© Copyright. 2004. Metrowerks Corp. ALL RIGHTS RESERVED.

Metrowerks reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Metrowerks does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Metrowerks products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Metrowerks product could create a situation where personal injury or death may occur. Should Buyer purchase or use Metrowerks products for any such unintended or unauthorized application, Buyer shall indemnify and hold Metrowerks and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Metrowerks was negligent regarding the design or manufacture of the part.

Legal Notices

The information in this document has been carefully checked and is believed to be entirely reliable, however, no responsibility is assumed for inaccuracies. Furthermore, Metrowerks reserves the right to make changes to any products herein to improve reliability, function or design. Metrowerks does not assume liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

No part of this publication may be reproduced or transmitted in any form or by any means - graphic, electronic, electrical, mechanical, chemical, including photocopying, recording in any medium, taping, by any computer or information storage retrieval systems, etc., without prior permissions in writing from Metrowerks.

Permission is granted to reproduce and transmit the Problem Report Form, the Customer Satisfaction Survey, and the Registration Form to Metrowerks.

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Metrowerks assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Metrowerks further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. Metrowerks disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

Trademarks

OSEK/VDX is a registered trademark of Siemens AG.
OSEKt uro is a trademark, Metrowerks, the Metrowerks logo and CodeWarrior are registered trademarks of Metrowerks.
Motorola and ™ are registered trademarks of Motorola, Inc.
Microsoft, MS-DOS and Windows are trademarks of Microsoft.
All other marks are trademarks or registered trademarks of their respective owners.

For More Information: www.freescale.com
Contents

1 Introduction .............................................................. 13
   OSEK OS Overview ................................................... 15
   Typographical Conventions ......................................... 17
   References ............................................................. 17
   Definitions, Acronyms and Abbreviations ................. 18
   Technical Support Information .................................. 19

2 Operating System Architecture ...................................... 21
   Processing Levels ..................................................... 21
   Conformance Classes ................................................ 22
   OSEK OS Overall Architecture .................................. 25
   Application Program Interface .................................. 26

3 Task Management ......................................................... 29
   Task Concept ............................................................ 29
      Extended Tasks ...................................................... 30
      Basic Tasks ........................................................ 32
   Task Priorities ........................................................ 33
   Tasks Stacks ............................................................. 34
      Stack Allocation ................................................... 34
      Single Stack ......................................................... 34
   Programming Issues .................................................. 35
      Configuration Options ........................................... 35
      Data Types .......................................................... 35
      Task Definition ..................................................... 35
      Run-time Services ................................................ 37
      Constants ............................................................ 38
      Conventions ........................................................ 38

4 Scheduler ...................................................................... 41
   General ................................................................. 41
   Scheduling Policy ...................................................... 42
      Non-preemptive Scheduling ...................................... 42
      Full-preemptive Scheduling .................................... 43
      Mixed-preemptive Scheduling .................................. 44
      Groups of Tasks ..................................................... 45
   Programming Issues .................................................... 45
      Configuration Options ............................................ 45

For More Information: www.freescale.com
Contents

Run-time Services ................................. 46

5 Interrupt Processing ............................ 47
   General ........................................ 47
   ISR Categories ................................ 48
      ISR Category 1 ................................ 48
      ISR Category 2 ................................ 49
   Interrupt Level Manipulation ............... 50
   ISR Stack ....................................... 50
   Interrupt Dispatcher ......................... 51
   Programming Issues ......................... 52
      Configuration Options .................... 52
   Run-time Services ......................... 52
   Conventions .................................. 53
   ISR Definition ................................ 54
   Interrupt Vector Definition ............... 55

6 Resource Management ......................... 57
   General ........................................ 57
   Access to Resources ......................... 58
      Restrictions when using resources .... 59
      Priority Ceiling Protocol ............... 59
      Scheduler as a Resource ............... 61
   Internal resources ......................... 61
   Programming Issues ......................... 62
      Configuration Option .................... 62
      Data Types ................................ 63
      Run-time Services ....................... 63
      Resource Definition ..................... 63

7 Counters and Alarms ............................ 65
   General ........................................ 65
   Counters ...................................... 66
   Alarms ......................................... 68
   Alarm Callback ............................... 70
   TimeScale ...................................... 71
   Programming Issues ......................... 71
      Configuration Options .................... 71

For More Information: www.freescale.com
Contents

Data Types ..................................................... 72
Counters and Alarm Generation .......................... 73
Run-time Services ........................................... 75
Constants ......................................................... 75

8 Events .......................................................... 77
General ............................................................. 77
Events and Scheduling ....................................... 78
Programming Issues .......................................... 79
  Configuration Options ................................... 79
Data Types ....................................................... 80
Events Definition ............................................. 80
Run-time Services ............................................. 81

9 Communication ................................................ 83
Message Concept ............................................... 83
Unqueued Messages .......................................... 84
Queue Messages ................................................ 86
Data Consistency .............................................. 87
Programming Issues .......................................... 87
  Configuration Options ................................... 87
Identifiers ....................................................... 87
Message Definition .......................................... 88
Run-time Services ............................................. 88
Callback Function ............................................ 89
Usage of Messages ............................................ 90

10 Error Handling and Special Routines .................. 91
General ............................................................. 91
Hook Routines ................................................... 91
Error Handling .................................................. 96
  Error Interface ............................................ 96
    Macros for ErrorHook .................................. 97
  Extended Status .......................................... 98
  Possible Error Reasons .................................. 98
Start-up Routine ............................................. 99
Application Modes ......................................... 99
System Shutdown ............................................. 100

For More Information: www.freescale.com
Contents

Programming Issues ................................................. 100
Configuration Options ............................................ 100

11 System Configuration .......................................... 103
General ............................................................. 103
Application Configuration File ................................. 104
OIL Concept ......................................................... 104
OIL File ............................................................. 104
OIL Format .......................................................... 105
Implementation Definition .......................................... 105
  Implementation Definition Grammar ......................... 105
Application Definition .............................................. 108
  Object Definition ................................................ 108
  Include Directive ............................................... 109
  Comments ......................................................... 109
  File Structure ................................................... 109
Configuration File Rules ........................................... 110

12 System Objects Definition ..................................... 111
General ............................................................. 111
OS Definition ....................................................... 112
  Global System Attributes ..................................... 112
  CPU Related Attributes ...................................... 114
  Stack Related Attributes .................................... 117
  Task Related Attributes ..................................... 118
  Interrupt Related Properties ................................ 119
  Hook Routines Related Attributes .......................... 120
  Floating Point Related Attributes .......................... 122
Task Definition ..................................................... 122
  Attributes ....................................................... 122
ISR Definition ...................................................... 124
  Attributes ....................................................... 124
Resource Definition ............................................... 125
Event Definition .................................................... 126
  Attribute ....................................................... 126
Counter Definition ................................................ 127
  Attributes ....................................................... 127

For More Information: www.freescale.com
Contents

Alarm Definition .............................................................. 128
Attributes ................................................................. 128
Message Definition ........................................................ 130
Attributes ................................................................. 130
Application Modes Definition ........................................... 131
COM Definition ............................................................ 132
Attributes ................................................................. 132
NM Definition ............................................................... 132
OSEKturbo Performance Dependency .................................... 133

13 Building of Application .................................................. 135
Application Structure ....................................................... 135
Action Sequence to Build Application ................................ 136
Application Configuration ................................................ 137
Source Files ............................................................... 139
Compiling and Linking ..................................................... 142
OS Object Files Dependency .............................................. 142
Sample Application ........................................................ 144

