									
ScheduleTable defined		457	7.14	496	7.75	438	6.84	488	7.62
CallTrustedFunction									
TRUSTED_FUNCTION defined	the body of the function is empty	N/A	N/A	N/A	N/A	75	1.17	75	1.17

The table below contains interrupt frame characteristics for Freescale AUTOSAR OS / MPC56xxAM compiled with WindRiver

Interrupt frame characteristics are the data about delay from interrupt until start of first ISR command and from end of last ISR command to first instruction of Task for ISR category 2. The timing data for Time Protection interrupt - time from TP interrupt to ProtectionHook and from ProtectionHook to another Task in case current Task is killed are also presented.

Table B.12 Interrupt Frame Characteristics for ISRs (WindRiver)

Conditions	SC1		SC2		SC3		SC4	
	cycles	µs	cycles	µs	cycles	µs	cycles	µs
From interrupt to 1st ISR cat.1 command	259,0	4.04	269,0	4.20	260,0	4.06	229,0	3.57
Return to interrupted task from ISR cat.1	182,0	2.84	211,0	3.29	183,0	2.85	203,0	3.17
From interrupt to 1st ISR cat.2 command	389,0	6.07	1039,0	16.23	522,0	8.15	1058,0	16.53
Return to interrupted task from ISR cat.2	328,0	5.12	688,0	10.75	479,0	7.48	779,0	12.17
Return with Task rescheduling	390,0	6.09	881,0	13.77	852,0	13.31	1108,0	17.31
Time Protection event occurs	N/A	N/A	578,0	9.03	N/A	N/A	575,0	8.98
Return from TP interrupt	N/A	N/A	948,0	14.81	N/A	N/A	1216,0	19.00

C

Memory Requirements

This appendix provides information about the amount of ROM and RAM directly used by various versions of the Freescale AUTOSAR OS. The numbers in the tables may be not quite correct due to last minute changes in the OS code.

This appendix consists of the following sections:

Freescale AUTOSAR OS Memory Usage

The table below contains the data about ROM and RAM needed for the Operating System kernel and system objects. The amount of memory depends on the system configuration and on the number of certain objects (e.g., tasks, counters, etc.). The table does not reflects all possible configurations so the overall number of them is too big. Therefore, only some most important configurations are presented.

The following initial system property settings were used to determine memory requirements:

```
CC = BCC1;
STATUS = STANDARD;
STARTUPHOOK = FALSE;
SHUTDOWNHOOK = FALSE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
ERRORHOOK = FALSE;
USEGETSERVICEID = FALSE;
USEPARAMETERACCESS = FALSE;
USERESSCHEDULER = FALSE;
```

```
FastTerminate = TRUE;
STACKMONITORING = FALSE;
ACTIVATION = 1; (for all TASK objects)
SCHEDULER = FULL; (for all TASK objects)
no ISRs defined
```

This initial property list was used for the first row in the table. It conforms to the SC1, BCC1 classes without any additional mechanisms and this is the minimal AUTOSAR OS configuration. The rows below reflects memory requirements for the other OS configurations. System properties are shown in the rows which are turned on for the corresponded Conformance Class.

All other rows below the first one ("Initial") has a title "Initial" or "Changed:" and one or more options turned ON or OFF. If a row has a title "Initial" it means that for such OS configuration the Initial property list is used with particular options changed as shown. If a row has a title "Changed:" it means that for such OS configuration the setting list as for the previous row is used with particular options changed as shown.

Since each system object (a task, a message, an alarm, etc.) requires some ROM and RAM the total amount of memory depends on the number of objects. Therefore, the formulas should be used to calculate the exact memory amount for each case. These formulas are provided in the table.

Data presented in the table do not include ISR, main and Task's stacks for RAM; the compiler library and startup code, task functions and the primary vector table for ROM. However, the stacks used by OS internaly for protection handlers are incuded into RAM consumption numbers.

The RAM and ROM memory data does not include compiler padding of C variables for alignment, so actual RAM size may be more then calculated.

The following symbols are used in the formulas in the tables:

- T - number of Tasks
- Tn - number of Tasks belonging to nontrusted Applications
- Rt - number of task's Resources

- Ri - number of ISR‘s Resources
- C - number of Counters
- A - number of Alarms
- M - total number of Messages
- Mr - number of receive Messages
- I1 - number of category 1 ISRs
- I2 - number of category 2 ISRs
- I2n - number of category 2 ISRs belonging to nontrusted Applications
- IPL - number of interrupt processing levels
- Lr - number of timing protection resource locks objects
- V - number of tasks and ISRs category 2 with arrival rate
- AppN - number of nontrusted Applications

Data Sheets for SC1 Scalability Class

Table C.1 OS Memory Requirements for CodeWarrior, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1b0	Initial	**BCC1**	1460+5*T	21+1*T
1b1	Initial, STATUS=EXTENTED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS=TRUE		3060+8*T	126+96*T
1b2	Initial, Alarm with ACTION=ACTIVATETASK defined		2620+5*T+8*C+4*A	21+1*T+12*C+20*A
1b3	Initial, Message with ACTION=NONE defined		1832+5*T+4*Ms+14*Mr	22+1*T+4*Ms+12*Mr
1b4	Initial, Message with ACTION=ACTIVATETASK defined		1908+5*T+4*Ms+18*Mr	22+1*T+4*Ms+16*Mr
1b5	Initial, Resource for task defined		2296+8*T+1*R	125+104*T+12*R
1b6	Changed: ISR category 2 defined		150+104*T+16*I+4*IPL+12*R	150+104*T+16*I+4*IPL+12*R
1b7	Changed: Resource referenced by ISR		152+104*T+16*I+4*IPL+16*R	152+104*T+16*I+4*IPL+16*R

Table C.1 OS Memory Requirements for CodeWarrior, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1e0	Initial, Event defined	**ECC1**	129+108*T	129+108*T
1e1	Changed: STATUS=EXTENDED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS= TRUE		138+108*T	138+108*T
1e2	Initial, Event defined Alarm with ACTION=ACTIVATETASK defined		3680+12*T+8*C+12*A	129+108*T+8*C+28*A
1e3	Initial, Event defined Message with ACTION=NONE defined		2672+12*T+4*Ms+14*Mr	130+108*T+4*Ms+12*Mr
1e4	Initial, Event defined Message with ACTION=ACTIVATETASK defined		2776+12*T+4*Ms+18*Mr	130+108*T+4*Ms+16*Mr

Table C.2 OS Memory Requirements for GreenHills, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1b0	Initial	**BCC1**	972+5*T	21+1*T
1b1	Initial, STATUS=EXTENTED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS=TRUE		2588+8*T	126+96*T
1b2	Initial, Alarm with ACTION=ACTIVATETASK defined		1962+5*T+8*C+4*A	21+1*T+12*C+20*A
1b3	Initial, Message with ACTION=NONE defined		1334+5*T+4*Ms+14*Mr	22+1*T+4*Ms+12*Mr
1b4	Initial, Message with ACTION=ACTIVATETASK defined		1414+5*T+4*Ms+18*Mr	22+1*T+4*Ms+16*Mr
1b5	Initial, Resource for task defined		1828+8*T+1*R	125+104*T+12*R
1b6	Changed: ISR category 2 defined		150+104*T+16*I+4*IPL+12*R	4240+8*T+12*I+1*R
1b7	Changed: Resource referenced by ISR		152+104*T+16*I+4*IPL+16*R	4548+8*T+12*I+1*R

Table C.2 OS Memory Requirements for GreenHills, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1e0	Initial, Event defined	**ECC1**	129+108*T	2308+12*T
1e1	Changed: STATUS=EXTENDED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS= TRUE		138+108*T	3776+12*T
1e2	Initial, Event defined Alarm with ACTION=ACTIVATETASK defined		3034+12*T+8*C+12*A	129+108*T+8*C+28*A
1e3	Initial, Event defined Message with ACTION=NONE defined		2302+12*T+4*Ms+14*Mr	130+108*T+4*Ms+12*Mr
1e4	Initial, Event defined Message with ACTION=ACTIVATETASK defined		2404+12*T+4*Ms+18*Mr	130+108*T+4*Ms+16*Mr

Table C.3 OS Memory Requirements for WindRiver, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1b0	Initial	**BCC1**	1132+5*T	21+1*T
1b1	Initial, STATUS=EXTENTED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS=TRUE		2616+8*T	126+96*T
1b2	Initial, Alarm with ACTION=ACTIVATETASK defined		2104+5*T+8*C+4*A	21+1*T+12*C+20*A
1b3	Initial, Message with ACTION=NONE defined		1612+5*T+4*Ms+14*Mr	22+1*T+4*Ms+12*Mr
1b4	Initial, Message with ACTION=ACTIVATETASK defined		1680+5*T+4*Ms+18*Mr	22+1*T+4*Ms+16*Mr
1b5	Initial, Resource for task defined		1940+8*T+1*R	125+104*T+12*R
1b6	Changed: ISR category 2 defined		150+104*T+16*I+4*IPL+12*R	3982+8*T+12*I+1*R
1b7	Changed: Resource referenced by ISR		152+104*T+16*I+4*IPL+16*R	4340+8*T+12*I+1*R

Table C.3 OS Memory Requirements for WindRiver, SC1

Number	System Properties (configuration)	CC	ROM	RAM
1e0	Initial, Event defined	**ECC1**	129+108*T	1940+12*T
1e1	Changed: STATUS=EXTENDED ERRORHOOK=TRUE USEGETSERVICEID=TRUE USEPARAMETERACCESS=TRUE		138+108*T	3520+12*T
1e2	Initial, Event defined Alarm with ACTION=ACTIVATETASK defined		3140+12*T+8*C+12*A	129+108*T+8*C+28*A
1e3	Initial, Event defined Message with ACTION=NONE defined		2488+12*T+4*Ms+14*Mr	130+108*T+4*Ms+12*Mr
1e4	Initial, Event defined Message with ACTION=ACTIVATETASK defined		2580+12*T+4*Ms+18*Mr	130+108*T+4*Ms+16*Mr

Data Sheets for SC2 Scalability Class

The same initial system properties as for SC1 with addition of execution budget definition for Tasks and ISRs are used.

Table C.4 OS Memory Requirements for CodeWarrior, SC2

Number	System Properties (configuration)	CC	ROM	RAM
2e0	Initial, Event defined	**ECC1**	7432+20*T+16*I	330+128*T+24*I+88*IPL
2e1	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = undefined RESOURCELOCKTIME defined For ISR: Resource defined. RESOURCELOCKTIME defined			
2e2	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = 0 RESOURCELOCKTIME defined For ISR: Resource defined, RESOURCELOCKTIME defined		8876+20*T+20*I+1*R+8*Lr	348+124*T+32*I+88*IPL+ 24*R+8*Lr
2e3	Initial, For Tasks: EXECUTIONBUDGET, TIMEFRAME, TIMELIMIT undefined Interrupt LOCKINGTIME defined For ISR: Interrupt LOCKINGTIME defined		6408+16*T+16*I	294+108*T+20*I+88*IPL