14 PowerPC Platform-Specific Features .................................. 145
Compiler-Specific Features ................................................ 145
Used Library Functions ..................................................... 145
Compiler Issues ............................................................ 145
Options for Metrowerks Software ....................................... 146
Link OS Code into Special Segment .................................... 147
Linker Command File for Metrowerks Linker for Embedded
PowerPC .......................................................... 147
General and Special Purpose Registers Usage ....................... 148
Stack Size ................................................................. 149
Stack Usage for OS Services ............................................. 149
Features ................................................................. 151
Programming Model ....................................................... 151
Target Hardware Initialization ......................................... 151
Interrupt Vector Table ..................................................... 151
Low-Power Mode .......................................................... 152
MSR Bits Manipulation .................................................... 152
Implementation Background .............................................. 153
Using Floating Point ....................................................... 153

For More Information: www.freescale.com
Contents

Nested Interrupts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153
Interrupt Dispatcher . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153
Stack Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
Timer Hardware . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
;DEC-based System Timer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
The ISRPRIORITY attribute specifies the periodic interrupt request priority. This attribute shall have the value from thr range 1..15.. 155
FIT-based System(Second) Timer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
Example of Timers Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 156

15 Application Troubleshooting 157
System Generation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157
Stack Errors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157
Unregistered External Interrupts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
Known Problems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
Troubleshooting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158

16 System Services 161
General . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161
Task Management Services . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 162
Data Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 162
Constants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 163
Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 163
Task Declaration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164
ActivateTask . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164
TerminateTask . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165
ChainTask . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165
Schedule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166
GetTaskId . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
GetTaskState . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 168
Examples for Task Management Services . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 169
ISR Management Services . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171
Data Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171
Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171
ISR Declaration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172
EnableAllInterrupts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172
DisableAllInterrupts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
ResumeAllInterrupts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuspendAllInterrupts</td>
<td>174</td>
</tr>
<tr>
<td>ResumeOSInterrupts</td>
<td>175</td>
</tr>
<tr>
<td>SuspendOSInterrupts</td>
<td>176</td>
</tr>
<tr>
<td>Examples for Interrupt Management Services</td>
<td>177</td>
</tr>
<tr>
<td>Resource Management Services</td>
<td>179</td>
</tr>
<tr>
<td>Data Types</td>
<td>179</td>
</tr>
<tr>
<td>Constants</td>
<td>179</td>
</tr>
<tr>
<td>Resource Declaration</td>
<td>179</td>
</tr>
<tr>
<td>GetResource</td>
<td>180</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>181</td>
</tr>
<tr>
<td>Examples of Using Resources</td>
<td>182</td>
</tr>
<tr>
<td>Event Management Services</td>
<td>184</td>
</tr>
<tr>
<td>Data Types</td>
<td>184</td>
</tr>
<tr>
<td>Event Declaration</td>
<td>184</td>
</tr>
<tr>
<td>SetEvent</td>
<td>185</td>
</tr>
<tr>
<td>ClearEvent</td>
<td>186</td>
</tr>
<tr>
<td>GetEvent</td>
<td>186</td>
</tr>
<tr>
<td>WaitEvent</td>
<td>187</td>
</tr>
<tr>
<td>Examples of Using Events</td>
<td>188</td>
</tr>
<tr>
<td>Counter Management Services</td>
<td>191</td>
</tr>
<tr>
<td>Data Types and Identifiers</td>
<td>191</td>
</tr>
<tr>
<td>Constants</td>
<td>192</td>
</tr>
<tr>
<td>Counter Declaration</td>
<td>193</td>
</tr>
<tr>
<td>InitCounter</td>
<td>194</td>
</tr>
<tr>
<td>CounterTrigger</td>
<td>194</td>
</tr>
<tr>
<td>GetCounterValue</td>
<td>195</td>
</tr>
<tr>
<td>GetCounterInfo</td>
<td>196</td>
</tr>
<tr>
<td>Examples for Counter Management</td>
<td>197</td>
</tr>
<tr>
<td>Alarm Management Services</td>
<td>198</td>
</tr>
<tr>
<td>Data Types and Identifiers</td>
<td>198</td>
</tr>
<tr>
<td>Constants</td>
<td>199</td>
</tr>
<tr>
<td>Alarm Declaration</td>
<td>199</td>
</tr>
<tr>
<td>GetAlarmBase</td>
<td>199</td>
</tr>
<tr>
<td>GetAlarm</td>
<td>200</td>
</tr>
<tr>
<td>SetRelAlarm</td>
<td>201</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td>202</td>
</tr>
<tr>
<td>CancelAlarm</td>
<td>203</td>
</tr>
</tbody>
</table>
# Contents

### 17 Debugging Application 229
- General .......................... 229
  - Extended Status .................. 229
- ORTI .................................. 229
  - Stack Debugging Support ........ 232
- Using OS Extended Status for Debugging .................. 232
- Context Switch Monitoring .................. 232
- ORTI Features .......................... 233
  - ORTI Trace Interfaces ............ 233
  - ORTI Breakpoint Interface ........ 235
- Stack Debugging Support .................. 240
  - Stack labels .................. 240
  - Stack Overflow Checking ........ 240

### A Sample Application 241
- Description .......................... 241
- Source Files .................. 243

### B System Service Timing 245
- General Notes .......................... 245
- Data Sheet .......................... 246

### C Memory Requirements 253
- OSEK Operating System Memory Usage .................. 253
- Data Sheet for CodeWarrior .................. 255

### D System Generation Error Messages 259
- Severity Level .................. 259
- Error Message Format .................. 260
- List of Messages .................. 261
  - SysGen Engine Messages ........ 261
  - OIL Reader Messages ........ 264
  - Target-Specific DLL Messages .... 277

### Index 293
Introduction

This Technical Reference describes OSEKturbo OS/MPC55xx, the version of the OSEK Operating System (OSEK OS) for PowerPC providing high speed performance and low RAM usage. All the system mechanisms, particularities, services and programming techniques are described in detail with numerous examples. Data for performance characteristics and memory requirements are provided.

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

“The Task Management” chapter explains the task concept in OSEK and all other questions related to tasks.

“The Scheduler” chapter provides a description of scheduling policies in OSEK OS.

“The Interrupt Processing” chapter highlights the OSEK approach to interrupt handling.

“The Resource Management” chapter describes resource management and task coordination by resources.

“The Counters and Alarms” chapter describes usage of these control mechanisms in OSEK OS.

“The Events” chapter is devoted to event management and task coordination by events.

1. The term OSEK means ‘Open systems and the corresponding interfaces for automotive electronics’ (in German). A real-time operating system, software interfaces and functions for communication and network management tasks are thus jointly specified within the OSEK standard.
“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 OSEK 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 OSEK OS. It also describes memory requirements.

“PowerPC Platform-Specific Features” chapter describes special OSEK 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 OSEK OS.

“System Services” chapter provides a detailed description for all OSEK 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 OSEK terms.

“Sample Application” appendix contains the text and listing of a sample customer application developed using the OSEK 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 various versions of the OSEK OS.
“System Generation Error Messages” appendix explains OSEK OS System Generator error messages.

The “Introduction” chapter consists of the following sections:

- OSEK OS Overview
- Typographical Conventions
- References
- Definitions, Acronyms and Abbreviations
- Technical Support Information

OSEK OS Overview

The OSEK Operating System is a real-time operating system which conforms to the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification.

The OSEK OS meets the following requirements:

- The OS is fully configured and statically scaled;
- The OS performance parameters are well known;
- The most part of the OS is written in strict correspondence with ANSI C standard, the OS and the application on its basis can be easily ported from one platform to another.

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

The OSEK 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.