Table C.5 OS Memory Requirements for GreenHills, SC2

Number	System Properties (configuration)	CC	ROM	RAM
2e0	Initial, Event defined	**ECC1**	6512+20*T+16*I	330+128*T+24*I+88*IPL
2e1	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = undefined RESOURCELOCKTIME defined For ISR: Resource defined. RESOURCELOCKTIME defined		5050+12*T+16*I	328+104*T+32*I+88*IPL
2e2	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = 0 RESOURCELOCKTIME defined For ISR: Resource defined, RESOURCELOCKTIME defined		8124+20*T+20*I+1*R+8*Lr	348+124*T+32*I+88*IPL+ 24*R+8*Lr
2e3	Initial, For Tasks: EXECUTIONBUDGET, TIMEFRAME, TIMELIMIT undefined Interrupt LOCKINGTIME defined For ISR: Interrupt LOCKINGTIME defined		5296+16*T+16*I	294+108*T+20*I+88*IPL

Table C.6 OS Memory Requirements for WindRiver, SC2

Number	System Properties (configuration)	CC	ROM	RAM
2e0	Initial, Event defined	**ECC1**	6780+20*T+16*I	330+128*T+24*I+88*IPL
2e1	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = undefined RESOURCELOCKTIME defined For ISR: Resource defined. RESOURCELOCKTIME defined		5176+12*T+16*I	328+104*T+32*I+88*IPL
2e2	Initial, Event defined For Tasks: Resource defined EXECUTIONBUDGET = 0 RESOURCELOCKTIME defined For ISR: Resource defined, RESOURCELOCKTIME defined		8100+20*T+20*I+1*R+8*Lr	348+124*T+32*I+88*IPL+ 24*R+8*Lr
2e3	Initial, For Tasks: EXECUTIONBUDGET, TIMEFRAME, TIMELIMIT undefined Interrupt LOCKINGTIME defined For ISR: Interrupt LOCKINGTIME defined		5416+16*T+16*I	294+108*T+20*I+88*IPL

Data Sheets for SC3 Scalability Class

The same initial system properties as for SC1 are used, but the STATUS is set to EXTENDED and OS-Applications are defined - one Trusted and one Non-Trusted.

Table C.7 OS Memory Requirements for CodeWarrior, SC3

Number	System Properties (configuration)	CC	ROM	RAM
3e0	Initial, Event defined	**ECC1**	5805+24*T+4*App+12*AppN	651+116*T+4*App+12*AppN
3e1	Initial, Event defined For tasks 4: Resource defined		6837+24*T+8*R+4*App+12*AppN	663+124*T+20*R+4*App+12*AppN
3e2	Initial, Event defined Message with ACTION=NONE defined		6877+24*T+26*Mr +12Ms+4*App+12*AppN	656+116*T+20*Mr+12*Ms+4*App+12*AppN
3e3	Initial, Event defined ISR with resource defined		11449+24*T+24*I+8*R+4*App+12*AppN	698+124*T+32*I+88*IPL+24*R+4*App+12*AppN
3e4	Initial, Event defined Counter defined Alarm defined		8477+24*T+20*C+12*A+4*App+12*AppN	655+116*T+16*C+28*A+4*App+12*AppN

Table C.8 OS Memory Requirements for GreenHills, SC3

Number	System Properties (configuration)	CC	ROM	RAM
3e0	Initial, Event defined	**ECC1**	5253+24*T+4*App+12*AppN	651+116*T+4*App+12*AppN
3e1	Initial, Event defined For tasks 4: Resource defined		6413+24*T+8*R+4*App+12*AppN	663+124*T+20*R+4*App+12*AppN
3e2	Initial, Event defined Message with ACTION=NONE defined		6433+24*T+26*Mr +12Ms+4*App+12*AppN	656+116*T+20*Mr+12*Ms+ 4*App+12*AppN
3e3	Initial, Event defined ISR with resource defined		10853+24*T+24*I+8*R+4*App+ 12*AppN	698+124*T+32*I+88*IPL+24*R+4*App+12*AppN
3e4	Initial, Event defined Counter defined Alarm defined		7919+24*T+20*C+12*A+4*App +12*AppN	655+116*T+16*C+28*A+4* App+12*AppN

Table C.9 OS Memory Requirements for WindRiver, SC3

Number	System Properties (configuration)	CC	ROM	RAM
3e0	Initial, Event defined	**ECC1**	5456+24*T+4*App+12*AppN	651+116*T+4*App+12*AppN
3e1	Initial, Event defined For tasks 4: Resource defined		5553+24*T+8*R+4*App+12*AppN	663+124*T+20*R+4*App+12*AppN
3e2	Initial, Event defined Message with ACTION=NONE defined		6589+24*T+26*Mr +12Ms+4*App+12*AppN	656+116*T+20*Mr+12*Ms+4*App+12*AppN
3e3	Initial, Event defined ISR with resource defined		10793+24*T+24*I+8*R+4*App+ 12*AppN	698+124*T+32*I+88*IPL+24*R+4*App+12*AppN
3e4	Initial, Event defined Counter defined Alarm defined		8053+24*T+20*C+12*A+4*App +12*AppN	655+116*T+16*C+28*A+4*App+12*AppN

Data Sheets for SC4 Scalability Class

The same initial system properties as for SC3 with addition of execution budget definition for Tasks and ISRs are used.

Table C.10 OS Memory Requirements for CodeWarrior, SC4

Number	System Properties (configuration)	CC	ROM	RAM
4e0	Initial, Event defined Resource for task and ISR defined Message with ACTION=none defined Counter defined Alarm defined EXECUTIONBUDGET for task and ISR defined RESOURCELOCKTIME for task and ISR defined INTERRUPTLOCKTIME for task and ISR defined	**ECC1**	21385+40*T+44*I+12*A+20*C+8*R+12*Ms+26*Mr+8*Lr+4*App+12*AppN	1242+164*T+72*I+88*IPL+16*C+28*A+32*R+8*Lr+12*Ms+20*Mr+4*App+12*AppN