For More Information: www.freescale.com
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
• 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.
• TimeScale enables periodic activations of tasks in accordance with a static defined schedule.

Error Treatment
• Mechanisms supporting the user in case of various errors.

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

---

1. The Communication part of the OSEK Operating System conforms to the OSEK/VDX Communication, v.2.2.2, 18 December 2000 specification.


3. TimeScale is an OSEKturbo extension of OSEK OS.
The OSEK Operating System is scaled in two ways: either by changing the set of system services or through the so-called Conformance Classes. They are available to meet different requirements concerning the OS functionality and capability. These Conformance Classes differ not only in the number of services they provide, but also in their capabilities and scalability. The classes are based on one another in upwardly compatible fashion. (see “Conformance Classes”)

The OSEK 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 OSEK names of directives, macros, constants, routines and variables.

**Courier font**

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

**References**


For More Information: www.freescale.com
The following acronyms and abbreviation are used in this Technical Reference.

API  Application Program Interface (a set of data types and functions)
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 never has 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
N/A  Not applicable
OIL  OSEK Implementation Language
ORTI OSEK Run Time Interface
OS   Operating System
OSEK turbo OS/MPC55xx

Technical Support Information

To order Motorola/Metrowerks products or literature, consult your local sales representative.

For technical support please use:

**US**
Phone (US): 800 377 5416  
Phone: +1 512 996 5300  
Fax: +1 512 996 5405  
Email: support@metrowerks.com

**Europe**
Phone: +41 61 69 07 505  
Fax: +41 61 69 07 501  
Email: support_europe@metrowerks.com


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
- Conformance Classes
- OSEK OS Overall Architecture
- Application Program Interface

Processing Levels

The OSEK 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 OSEK OS enables controlled real-time execution of several processes which virtually run in parallel.

The OSEK 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.

### 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 OSEK operating system.
- allowing partial implementations along pre-defined lines. These partial implementations may be certified as OSEK compliant.
- creating an upgrade path from the classes of less functionality to the classes of higher functionality with no changes to the application using OSEK 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 conditions.
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 OSEK turbo;
- 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 OSEK turbo does not support multiply activation and therefore it doesn’t have BCC2 and ECC2 classes.

The OSEK turbo 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 2.1 indicates the minimum resources to which an application may resort, determined for each Conformance Class in the OSEK OS.

**Table 2.1 OSEK OS Conformance Classes**

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

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

Maximal numbers of the OSEKturbo OS/MPC55xx system objects are indicated in **Table 2.2**.

**Table 2.2 OSEKturbo OS/MPC55xx Maximal System Resources**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of task’s priorities</td>
<td>64</td>
</tr>
<tr>
<td>Number of tasks which are not in suspended state</td>
<td>64</td>
</tr>
<tr>
<td>Number of events per task</td>
<td>32</td>
</tr>
<tr>
<td>Number of resources (including RES_SCHEDULER)</td>
<td>255</td>
</tr>
<tr>
<td>Number of Application Modes</td>
<td>8</td>
</tr>
<tr>
<td>Number of other OSEK objects (alarms, messages, counters)</td>
<td>254</td>
</tr>
</tbody>
</table>
OSEK OS Overall Architecture

The OSEK 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 OSEK 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;
- **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 timer** provides implementation-independent time management.

As it is shown in Table 2.1, 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 OSEK OS service mechanisms.

The OSEK Operating System is not scaled through the Conformance Classes only, but it also has various extensions which can be used in any Conformance Class. 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”.

For More Information: www.freescale.com
Since the OSEK 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 OSEK 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 OSEK 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 OSEK 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.

The OSEKturbo 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.
Task Management

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

This chapter consists of the following sections:

- Task Concept
- Task Priorities
- Tasks Stacks
- Programming Issues

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 OSEK OS:

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

The Basic Tasks release the processor only if:

- they are being terminated,
- the OSEK 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 OSEK 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.

Every running task is represented by its run-time context. This refers to the CPU registers and some compiler-dependent ‘pseudoregisters’ in RAM. 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 OSEK 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

<table>
<thead>
<tr>
<th>Transition</th>
<th>Former state</th>
<th>New state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activate</td>
<td>suspended</td>
<td>ready</td>
<td>A new task is entered into the ready list by a system service.</td>
</tr>
<tr>
<td>start</td>
<td>ready</td>
<td>running</td>
<td>A ready task selected by the scheduler is executed.</td>
</tr>
<tr>
<td>wait</td>
<td>running</td>
<td>waiting</td>
<td>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.</td>
</tr>
<tr>
<td>release</td>
<td>waiting</td>
<td>ready</td>
<td>Events have occurred which a task has been waiting for.</td>
</tr>
<tr>
<td>preempt</td>
<td>running</td>
<td>ready</td>
<td>The scheduler decides to start another task. The running task is put into ready state.</td>
</tr>
<tr>
<td>terminate</td>
<td>running</td>
<td>suspended</td>
<td>The running task causes its transition into suspended state by a system service.</td>
</tr>
</tbody>
</table>
Termination of tasks is possible only if the task terminates itself (‘self-termination’).

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

\section*{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 \textit{waiting} state.

- \textit{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.

- \textit{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.

- \textit{suspended}
  
  In suspended state the task is passive and does not occupy any resources, merely ROM.

\begin{table}[h]
\centering
\caption{States and Status Transitions for a Basic Task}
\begin{tabular}{|l|l|l|l|}
\hline
\textbf{Transition} & \textbf{Former state} & \textbf{New state} & \textbf{Description} \\
\hline
activate & suspended & ready & A new task is entered into the \textit{ready} list by a system service. \\
start & ready & running & A \textit{ready} task selected by the scheduler is executed. \\
preempt & running & ready & The scheduler decides to start another task. The \textit{running} task is put into \textit{ready} state. \\
terminate & running & suspended & The \textit{running} task causes its transition into \textit{suspended} state by a system service. \\
\hline
\end{tabular}
\end{table}
Task Priorities

The OSEK 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 OSEK 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”.

For More Information: www.freescale.com
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

Each extended task has its own statically allocated 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 crash.

Single Stack

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

In the ECC1 class all the basic tasks use the single stack. But the extended tasks have their own stacks. Interrupts have a separate stack as well.

The OSEKturbo OS uses the main application stack for a single stack. Its size is defined by the user at the link stage.

For More Information: www.freescale.com
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.

- **CC**
  Specifies the conformance class. If AUTO, the conformance class is defined according to the task definitions.

Data Types

The OSEK 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:

```c
TASK TASKSENDER {
    PRIORITY = 5;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    RESOURCE = MYRESOURCE;
    RESOURCE = SECONDRESOURCE;
    EVENT = MYEVENT;
    STACKSIZE = 256;
    ACCESSOR = SENT {
        MESSAGE = MYMESSAGE;
        WITHOUTCOPY = TRUE;
        ACCESSNAME = "MessageBuffer";
    };
};
```

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

### Table 3.3 Task Properties

<table>
<thead>
<tr>
<th>Object Parameters</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Attributes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>integer [0..0x7FFFFFFF]</td>
<td>Defines the task priority. The lowest priority has the value 0</td>
</tr>
<tr>
<td>SCHEDULE</td>
<td>FULL, NON</td>
<td>Defines the run-time behavior of the task</td>
</tr>
<tr>
<td>AUTOSTART</td>
<td>TRUE, FALSE</td>
<td>Defines whether the task is activated during the system start-up procedure or not</td>
</tr>
<tr>
<td>APPMODE</td>
<td>name of APPMODE</td>
<td>Defines the application mode in which the task is auto-started</td>
</tr>
<tr>
<td>ACTIVATION</td>
<td>1</td>
<td>Specifies the maximum number of queued activation requests for the task (OSEKturbo does not allow multiply activations)</td>
</tr>
</tbody>
</table>
The application definition file contains one such statement per task. The task generation statement is described in detail in “System Objects Definition”.

The constructional statement DeclareTask may be used for compatibility with the previous OSEK versions. It may be omitted in application code.

DeclareTask is as follows:

DeclareTask( <TaskName> );

Run-time Services

The OSEK 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>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateTask</td>
<td>Activates the task, i.e. moves it from suspended to ready state</td>
</tr>
<tr>
<td>TerminateTask</td>
<td>Terminates the running task, i.e. moves it from ready to suspended state</td>
</tr>
</tbody>
</table>

Table 3.3 Task Properties

<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>name of RESOURCE</th>
<th>Resources accessed by the task. There can be several resource references</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td>name of EVENT</td>
<td>Event owned by the task. There can be several event references</td>
</tr>
<tr>
<td>ACCESSOR</td>
<td>SENT, RECEIVED</td>
<td>Defines the type of usage of the message</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>name of MESSAGE</td>
<td>Specifies the message to be sent or received by the task</td>
</tr>
<tr>
<td>WITHOUTCOPY</td>
<td>TRUE, FALSE</td>
<td>Defines whether a local copy of the message is used or not</td>
</tr>
<tr>
<td>ACCESSNAME</td>
<td>string</td>
<td>Defines the reference which can be used by application to access the message data</td>
</tr>
</tbody>
</table>

OSEKturbo Specific Attribute

STACKSIZE integer Defines the size of the task stack in bytes (only for the Extended Tasks)

Table 3.4 Task Management Run-time Services

For More Information: www.freescale.com
Examples of using the run-time services are provided in “Examples for Task Management Services”.

**Constants**

The following constants are used within the OSEK 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 OSEK OS to indicate the task:

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

**Conventions**

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

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

The name of the task function will be generated from TaskName by macro TASK.
Scheduler

This chapter provides a description of scheduling policies in OSEK OS.

This chapter consists of the following sections:

- General
- Scheduling Policy
- Programming Issues

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. It is possible to call a special user’s hook from the scheduler idle loop. This property is turned on via the system configuration option IdleLoopHook. An instruction that puts the CPU in low power mode may be inserted into idle loop to reduce power consumption. This
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 \textit{SCHEDULE}. The valid values are – \textit{NON} and \textit{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:
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.

**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 of 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.

- **IdleLoopHook**
  If this option is turned on, then user supplied hook will be called from the scheduler idle loop.

- **HCLowPower**
  If this option is turned on, an instruction that puts the CPU in low power mode is used instead of the scheduler’s idle loop.

- **ResourceScheduler**
  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:

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

For More Information: www.freescale.com
Interrupt Processing

This chapter highlights OSEK approach to the interrupt handling.

This chapter consists of the following sections:

- General
- ISR Categories
- Interrupt Level Manipulation
- ISR Stack
- Interrupt Dispatcher

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. Addresses of ISRs should be pointed to in the vector table.

In OSEKturbo OS/MPC55xx in ECC1 class all ISRs of category 2 use the separate stack (ISR stack) which is used only by ISRs during their execution. The size of the ISR stack is defined by the user.

According to the OSEK/VDX Operating System, v.2.2, 10 September 2001 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.
NOTE
If application manipulates directly with the CPU registers that control interrupts then OS behavior is unpredictable.

ISR Categories

ISR Categories

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

ISR Category 1

ISRs of these type are executed on the current stack. 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 */
  ...
}

**WARNING!**  
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 OSEK Operating System provides an automatic switch to the ISR stack (for ECC1 only) and enters OSEK OS execution context. After that, any user’s routine can be executed, including allowed OS calls (to activate a task, send a message or trigger a counter). See “Run-time Services” for the list of services allowed for ISR. At the end of the ISR, the System automatically switches back to the task stack 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.
Interrupt Processing

Interrupt Level Manipulation

**WARNING!** Interrupts which can not be disabled, such as NMI or any synchronous exception, shall not be assigned to ISRs of category 2. If OS could 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(s) or SystemTimer configured, otherwise OSEKturbo OS dispatcher does not controls 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 frame of 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 frame of task or ISR where SuspendOSInterrupts is called.

Resources can be used to temporary disabling interrupts. If 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 rescheduling is disabled. Interrupts are reenabled and task rescheduling is reenabled after releasing the resource.

**ISR Stack**

The purpose of the ISR stack is to save memory. Since interrupts can occur while any task is executed and can be nested it means that
every task stack must be big enough to store several interrupt stack frames (in addition to task and ISR(s) needs for local variables, function calls, etc.). To avoid this overhead, the separate ISR stack is used in the OSEKturbo OS. Switching to this stack is performed by the OS at the beginning of ISR category 2. This stack is used only by ISRs of category 2. If nested interrupts occur after the stack has been switched, they will continue to use this stack. After completion of ISR category 2 OS switches back from ISR stack.

In BCC1 class ISRs use common single stack.

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.

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.

**Interrupt Dispatcher**

OSEKturbo OS/MPC55xx provides interrupt dispatcher for the external interrupt distinction. There is an InterruptDispatcher attribute which defines the type of OS interrupt dispatcher. It may have value of OneLevel or MultiLevel. In order to use Interrupt Dispatcher it is necessary for each external interrupt to define an ISR object in the OIL file and assign the EXTERNAL value to the IqrChannel attribute, set appropriated value for the IqrNumber attribute (For the correspondence between IqrNumber value and external interrupt sources see the Hardware Technical References) and assign the PRIORITY attribute value.

More details about the interrupt dispatcher see “Interrupt Related Properties” and “Interrupt Dispatcher”. For more detailed information about MPC interrupt controller see MPC User’s Manuals.
Programming Issues

Configuration Options

- **InterruptDispatcher**
  Specifies mode of interrupt dispatcher.

Run-time Services

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

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableAllInterrupts</td>
<td>Disable all interrupts, does not allow nesting</td>
</tr>
<tr>
<td>EnableAllInterrupts</td>
<td>Restore state of interrupts saved by DisableAllInterrupts service</td>
</tr>
<tr>
<td>SuspendAllInterrupts</td>
<td>Disable all interrupts, allows nesting</td>
</tr>
<tr>
<td>ResumeAllInterrupts</td>
<td>Restore state of interrupts saved by SuspendAllInterrupts service</td>
</tr>
<tr>
<td>SuspendOSInterrupts</td>
<td>Disable interrupts of category 2</td>
</tr>
<tr>
<td>ResumeOSInterrupts</td>
<td>Restore state of interrupts saved by SuspendOSInterrupts service</td>
</tr>
</tbody>
</table>

Not all OS services may be used inside ISRs. All OS services that are allowed for use in the ISRs of category 2 are listed in the Table 5.2.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateTask</td>
<td>Activates the specified task (puts it into the ready state)</td>
</tr>
<tr>
<td>GetTaskId</td>
<td>Gets reference to a task</td>
</tr>
<tr>
<td>GetTaskState</td>
<td>Gets state of the task</td>
</tr>
<tr>
<td>GetResource</td>
<td>Occupies a resource</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>Releases a resource</td>
</tr>
<tr>
<td>SetEvent</td>
<td>Sets event for the task</td>
</tr>
<tr>
<td>GetEvent</td>
<td>Gets event of the task</td>
</tr>
</tbody>
</table>