Table C.11 OS Memory Requirements for GreenHills, SC4

Number	System Properties (configuration)	CC	ROM	RAM
4e0	Initial, Event defined Resource for task and ISR defined Message with ACTION=none defined Counter defined Alarm defined EXECUTIONBUDGET for task and ISR defined RESOURCELOCKTIME for task and ISR defined INTERRUPTLOCKTIME for task and ISR defined	**ECC1**	20467+40*T+44*I+12*A+20*C+8*R+12*Ms+26*Mr+8*Lr+4*App+12*AppN	1242+164*T+72*I+88*IPL+16*C+28*A+32*R+8*Lr+12*Ms+20*Mr+4*App+12*AppN

Table C.12 OS Memory Requirements for WindRiver, SC4

Number	System Properties (configuration)	CC	ROM	RAM
4e0	Initial, Event defined Resource for task and ISR defined Message with ACTION=none defined Counter defined Alarm defined EXECUTIONBUDGET for task and ISR defined RESOURCELOCKTIME for task and ISR defined INTERRUPTLOCKTIME for task and ISR defined	**ECC1**	19625+40*T+44*I+12*A+20*C+8*R+12*Ms+26*Mr+8*Lr+4*App+12*AppN	1242+164*T+72*I+88*IPL+16*C+28*A+32*R+8*Lr+12*Ms+20*Mr+4*App+12*AppN

D

System Generation Error Messages

This appendix explains AUTOSAR OS System Generator error messages.

The System Generator checks the compatibility of properties, parameters and limits and reports about possible errors via error messages. The error messages can be associated with the wrong syntax, mistakes in the implementation definition, wrong definitions of the application objects.

This appendix consists of the following sections:

Severity Level

The messages vary in their severity level, they can be one of the following types: ***information***, ***warning***, ***error***, ***fatal error***. Usually an information message attends other type of error message and contains reference to necessary information associated with error situation. A warning message only prevents about possible error. If an error message is detected, than the operation that should be started after the current one, is will not be executed. For example, if the error messages were found in project verification, the configuration file will not be generated however project settings check will be continued. When a fatal error message is found, than anyone of build command is terminated.

Error Message Format

The error message format depends on mode in which SysGen has been running. By default an error message includes the file name, the line number, the error code and a short error description. The error messages have one of the following formats in accordance with the severity level of message:

```
[<filename>(<line_number>) : ]information ####:<message>
[<filename>(<line_number>) : ]warning ####:<message>
[<filename>(<line_number>) : ]error ####:<message>
[<filename>(<line_number>) : ]Fatal Error ####:<message>
```

where:

`<filename>` - file name

`<line_number>` - line number

`####` - number of message

`<message>` - short description of the error

When the input file format is XML SysGen outputs the full XML path instead of [<filename>(<line_number>)] part of message:

List of Messages

0003: Option <option> is used, ignoring other options

Information

If command line option <option> is used, other options are ignored.

0006: EVENTs are not allowed for <class>

Error

According to OSEK OS spec. the events are not supported in Basic conformance classes. This message is generate if the EVENT object is defined, but one of the Basic conformance classes is defined for OS.

0007: EVENT is defined in <class>

Information

According to OSEK OS spec. the events are not supported in Basic conformance classes. This message is generate for each EVENT object defined when BCC is defined for OS.

0023: IsrStackSize shall be defined

Error

The *IsrStackSize* attribute shall be defined for SC2 and for ECC1 within SC1 if the application contains ISR category 2 or System Timer.

0038: At least one OS/TASK/APPMODE shall be defined

Error

At least one object of OS, TASK, and APPMODE types should be defined in an application. Only one OS object has to be defined.

0046: Same COUNTER cannot be used for SysTimer and SecondTimer

Error

The same counter cannot be attached to both system and second timers.

0051: Extended TASK is not supported in <class> class

Error

The OS conformance class is defined as one of the BCC classes, but extended task is defined in the application via EVENT object.

0056: ISR category 1 cannot have RESOURCE reference

Error

The RESOURCE reference cannot be defined for ISR object if its *CATEGORY* attribute is 1.

0057: ISR category 1 cannot use MESSAGE

Error

The message is generated if the *CATEGORY* attribute of ISR object is set to 1 and MESSAGE reference is defined.

0058: ISR cannot have reference to INTERNAL RESOURCE
Error

The message is generated if the *RESOURCE* attribute of the ISR object refers to the *RESOURCE* object with the *RESOURCEPROPERTY* attribute with **INTERNAL** value.

0065: Basic TASK cannot be notified by SETEVENT method
Error

If the referenced task has no events, then *ACTION* attribute of ALARM and MESSAGE objects cannot be defined as **SETEVENT**.

0066: TASK has no EVENT reference
Information

The referenced task is a basic one.

0072: EVENT <name> does not belong to TASK <name>
Warning

The task has no event, which is referenced by the EVENT reference of the ALARM object.

0073: More than one TASK per priority is defined for <class>
Error

Only one task per priority may be used in BCC1 and ECC1 classes, i.e., each task should have unique priority.

0086: Basic task doesn't require stack size
Warning

In SC1 the value of the *STACKSIZE* attribute is ignored for basic tasks.

0087: ISR category 1 doesn't require stack size
Warning

The value of the *STACKSIZE* attribute is ignored for ISRs of category 1.

0088: In SC1 and SC2 classes STACKSIZE value for ISR is ignored
Warning

In SC1 and SC2 the ISRs use common stack - “single stack” in BCC1 within SC1, common ISR stack otherwise.

0101: Name or ACCESSNAME shall be C-identifier

Error

Only C-identifier can be used in the *Name* or *ACCESSNAME* attributes' names.

0107: Identifier is longer than 32. It can cause compilation problem

Warning

If object identifier is longer than 32 characters, the result of compilation depends of compiler used. Make identifier shorter if there is an error during compilation of configuration or application source files.

0109: MAXALLOWEDVALUE of hardware counter is assumed to be <number>. It will be changed

Warning

Hardware counter has hardware defined maximal value for counter, so it cannot differ from assumed value. Counter information available via API will be changed to assumed value.