For More Information: www.freescale.com
Interrupt Processing

Programming Issues

Conventions

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

ISR( <ISRName> )
{
    ...
}

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

1. OSEK/VDX does not specifies using of keyword ISR for ISRs of category 1, it is OSEKturbo specific for this category.

---

Table 5.2  Services allowed for use in ISRs of category 2

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterTrigger(\textsuperscript{a})</td>
<td>Increments a counter value and process attached alarms</td>
</tr>
<tr>
<td>GetAlarmBase</td>
<td>Gets alarm base characteristics</td>
</tr>
<tr>
<td>GetAlarm</td>
<td>Gets value in ticks before the alarm expires</td>
</tr>
<tr>
<td>SetRelAlarm</td>
<td>Sets relative alarm</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td>Sets absolute alarm</td>
</tr>
<tr>
<td>CancelAlarm</td>
<td>Cancels alarm</td>
</tr>
<tr>
<td>SendMessage</td>
<td>Sends an unqueued message in WithCopy configuration to the specified task</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>Delivers data of an unqueued message in WithCopy configuration to the application message copy</td>
</tr>
<tr>
<td>GetRunningStackUsage(\textsuperscript{a})</td>
<td>Gets amount of stack used by running task</td>
</tr>
<tr>
<td>GetStackUsage(\textsuperscript{a})</td>
<td>Gets amount of stack used by certain task</td>
</tr>
<tr>
<td>GetTimeStamp(\textsuperscript{a})</td>
<td>Gets current value of system counter</td>
</tr>
</tbody>
</table>

\(\textsuperscript{a}\) This service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification.
The constructional statement *DeclareISR* declares ISR function. It may be useful in vector file if it does not include OS configuration file.

*DeclareISR* looks like the following:

```
DeclareISR( <name of ISR> );
```

The same name shall be used for corresponding ISR object definition (see “*ISR Definition*”) and for interrupt vector definition (see “*Interrupt Vector Table*”).

### ISR Definition

To define common ISR parameters like ISR stack size the corresponding OS properties should be specified in the configuration file.

Definition of Interrupt related properties looks like:

```plaintext
OS <name> {
    ....
    IsrStackSize = 256;
};
```

Definition of specific ISR object looks like:

```plaintext
ISR Handler {
    PRIORITY = 8;
    CATEGORY = 2;
    RESOURCE = ISRresource;
    ACCESSOR = RECEIVED {
        MESSAGE = messageA;
        ACCESSNAME = myBuffer;
    };
    IrqChannel = EXTERNAL {
        IrqNumber = 146; /* SCI_A combined interrupt request */
    };
};
```

See “*ISR Definition*” for details.
Interrupt Vector Definition

To provide starting ISR routine during interrupt at run-time the interrupt vector shall be defined and added to interrupt vector table (see “Interrupt Vector Table”).
Resource Management

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

This chapter consists of the following sections:

- General
- Access to Resources
- Programming Issues

General

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

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 OSEKturbo extension of OSEK OS, which fully supports resources only in BCC2 and ECC conformance classes.

For More Information: www.freescale.com
• 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 OSEK 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 OSEK 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 OSEK 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:

```c
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 OSEK 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 Management
Access to Resources

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 OSEK 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, Schedule and WaitEvent must not be called while a resource is occupied. The interrupt service routine must not be completed with the resource occupied.

OSEK 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 OSEK 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.

For More Information: www.freescale.com
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 and interrupt flags 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 OSEK 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 OSEK OS scheduler.

The example shown on Figure 6.1 illustrates the mechanism of the 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 OSEK 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) priority 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.

## 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 preemtable tasks - they can not preemt each other; while the tasks with the priority higher than the highest priority within the group preempt the tasks within the group.

The non preemtable 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 preemtable 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:

For More Information: www.freescale.com
Resource Scheduler
If this option is set to FALSE, RES_SCHEDULER is not supported by the OS.

Data Types
The OSEK 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 OSEK 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>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetResource</td>
<td>This call serves to occupy the resource (to enter critical section of the code, assigned to the resource)</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>Releases the resource assigned to the critical section (to leave the critical section)</td>
</tr>
</tbody>
</table>

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”.

The declaration statement `DeclareResource` may be used for compatibility with previous OSEK versions. It may be omitted in application code.

For More Information: www.freescale.com
DeclareResource is as follows:

DeclareResource( <ResourceName> );
Counters and Alarms

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

This chapter consists of the following sections:

- General
- Counters
- Alarms
- Alarm Callback
- TimeScale
- Programming Issues

General

The OSEK 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 OSEKturbo OS. Additionally OSEKturbo OS provides a TimeScale mechanism for fast tasks activations in accordance with statically defined schedule. The recurring occasions (sources) can be registered by counters. Based on counters, the OSEK OS offers an alarm mechanism to the application software. Counters and alarms are provided by the OSEK OS in all Conformance Classes. Counter concept and counter management system services are based on the OSEK Operating System, Concept v.1.00, September 1995 and OSEK Operating System, Application Program Interface, v.1.00, 11 September 1995 documents.

OSEKturbo OS provides two types of counters: software (SW) counters, that are the same as in previous versions of OSEK 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.
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.

A 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 CounterTrigger.

The HW counters may use only System and Second Timers and has a maximum allowed value 0xFFFFFFFF. Only DEC timer may be used with 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 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 this parameter can be treated as a counter-specific reference value.

The operating system provides the standard service `GetCounterInfo` to read these counter specific values. Also the service `GetCounterValue` is designed to read the current counter value.

OSEKturbo OS provides two timers (the internal system clocks): a system timer and a second timer. The timers are not defined in OSEK OS specifications starting from OSEK OS v.2.0 specification. This is OSEKturbo extension of OSEK 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 “PowerPC 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 OSEK 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 he/she can not change the provided code.
Counters and Alarms

Alarms

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 CounterTrigger system service. It is not allowed to use CounterTrigger in ISR category 1 (see section “ISR Categories”).

NOTE

CounterTrigger service is not defined starting from OSEK OS v.2.0 specifications, it is defined in the OSEK OS v.1.0 specification.

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 OSEK 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 OSEK 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 $T_2$. 

For More Information: www.freescale.com
Counters and Alarms

Alarm Callback

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 recommended to use values of cycle and/or increment parameters close to 0xFFFFFFFF (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 ALARM CALLBACKNAME attribute. The alarm callback is the usual user's function. It can have neither parameter(s) nor return value.

Only the SuspendAllInterrupts and ResumeAllInterrupts services may be used within alarm callback. No other OS services shall be called.

For More Information: www.freescale.com
The callback function shall have next definition:

```c
ALARM_CALLBACK(CallbackName)
{
    /* user application code */
}
```

**TimeScale**

OSEKturbo OS provides a special feature for fast tasks activations in accordance with statically defined schedule named *TimeScale*. User can define in a configuration file a sequence of tasks to be cyclically activated at predefined time points. *TimeScale* always uses the System Timer which must be defined as a *HWCOUNTER* in this case. Only one task may be activated at each *Step* of *TimeScale* but the *StepTime* of any *Step* may be set to ‘0’ to achieve a simultaneous activation of two or more tasks. *TimeScale* has the better performance than cyclic *Alarms* because of simplified algorithm and reduced system overhead. System Timer can not be used for *TimeScale* and *Alarms* simultaneously.

**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.
- *TimeScale*
  If this option is turned on the Time Scale is used.

---

1. *TimeScale* is OSEKturbo extension of OSEK OS.
Data Types

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

- **CtrRefType**
  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;
    - **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:

  ```c
  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.
NOTE

CtrRefType, CtrInfoType and CtrInfoRefType data types are not defined in OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

The following data type is established by OSEK 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:

```c
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:

```c
OS <name> {
    ...
    SysTimer = HWCOUNTER {
        COUNTER = <CounterName>;
        ISRPRORTITY = <priority>;
        TimerHardware = <TypeOfTimer> {
            ...
        };
    };
    SecondTimer = SWCOUNTER {
        ...
    };
};
```

For More Information: www.freescale.com
COUNTER = <CounterName>;
ISRPRIORITY = <priority>;
TimerHardware = <TypeOfTimer> {
    TimerModuloValue = <TimerModuloValue>;
};
}

TimeScale = TRUE {
    TimeUnit = <ticks, ns, us, ms>;
    ScalePeriod = <timescale period>;
    Step = SET {
        StepNumber = <1,2,3,...>;
        StepTime = <time to next step>;
        TASK = <ID of task to activate>;
    }
    ...
};

...
DeclareCounter( <CounterName> );
DeclareAlarm( <AlarmName> );

Run-time Services

OSEK 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>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitCounter</td>
<td>Sets the initial value of the counter</td>
</tr>
<tr>
<td>CounterTrigger</td>
<td>Increments the counter value and process attached alarms</td>
</tr>
<tr>
<td>GetCounterValue</td>
<td>Gets the counter current value</td>
</tr>
<tr>
<td>GetCounterInfo</td>
<td>Gets counter parameters</td>
</tr>
<tr>
<td>SetRelAlarm</td>
<td>Sets the alarm with a relative start value</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td>Sets the alarm with an absolute start value</td>
</tr>
<tr>
<td>CancelAlarm</td>
<td>Cancels the alarm: the alarm is transferred into the STOP state</td>
</tr>
<tr>
<td>GetAlarm</td>
<td>Gets the time left before the alarm expires</td>
</tr>
<tr>
<td>StartTimeScale</td>
<td>Starts TimeScale processing</td>
</tr>
<tr>
<td>StopTimeScale</td>
<td>Cancels TimeScale processing</td>
</tr>
<tr>
<td>GetAlarmBase</td>
<td>Gets alarm base (attached counter) parameters</td>
</tr>
</tbody>
</table>

NOTE InitCounter, CounterTrigger, GetCounterValue, GetCounterInfo,
StartTimeScale, and StopTimeScale services are not defined in
OSEK/VDX Operating System, v.2.2, 10 September 2001
specification. This is OSEKturbo extension of OSEK OS.

Examples of the run-time service usage are provided in “System
Services”.

Constants

For all counters, the following constants are defined:

- OSMAXALLOWEDVALUE_cname
  Maximum possible allowed value of counter <cname> in ticks.
**Counters and Alarms**

**Programming Issues**

- **OSTICKSPERBASE\_cname**
  Number of ticks required to reach a specific unit of counter <cname>.

- **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;

- **OSTICKSPERBASE2**
  Number of ticks that are required to reach a counter-specific value in the second counter;

- **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 OSEKturbo extension of OSEK OS.
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 OSEK 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 OSEK 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 WaitFor 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 (see example in “Examples of Using Events”).

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

**Programming Issues**

**Configuration Options**

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

The OSEK 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;
}
```

The declaration statement *DeclareEvent* may be used for compatibility with previous OSEK versions. It may be omitted in application code.

*DeclareEvent* looks like the following:

```
DeclareEvent( <Event> );
```

Some task event which used by the task as internal flags can be undefined in 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.

For More Information: www.freescale.com
Run-time Services

OSEK 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

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetEvent</td>
<td>Sets events of the given task according to the event mask</td>
</tr>
<tr>
<td>ClearEvent</td>
<td>Clears events of the calling task according to the event mask</td>
</tr>
<tr>
<td>GetEvent</td>
<td>Gets the current event setting of the given task</td>
</tr>
<tr>
<td>WaitEvent</td>
<td>Transfers the calling task into the waiting state until specified events are set</td>
</tr>
</tbody>
</table>

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

For More Information: www.freescale.com
Communication

This chapter describes message concept in OSEK and its usage.

This chapter consists of the following sections:

- Message Concept
- Unqueued Messages
- Queued Messages
- Data Consistency
- Programming Issues

Message Concept

In the OSEK Operating System communication between application tasks and/or ISRs takes place via messages. Communication concept and message management system services are based on the OSEK/VDX Communication, v.2.2.2, 18 December 2000 specification. OSEKturbo OS supports CCCB (Communication Conformance Class B) which includes support of internal Unqueued and Queued Messages.

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.

For More Information: www.freescale.com
In OSEK OS message objects are referenced by tasks and ISRs via the unique identifiers defined by the user at the configuration stage.

The OSEK 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 direct access operation.

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 want to 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 message.

OSEK OS communication services provide all means for internal message transfers. To transfer data over the network, the OSEK Communication System (COM) shall be used, which is designed to handle all other types of communication through the network. And OSEK OS communication services provide an interface for application tasks to exchange data. 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. Tasks and ISRs have personal accessors to read or to write message. Accessor type is defined by SENT and RECEIVED values of ACCESSOR attribute respectively. Also a message can be accessed directly through the message buffer or indirectly through the message copy. Access through the message copy guarantees consistency of information in the message between adjacent operations of send/receive. This behavior is defined by WITHOUTCOPY attribute for each defined ACCESSOR. 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. Allocation of memory for message copies depends on the MessageCopyAllocation\textsuperscript{1} attribute.
It specifies whether SysGen will generate copies of messages in
global memory or message copies are allocated in the application by
the user. This is defined by OS and USER values respectively. The
user can place copies of messages into the global or into the local
memory of the functions. ISR accessors access the messages through
the copies only.

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

When an Unqueued Message is received with copy the information
from the message buffer is copied to the message copy and further
work with message information is performed using this copy.

When an Unqueued Message is sent with copy the information from
the message copy is replicated to the message buffer and message
initialization is performed using the message copy.

When an Unqueued Message is received without copy the
information from the message buffer is accessed directly by the user
application and consistency of the information is not guaranteed.

When an Unqueued Message is sent without copy the information
from the message buffer is accessed directly by the user application
and consistency of the information is not guaranteed.

The \texttt{ACCESSNAME} attribute defines the symbol that will be used in
C code to access message data. \texttt{AccessNameRef} of \texttt{SendMessage} /
\texttt{ReceiveMessage} services is a pointer to symbol specified in
\texttt{ACCESSNAME} attribute. For example:

\begin{verbatim}
type _accessor;
_accessor.value1 = 0;
_accessor.value2 = 20;
SendMessage (MsgA, &_accessor);
\end{verbatim}

where type is the value of CDATATYPE attribute for MsgA
message object in OIL file. Then _accessor can be used as a pointer
to the message body.

\footnote{this attribute is OSEKturbo extension of the OSEK OS}
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. Only tasks (not ISRs) have personal accessors to read or to write message. Accessor type is defined by SENT and RECEIVED values of ACCESSOR attribute respectively. The first message in the queue can be accessed through the message copy. ACCESSOR for a queued message cannot be defined with WITHOUTCOPY attribute equal to TRUE. The send operation adds value of a message 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. Allocation of memory for message copies depends on the MessageCopyAllocation attribute. It specifies whether SysGen will generate copies of messages in the global memory or message copies are allocated in the application by the user. This is defined by OS and USER values respectively. 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.

When a Queued Message is sent the information from the message copy is added to the end of the message queue.

The ACCESSNAME attribute defines the symbol that will be used in C code to access message data. AccessNameRef of SendMessage/ReceiveMessage services is a pointer to symbol specified in ACCESSNAME attribute. For example:

```c
ctype _accessor;
_accessor.value1 = 0;
```
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

- **MessageCopyAllocation**
  This OSEKturbo specific attribute specifies whether System Generator generates copies of messages in global memory or message copies are allocated by the user.

Identifiers

The following names are used in the OSEK Operating System to operate with messages:

- **SymbolicName**
  This is a unique name representing a message. It can be used only in conjunction with calls of the message service. A SymbolicName has not to be a data type. Variables or constants of SymbolicName can be declared or used.

- **AccessName**
  This is a unique name defining access to a message object. Depending on the chosen configuration, a distinction is made between the following AccessName scheme:
WITHCOPY configuration:
An application variable exists as a copy of the message. The name of the variable is the AccessName. This variable contains a copy of the corresponding message object.

WITHOUTCOPY configuration:
The message object data is accessed via the AccessName. This AccessName is a static link: it refers directly to the message object data. The AccessName refers to the same data (RAM) as the message object.

- AccessNameRef
  This is the address of the message buffer or message copy.

- FlagType
  The abstract data type for flag identification.

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

```c
MESSAGE MsgA {
  TYPE = UNQUEUED;
  CDATATYPE = "long int";
  ACTION = SETEVENT {
    TASK = task1;
    EVENT = eventC;
  };
};
```

In detail message configuration statements is described in “Message Definition”.

There are no constructional elements defined for messages.

Run-time Services
OSEKturbo 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>
<thead>
<tr>
<th>Service Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendMessage</td>
<td>Updates the message</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>Gets the message</td>
</tr>
<tr>
<td>GetMessageStatus</td>
<td>Gets message status</td>
</tr>
<tr>
<td>GetMessageResource</td>
<td>Sets message status to BUSY</td>
</tr>
<tr>
<td>ReleaseMessageResource</td>
<td>Clears BUSY message status</td>
</tr>
<tr>
<td>ReadFlag</td>
<td>Returns the value of the specified notification flag</td>
</tr>
<tr>
<td>ResetFlag</td>
<td>Sets the specified notification flag to FALSE</td>
</tr>
<tr>
<td>InitCOM</td>
<td>Initializes COM hardware (empty in OSEKturbo)</td>
</tr>
<tr>
<td>CloseCOM</td>
<td>Resets COM hardware (empty in OSEKturbo)</td>
</tr>
<tr>
<td>StartCOM</td>
<td>Initializes COM internal data</td>
</tr>
<tr>
<td>StopCOM</td>
<td>Stops communications</td>
</tr>
</tbody>
</table>

Examples of the run-time services usage are provided in “System Services”.

**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 `CALLBACKNAME` attribute, in this case `ACTION` shall be defined as `CALLBACK`. The function is called when message arrives.

The callback function is usual user’s function. It is executed on OS or ISR level and only `SendMessage` and `ReceiveMessage` services are allowed in it.

The callback function shall have next definition:

```c
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.

If the message is used in the WITHCOPY configuration and the value of `MessageCopyAllocation` attribute is USER, then the variables to hold message’s copies must be defined within the user’s code by means of using the regular C-language definitions. If `MessageCopyAllocation` attribute is set to OS, then the variables to hold message’s copies are defined by SysGen and the user shall not define this variables in code.

For example, if the user defines the message `MsgA` having type `int`, then user’s code may access message using the following statements:

```c
int _MsgA;

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

If the message is configured as WITHOUTCOPY property, then the pointer to the message body should be defined within the user’s code using regular C-language statements. Again, because system generator creates `typedef` declaration for message item, it is recommended to use this declaration for definition of pointer, which is used to access message data.

For example, if the user defines the WITHOUTCOPY message `MsgB`, having the type `int` and `ACCESSNAME MESSB`, then user’s code may access message using the following statements:

```c
ReceiveMessage( MsgB, &MESSB );
if( MESSB == 2 ) { MESSB = 1; }
SendMessage( MsgB, &MESSB );
```
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
- Hook Routines
- Error Handling
- Start-up Routine
- Application Modes
- System Shutdown
- Programming Issues

General

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

These hook routines in OSEK OS are:

For More Information: www.freescale.com
Error Handling and Special Routines

Hook Routines

- 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
- Using an implementation-dependent calling interface
- Part of the operating system, but user defined
- Implemented by the user
- Standardized in interface per OSEK OS implementation, 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 OSEK OS hook routines are intended for:

- System startup. The corresponding hook routine (StartupHook) is called after the operating system startup and before the scheduler is running
- 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
- System shutdown. The corresponding hook routine (ShutdownHook) is called

Besides standard OSEK OS hook routines there are additional hook routine in OSEKturbo OS/MPC55xx used for:

- Performing user’s specific operations when no task is running (IdleLoopHook)

The OSEKturbo OS provides the following hook routines: ErrorHook, PreTaskHook, PostTaskHook, StartupHook, ShutdownHook and IdleLoopHook\(^1\). The user must create the code of these routines, the OS only provides description of function prototypes.

---

\(^1\) IdleLoopHook routine is OSEKturbo extension of OSEK OS.
• **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.

• **PreTaskHook** – this hook is called before the operating system enters the context of the task. 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 context of the task. 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.

• **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.

• **ShutdownHook** – This hook is called when the service ShutdownOS has been called. It is called before the Operating System shuts down itself.

• **IdleLoopHook** – This hook is called from scheduler idle loop (see “General”). It is not possible to call any OSEK OS directives from this hook. Hardware dependent code may be placed here. If the user uses IdleLoopHook to enter one of lower power modes it is recommended to set HCLowPower to FALSE.

Time stamps can be integrated individually into an application software with the help of hook routines PreTaskHook and PostTaskHook. The user can optionally use the hook routines or establish a watchdog task that takes ‘one-shot displays’ of the operating system status.
Not all OS services may be called by the hook routines, please refer to the table below:

### Table 10.1  OSEK OS System Services for Hook Routines

<table>
<thead>
<tr>
<th>Service</th>
<th>Hook routines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Error Hook</td>
</tr>
<tr>
<td>ActivateTask</td>
<td>--</td>
</tr>
<tr>
<td>TerminateTask</td>
<td>--</td>
</tr>
<tr>
<td>ChainTask</td>
<td>--</td>
</tr>
<tr>
<td>Schedule</td>
<td>--</td>
</tr>
<tr>
<td>GetTaskId</td>
<td>allowed&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>GetTaskState</td>
<td>allowed</td>
</tr>
<tr>
<td>DisableAllInterrupts</td>
<td>--</td>
</tr>
<tr>
<td>EnableAllInterrupts</td>
<td>--</td>
</tr>
<tr>
<td>SuspendAllInterrupts</td>
<td>allowed</td>
</tr>
<tr>
<td>ResumeAllInterrupts</td>
<td>allowed</td>
</tr>
<tr>
<td>SuspendOSInterrupts</td>
<td>--</td>
</tr>
<tr>
<td>ResumeOSInterrupts</td>
<td>--</td>
</tr>
<tr>
<td>GetResource</td>
<td>--</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>--</td>
</tr>
<tr>
<td>SetEvent</td>
<td>--</td>
</tr>
<tr>
<td>ClearEvent</td>
<td>--</td>
</tr>
<tr>
<td>GetEvent</td>
<td>allowed</td>
</tr>
<tr>
<td>WaitEvent</td>
<td>--</td>
</tr>
<tr>
<td>InitCounter</td>
<td>--</td>
</tr>
<tr>
<td>CounterTrigger</td>
<td>--</td>
</tr>
<tr>
<td>GetCounterValue</td>
<td>allowed</td>
</tr>
</tbody>
</table>

<sup>a</sup> No restrictions
Table 10.1 OSEK OS System Services for Hook Routines

<table>
<thead>
<tr>
<th>Service</th>
<th>Hook routines</th>
<th>Error Hook</th>
<th>PreTask Hook</th>
<th>PostTask Hook</th>
<th>Startup Hook</th>
<th>Shutdown Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCounterInfo</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>GetAlarmBase</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>GetAlarm</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>SetRelAlarm</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>CancelAlarm</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>SendMessage</td>
<td>allowed for unqueued messages</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>allowed for unqueued messages</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>GetActiveApplicationMode</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
</tr>
<tr>
<td>StartOS</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>ShutdownOS</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>allowed</td>
<td>--</td>
</tr>
<tr>
<td>GetRunningStackUsage</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>GetStackUsage</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>GetTimeStream</td>
<td>allowed</td>
<td>allowed</td>
<td>allowed</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

* It may happen that currently no task is running. In this case the service returns the task ID INVALID_TASK.