0116: One OS/APPMODE shall be defined

Error

One object of OS and at least one APPMODE types shall be defined in an application.

0120: USERESSCHEDULER is FALSE but RES_SCHEDULER is defined

Error

RES_SCHEDULER is supported if the value of the *USERESSCHEDULER* attribute is **TRUE**.

0123: TASK can have one INTERNAL RESOURCE

Error

The *TASK* object can has only one reference to the *RESOURCE* object which has the *RESOURCEPROPERTY* subattribute with the **INTERNAL** value.

0201: <Target Name> target is not supported by your license!

Fatal Error

The license for target platform is not provided.

0202: License <featurename> expires in <number> days

Warning

The message warns when the license expires in less than 30 days. The warning appears every work session.

0203: License <featurename> will expire tonight at midnight

Warning

This warning appears when the license expires at this day.

0252: Priority of ISR category 2 higher than ISR category 1

Error

The priority of any ISR category 2 shall be lower than priority of any ISR category 1.

0253: Priority of ISR category 1 less than ISR category 2

Error

The priority of any ISR category 2 shall be lower than priority of any ISR category 1.

0254: Priority of System/SecondTimer higher than ISR category 1

Error

The priority of any System or Second timer (considered as ISR category 2.) shall be lower than priority of any ISR category 1.

0264: <timer> Timer Channel <n> can not be defined for both timers

Error

The same hardware can be used by the System and Second timer only if different Channel attribute values are defined.

0500: ORTIFULL license not found. ORTI file will not be generated

Warning

ORTIFULL license shall be installed for ORTI files generation.

0501: EVENT masks don't arrange into 32 bits

Error

The number of supported event is 32 per task, but this error can be generated also in case if some events have AUTO mask but other events have user defined mask with several bits set.

0502: Event mask can not be equal to zero

Error

The number ‘0’ can not be used as a mask.

0507: <name> RESOURCE/MESSAGE/EVENT is declared but never used

Warning

The resource Ceiling priority is calculated automatically on the basis of information about priorities of tasks using the resource. If RESOURCE object is defines but not used, calculation of resource Ceiling priority is incorrect.

0508: RESOURCE is linked to itself, this link is ignored

Error

The RESOURCE can not be linked to itself.

0509: RESOURCE with property INTERNAL can not be linked

Error

This message is generated if a *RESOURCE* object with *RESOURCEPROPERTY* = **LINKED** refers through the *LINKEDRESOURCE* subattribute to a *RESOURCE* object with *RESOURCEPROPERTY* = **INTERNAL**.

0513: <value> must be defined as SysTimer

Error

The *SecondTimer* attribute can be SWCOUNTER if the *SysTimer* is equal to SWCOUNTER or HWCOUNTER. The *SecondTimer* can be HWCOUNTER if the *SysTimer* attribute is HWCOUNTER.

0515: SecondTimer hardware cannot be the same as SysTimer

Error

The *TimerHardware* attributes inside *SysTimer* and *SecondTimer* attributes along with *Timer* and *Channel* attributes identify a hardware used for System and Second Timers. This hardware shall be different for each Timer, i.e. each Timer shall have its own interrupt source.

0516: Prescaler attributes shall be both USER or OS

Error

The *Prescaler* attribute in the *SysTimer* attribute scope and the *Prescaler* attribute inside the *SecondTimer* attribute shall have the same value.

0750: EVENT <name> and EVENT <name> share the same bit and used by TASK all together

Warning

Sharing of the same bit by event masks inside one task may cause unexpected event notification and/or misinterpretation of the event.

0753: Period is out of representable range

Warning

The message is generated if the period size is more than 32 bits, the value of *OSTICKDURATION* attribute is truncated.

0755: The name of OS property file is not defined explicitly, <filename> will be used

Warning

Sysgen uses default file name, use the **-p** option to define location of OS property file.

0756: The name of OS header configuration file is not defined explicitly, <filename> will be used

Warning

Sysgen uses default file name, use the **-h** option to define the name of OS header file.

0757: The name of OS source configuration file is not defined explicitly, <filename> will be used

Warning

Sysgen uses default file name, use the **-c** option to define the name of OS source file.

0758: The name of ORTI file is not defined explicitly, <filename> will be used

Warning

Sysgen uses default file name, use the **-o** option is used to define the name of ORTI file.

0759: DEBUG_LEVEL equals 0. Option -o is ignored. ORTI file will not be generated

Warning

If the *DEBUG_LEVEL* attribute is set 0, then ORTI is not supported, and ORTI file is not generated.

0765: EVENT is treated as it is used by all the TASKs

Information

The EVENT object is not referred in any Task description. This means that event can be used by any task.

0766: EVENT is declared but never used and there are no Extended TASKs

Error

The EVENT object is defined in OIL file, but tasks have no references to this event. The task is considered as extended, if any event is referenced.

0775: Specified Period cannot be represented in hardware. Period <period value> is calculated

Warning

If the *Period* specified by user cannot be represented precisely in hardware, SysGen calculates closest period to it.

0800: Number of priorities exceeds <number>

Error

Number of defined Task objects exceeds OS capability.

0801: Number of RESOURCEs exceeds 2047

Error

Number of RESOURCE objects defined in system is restricted by 2047. (including RES_SCHEDULER).

0802: Number of ALARMs exceeds 2047

Error

Number of ALARM objects defined in system is restricted by 2047.

0803: Number of MESSAGEs exceeds 2047

Error

Number of MESSAGE objects defined in system is restricted by 2047.

0804: Number of COUNTERs exceeds 2047

Error

Number of COUNTER objects defined in system is restricted by 2047.

0805: ALARMTIME for autostarted alarm can not be more than MAXALLOWEDVALUE of COUNTER

Error

The value of the *AUTOSTART/ALARMTIME* attribute of the ALARM object shall be less than the value of the *MAXALLOWEDVALUE* attribute of the COUNTER object.

0806: CYCLETIME for autostarted alarm shall be more than MINCYCLE and less than MAXALLOWEDVALUE of COUNTER

Error

The value of the *AUTOSTART/CYCLETIME* attribute of the ALARM object shall be more than the value of the *MINCYCLE* attribute and less than the value of the *MAXALLOWEDVALUE* attribute of the COUNTER object.

0807: Number of APPMODEs exceeds 8

Error

Number of APPMODE objects defined in system is restricted by 8.

0808: APPMODE attribute shall be defined

Error

The *APPMODE* attribute shall be defined for TASK/ALARM with *AUTOSTART* = **TRUE** if there are more than one APPMODE defined in the application.

0809: SENDINGMESSAGE attribute shall refer to message with MESSAGEPROPERTY=SEND_STATIC_INTERNAL

Error

Only the message with *MESSAGEPROPERTY* = **SEND_STATIC_INTERNAL** can be assigned to *SENDINGMESSAGE* attribute of Receiving message.

0810: Different INITIALVALUE defined for SENDINGMESSAGE <name> receivers

Warning

The *UNQUEUED* receivers that has the same *SENDINGMESSAGE* value shares the same message object, so they shall have the same *INITIALVALUE*

0811: COM object shall be defined if at least one MESSAGE exists

Error

COM object shall be defined in order to use messages.

0812: The NOTIFICATION other than NONE can not be defined for SEND_STATIC_INTERNAL message

Error

Only message reception notification is supported for internal OSEK COM communications.

0813: The FLAGNAME <name> already defined

Error

The FLAGNAME names shall be unique.

0814: Message <messagename> with RECEIVE_QUEUED_INTERNAL property referenced by more than one TASK/ISR

Warning

The QUEUED message should have only one receiver.

0815: Reference to <messagename> MESSAGE

Information

This message is generated when the <messagename> is referenced from more than one object.

3000: Attribute not defined in the implementation: <name>

Error

This attribute is not specified in the implementation definition.

3001: Cannot open input file

Error

The file does not exist, or there are no permissions to open the file or directory.

3002: Target module for supported platform is not found

Error

Generation module is absent.

3003: OIL Reader error. <error type>

Error

Syntax or format error in the input file.

3004: Cannot open output file <filename>

Error

The output file cannot be opened if there is no enough free disk space or there is no appropriate permission to create or write the output file.

3005: Invalid command line option <option>

Error

The unknown command option found in command line.

3006: Option <option> requires argument

Error

The following options should be defined with an argument: **-c**, **-h**, **-i**, **-o**, **-p -m, -L, -M.**

3007: Only one input file allowed

Error

One and only one input file shall be specified on the command line.

3008: Input OIL file must be defined

Error

The input OIL file cannot be found if it does not exist or there are no permissions to open the file or directory.

3009: Type of object is not defined in the implementation

Error

Only standard objects defined by OIL specification shall be used in application definition. New object types are not allowed.

3010: Object is already defined with different type

Error

The given name has already been used for a system object of the other type.

3011: Attribute <name> is already defined

Error

The attribute with specified name has been already defined.

3012: Unknown reference type <name>

Error

This error may arise only if the OS implementation file is corrupted.

3013: Referenced object <type> <name> is not found

Error

The referenced object with specified name is not found.

3014: Attribute <name> must be defined

Error

The standard parameters and attributes with NO_DEFAULT specifier defined in implementation definition must be defined in the application definition (except references).

3015: <ObjectType> <Name> should belong to exactly one application

Error

Each OS object shall belong to exactly one OS-Application

3016: No application allowed in SC1 and SC2

Error

The OS-Applications shall be defined in SC3 or SC4 classes only.

3017: No timing protection allowed in SC1 and SC3

Warning

The Timing protection is used only in SC2 and SC4 classes.

3018: No protection hook allowed in SC1

Error

The Protection Hook can not be used in SC1.

3019: FastTerminate allowed for SC1 and BCC1 only, it is ignored

Warning

The attribute may be set only for BCC1 within SC1 class.

3021: Attribute value is out of range

Error

The attribute value does no fit in the allowed range.

3022: Attribute value does not match type

Error

The value does not correspond to attribute type.

3023: STACKSIZE shall be defined in <object>

Error

The *STACKSIZE* shall be defined for each ISR and Task object in the SC2..SC4 classes and for ISRs and Extended Tasks in SC1, ECC1.

3024: UnsatisfiedLinkError : <xxxx>

Error

Java Runtime environment error. Possibly FlexLM DLL (lmgr8c.dll) can not be found - check OS installations

3025: ALARMCALLBACK allowed in SC1 only

Error

ALARMCALLBACK can not be used in the Scalability classes with protection.

3026: <sc> requires timing protection attributes in TASKs or ISRs

Error

Timing protection attributes shall be defined for at least one Task or ISR in SC2 or SC4 class.

3027: <sc> requires TPTimer defined

Error

Time Protection Timer shall be defined for SC2, SC4.

3028: Attribute value is out of enum range

Error

The attribute value is wrong.

3040: Internal error: <description>

Fatal error

Syntax or format error in the input file: “unknown attribute type”, ‘no application’, “no implementation”.

3100: Timer <Name> not supported

Error

This error may occur only if the OS implementation file is corrupted.

3101: <attirbute> in all timers shall be equal

Error

GlobalPrescaler and *Prescaler* attributes shall be equal in all timers.

3102: Not enough hardware parameters to calculate period <parameter> for <object>

Error

The <parameter> shall be defined

3104: Period and all hardware timer parameters not allowed

Error

Timer Period shall have one definition: explicitly or via HW parameters.

3105: Timer parameters cannot be selected for given period

Error

The defined Period is too small or too big for this HW with given CPU clock

3107: TPTimer priority (max (isr2 priorities) + 1) too high

Error

In SC2 and SC4 classes the ISRs of category 2 shall have priority no more then 14.

3108: RESOURCELOCK cannot refer to internal resource

Error

Timing Protection can not be defined for *Internal Resource*.

3110: MAXRESOURCELOCKTIME cannot be 0

Error

Locking time can not be set to 0.

3111: Counter type mismatch

Error

HARDWARE counter is assigned to OS Timer defined as ‘SWCOUNTER’ or vise versa

3112: Counter cannot be HARDWARE if it is not used by hardware timer

Error

If the Counter type is set to HARDWARE then this counter shall be assigned to one of System Timers.

3115: Priority of ISR category 1 less than TPTimer priority

Error

All ISRs of category 1 should use the Priority higher then Time Protection Timer.

3116: Attribute value shouldn't be greater then scheduletable's length

Error

Any of numeric attributes of ScheduleTable can not be greater then its period.

3117: Referenced counter should be SOFTWARE

Error

Counter assigned into ‘*ACTION* = **INCREMENTCOUNTER**’ shall have the *TYPE* = **SOFTWARE**.

3118: Alarm is driven (directly or indirectly) with counter incremented by this alarm

Warning

Cycling of Counters via Alarms may lead to infinite loop.

3119: <object> <name> does not have access to task <name>

Error

<object> is ALARM, MESSAGE or SCHEDULETABLE. The object belongs to the application and refers to the task. The task shall belong to this application or have this application in its *ACCESSING_APPLICATION* list.

3122: TIMELIMIT cannot be zero when COUNLIMIT > 0

Error

When *COUNTLIMIT* is set to non-zero value the Arrival Rate protection is applied to ISR.

3125: Locking resource should be included into resource list

Error

The Resource referenced in *TIMING_PROTECTION* section shall be referenced in the object (Task or ISR) itself.

3128: Attribute STATUS cannot be STANDARD in SC3 and SC4

Error

Only EXTENDED status is allowed when memory protection is set.

3129: Number of TASKs exceeds 64

Error

Number of defined Task exceeds OS capability.

3130: Invalid link configuration file. System comment not found:

< expected comments printed here >

Error

Link configuration file supplied to SysGen shall have predefined structure with OS specific comments. This file is required as SysGen input for SC3, SC4 classes.

3131: No timing protection allowed in ISR category 1

Error

Timing Protection is not applicable for ISRs of category 1.

3135: ISR category 1 shall belong to a trusted application

Error

It is not allowed to have ISR of category 1 in nontrusted OS_Application.

3136: 'STATUS' and 'COMSTATUS' shall be equal

Warning

If this attributes have different values then SysGen assigns both attributes to EXTENDED value for simplicity.

3137: 'IsrStackSize' is defined but not used

Warning

Common stack for all ISRs of category 2 is used only in ECC1 within SC1.

3140: TPTimer is not allowed in SC1 and SC3

Warning

Time protection implies classes SC2 or SC4.

3141: Priority of ISR category 1 is equal to TPTimer priority

Warning

If the ISR of category 1 has the same priority as TP timer then its latency may significantly increase.

3142: <ISRname> can not be assigned to IRQ number <number>

Error

This error is generated if two ISRs (or OS Timer) are configured to use the same IRQ channel

3143: Timing protection is configured but timing protection parameters are not defined

Error

The TP timer is defined but there is no Task or ISR with none-zero protection parameters.

3144: MAXALLINTERRUPTLOCKTIME is ignored

Warning

Time protection can not be applied when ALL interrupts are disabled.

3145: Attribute 'DRIVER' should be OSINTERNAL

Error

Freescale AUTOSAR OS does not support GPT as driver for System Timers due to inconsistency in definition and usage of term “HARDWARE COUNTER” in AUTOSAR OS specification.

3146: Duplicate constant name
Error

The name of each TIMECONSTANT shall be unique.

3147: OFFSET value shall be less then scheduletable DURATION value
Error

The value of the *OFFSET* attribute shall be less than the value of the *DURATION* attribute of the SCHEDULETABLE object.

3148: The APPLICATION shall has at least one TASK or ISR object defined
Error

The OS does not supports OS-Applications without runnables.

3150: The timer <name> is used internally by the OS
Information

Used Timer names are displayed when switch -T is specified

3151: COMCALLBACK are not allowed in SC2, SC3 and SC4
Error

COMMCALLBACK is not allowed in classes with protection.

3152: Attribute Freeze in SystemTimer and SecondTimer shall be equal
Error

The Freeze attribute shall have the same value when both Timers use the same HW.

3155: The ISR with vector offset 201 is already used for STM timer
Error

If the value of the *IrqChannel* attribute is set to **EXTERNAL** and SystemTimer or SecondTimer is STM with Channel 1, then the value of the *IrqNumber* attribute can not be set to 201.

3156: EXECUTIONBUDGET should be more than <value> ns
Error

The value of the *TIMING_PROTECTION/EXECUTIONBUDGET* attribute is more than 0, but less than one TP Tick time for TASK.

3158: Each expiry point shall have a unique offset

Error

Any two *EXPIRYPOINTS* attributes belonging to the SCHEDULETABLE object shall have different OFFSET attribute values.

3159: Offset for initial expiry point shall be zero or be not less than MINCYCLE and less or equal to MAXALLOWEDVALUE

Error

The value of the *EXPIRYPOINTS*/*OFFSET* attribute of the SCHEDULETABLE object shall be zero or be not less than *MINCYCLE* attribute value and less or equal to *MAXALLOWEDVALUE* attribute value of the corresponding COUNTER object.

3160: The number of ticks between two adjacent expiry points shall be be not less than MINCYCLE and less or equal to MAXALLOWEDVALUE

Error

All the *EXPIRYPOINTS* attributes of a SCHEDULETABLE object may be ordered in accordance with their *EXPIRYPOINTS/OFFSET* attribute values. The number of ticks (X) between two adjacent expiry points of that ordered sequence is a difference of relevant *EXPIRYPOINTS/OFFSET* attribute values converted into timer ticks. X shall be not less than *MINCYCLE* attribute value and less or equal to *MAXALLOWEDVALUE* attribute value in corresponding COUNTER object.

3161: The value of final delay shall be not less than MINCYCLE and less or equal to MAXALLOWEDVALUE

Error

The value of the *EXPIRYPOINTS/OFFSET* attribute of the last ordered item in sequence (see 3160 error message description) shall be not less than the value of the *MINCYCLE* attribute and less or equal to the value fo the *MAXALLOWEDVALUE* attribute of the corresponding COUNTER object. This message may also indicate than the only the value fo the *EXPIRYPOINTS/OFFSET* attribute is equal to 0 of the SCHEDULETABLE object with *REPEATING* attribute set to TRUE.

3162: DURATION value shall be equal to MAXALLOWEDVALUE + 1 value of its associated counter

Error

The value of the *DURATION* attribute of the SCHEDULETABLE obejct (with *SYNCSTRATEGY* = **IMPLICIT**) shall be equal to the value (1 + *MAXALLOWEDVALUE*) of the corresponding COUNTER object.

3163: DURATION value shall be less or equal to MAXALLOWEDVALUE + 1 value of its associated counter

Error

The value of the *DURATION* attribute of the SCHEDULETABLE obejct shall be less or equal to the value (1 + *MAXALLOWEDVALUE*) of the corresponding COUNTER object.

3164: AUTOSTART may be TRUE only if AUTOSTART.TYPE is ABSOLUTE

Error

The value of the *AUTOSTART/TYPE* attribute shall be set to **RELATIVE** for SCHEDULETABLE with *SYNCSTRATEGY* = **IMPLICIT**.

3165: ADJUSTABLEEXPPOINT should be TRUE

Error

If SCHEDULETABLE has *SYNCSTRATEGY* = **EXPLICIT**, the *ADJUSTABLEEXPPOINT* attribute shall be set to **TRUE**.

3166: The value of (OFFSET - MAXRETARD) of an expiry point shall be greater than (OFFSET + MINCYCLE) of the previous expiry point

Error

Difference of the value *EXPIRYPOINTS/OFFSET* attribute and the value *ADJUSTABLEEXPPOINT/MAXRETARD* attribute shall be greater than (*OFFSET* + *MINCYCLE*), where *MINCYCLE* is the attribute of the corresponding COUNTER object.

3167: The value of (OFFSET + MAXADVANCE) of an expiry point shall be less than the DURATION of the schedule table

Error

Sum the value of *EXPIRYPOINTS/OFFSET* attribute and the value of *ADJUSTABLEEXPPOINT/MAXADVANCE* shall be less than the value of *DURATION* attribute of the SCHEDULETABLE object.

3168: The value of RELOFFSET shall be not equal to zero and shall be not less the (MAXALLOWEDVALUE - INITIAL OFFSET) value of its associated counter

Error

The value of the *AUTOSTART/RELOFFSET* attribute of the SCHEDULETABLE object shall be less than the difference of the value *MAXALLOWEDVALUE* attribute of the corresponding COUNTER object and the initial value of *EXPIRYPOINTS/OFFSET* attribute (the *OFFSET* attribute value of the first expirypoint in sequence) of the SCHEDULETABLE object .

3169: The value of ABSVALUE shall be not less the MAXALLOWEDVALUE value of its associated counter

Error

The value of the *AUTOSTART/ABSVALUE* attribute of the SCHEDULETABLE object shall be less than the value of the *MAXALLOWEDVALUE* attribute of the corresponding COUNTER object.

3170: Priority of ISR category 1 is equal to <ISRname> priority

Error

Priority of ISR category 1 shall be greater than priority of any ISR category 2 or System Timers.

3171: Timing protection is configured together with ISR category 1

Warning

ISRs category 1 should not be used in protected environment.

3172: Peripheral clock divider in all timers shall be equal

Error

Values of *PeripheralClockDivider* attribute shall be equal for the System and Second Timers

3173: Only one memory region is supported for platform <name>
Error

Only one of the attributes - *MemData0*, *MemData1* or *MemData2* - of the APPLICATION object may be set to **TRUE** in all the defined applications.

3176: RES_SCHEDULER or RESOURCE linked to RES_SCHEDULER are not allowed for ISR
Error

The ISR object can not have the *RESOURCE* reference equal to **RES_SCHEDULER**.

3177: ADJUSTABLEEXPPOINT is ignored as scheduletable <name> is implicit
Warning

If the value of the *SYNCSTRATEGY* attribute of the SCHEDULETABLE object is **IMPLICIT**, the value of the *ADJUSTABLEEXPPOINT* attribute is ignored (shall be set to **FALSE**).

3178: Timing protection shall be applied for <object> belonging to non-trusted application
Error

The value of the *TIMING_PROTECTION* attribute shall be set to **TRUE** for <object> belonging to non-trusted application in SC4 class.

3179: Task <name> activation is configured twice at the expiry point
Warning

The message is generated when the same task assigned into more than one '*ACTION* = **TASKACTIVATION**' inside one entry point of the SCHEDULETABLE object.

3180: Non-preemptive task <name> can not have internal resource <name>
Error

The message is generated if the *SCHEDULE* attribute of the task object is set to **NON** and the *RESOURCE* attribute refers to the RESOURCE object with the *RESOURCEPROPERTY* attribute with **INTERNAL** value.

Index

Q

R

S

T

U

V

W