**NOTE** It is not possible to call any OSEK OS services from `IdleLoopHook` hook routine.
Error Handling

Error Interface

The hook routine ErrorHook is provided to handle possible errors detected by the OSEK 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.

The special system routine ShutdownOS is intended to shut down the system. ShutdownOS may be called by the user on experiencing a fatal error. These service routine is provided by the OSEK Operating System as opposed to the ErrorHook routine, which should be written by the user. User hook ShutdownHook is called by ShutdownOS.

The OSEK 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. According to OSEK 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 OSEK OS.

The OSEKturbo OS specifies the following error codes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>No error, successful completion</td>
</tr>
<tr>
<td>E_OS_ACCESS</td>
<td>1</td>
<td>Access to the service/object denied</td>
</tr>
<tr>
<td>E_OS_CALLEVEL</td>
<td>2</td>
<td>Access to the service from the ISR is not permitted</td>
</tr>
<tr>
<td>E_OS_ID</td>
<td>3</td>
<td>The object ID is invalid</td>
</tr>
<tr>
<td>E_OS_LIMIT</td>
<td>4</td>
<td>The limit of services/objects exceeded</td>
</tr>
<tr>
<td>E_OS_NOFUNC</td>
<td>5</td>
<td>The object is not used, the service is rejected</td>
</tr>
<tr>
<td>E_OSRESOURCE</td>
<td>6</td>
<td>The task still occupies the resource</td>
</tr>
<tr>
<td>E_OS_STATE</td>
<td>7</td>
<td>The state of the object is not correct for the required service</td>
</tr>
<tr>
<td>E_OS_VALUE</td>
<td>8</td>
<td>A value outside of the admissible limit</td>
</tr>
</tbody>
</table>
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 occurred not inside a service function, but in the OS dispatcher then the special value *OSServiceId_NoService* is returned.

- the macros of type *<OSError_serviceIDParameterID>()*, where serviceID is the name of the service and parameterID is the name of the first parameter, returns the value of the first parameter.

---

1. it is OSEKturbo extension of OSEK OS, not specified in OSEK OS v.2.2 specification

### Table 10.2 OSEK OS Error Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OS_SYS_STACK(^a)</td>
<td>17</td>
<td>Task stack overflow</td>
</tr>
<tr>
<td>E_OS_SYS_ORDER (^a)</td>
<td>18</td>
<td>Incorrect order of function calling</td>
</tr>
<tr>
<td>E_OS_SYS_MAINSTACK(^a)</td>
<td>19</td>
<td>Main stack overflow</td>
</tr>
<tr>
<td>E_OS_SYS_ISRSTACK(^a)</td>
<td>20</td>
<td>ISR stack overflow</td>
</tr>
<tr>
<td>E_COM_BUSY</td>
<td>33</td>
<td>Message is in use by application task/function</td>
</tr>
<tr>
<td>E_COM_ID</td>
<td>35</td>
<td>Invalid message name passed as parameter</td>
</tr>
<tr>
<td>E_COM_LIMIT</td>
<td>36</td>
<td>Overflow of FIFO associated with queued messages</td>
</tr>
<tr>
<td>E_COM_LOCKED</td>
<td>39</td>
<td>Rejected service call, message object locked due to a pending operation</td>
</tr>
<tr>
<td>E_COM_NOMSG</td>
<td>41</td>
<td>No message available</td>
</tr>
</tbody>
</table>

\(^a\) E_OS_SYS_xxx are not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

For More Information: www.freescale.com
For example, `OSErroError_ActivateTask_TaskID()` returns the task identifier if `ErrorHook` was called because of error detected in `ActivateTask`.

In order use the `OSGetServiceId` 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 OSEKturbo OS/MPC55xx v.2.2, User’s Manual.

**Extended Status**

The OSEK 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.

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 OSEK 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 OSEK OS**

<table>
<thead>
<tr>
<th>(Re-)Start</th>
<th>hardware-specific initialization code</th>
<th>call to StartOS</th>
<th>operation system initialization code</th>
<th>OS executes StartupHook</th>
<th>timers initialization</th>
<th>OS kernel is running</th>
<th>first user task is running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

During StartupHook all system interrupts are disabled

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 OSEK 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.
OSEKturbo OS supports up to 8 application modes.

**System Shutdown**

The special `ShutdownOS` service exists in OSEK 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:

- **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

For More Information: www.freescale.com
• **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

• **IdleLoopHook**
  If this OSEKturbo specific option is turned on, the *IdleLoopHook* is called from the scheduler idle loop
System Configuration

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

This chapter consists of the following sections:

- General
- Application Configuration File
- OIL Concept

General

The OSEK 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.

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)\(^1\). 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.

\(^1\) One version of SG is delivered - the 32-bit version (`sysgen.exe`) for Windows 98 and Windows 2000.
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.

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 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 OSEK 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 = "2.3";

**OIL Format**

The Standard OIL is intended to configure OSEK OS Operating System (OS). It is strictly defined by the *OSEK/VDX System Generation OIL: OSEK Implementation Language, v.2.3, 10 September 2001* specification.

**Implementation Definition**

The Implementation Definition defines implementation specific features for the particular OSEK 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 OSEK implementation. Additional non-standard attributes can be defined for the objects for the particular OSEK 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 OSEK implementations and then included in user's OIL files. The OSEKturbo OS/MPC55xx implementation is described in the “ost22.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.

For More Information: www.freescale.com
The structure for Implementation Definition part is shown using the following syntax:

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

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

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

Object type is defined by the object keyword. For OSEKturbo OS/MPC55xx implementation the following object types are implemented:

OS, APPMODE, TASK, ISR, RESOURCE, EVENT, COUNTER, ALARM, 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:

```plaintext
<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 OSEK 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 OSEKturbo 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...
   };
   ...
```

For More Information: www.freescale.com
System Configuration
OIL Concept

};

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., the STACKSIZE is defined and no EVENT(s) is defined for a task) since it leads to error or warning messages.
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
- OS Definition
- Task Definition
- ISR Definition
- Resource Definition
- Event Definition
- Counter Definition
- Alarm Definition
- Message Definition
- Application Modes Definition
- COM Definition
- NM Definition
- OSEKturbo Performance Dependency

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
System Objects Definition

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>;
CC = <BCC1 / ECC1 / AUTO>;
DEBUG_LEVEL = <0 / 1 / 2 / 4>;
BuildNumber = <TRUE / FALSE>;
FastTerminate = <TRUE / FALSE>;
FastScheduler = <TRUE / FALSE>;
```
MessageCopyAllocation = <USER / OS>;
ResourceScheduler = <TRUE / FALSE>;

- **STATUS** (ENUM)
  The attribute specifies the debugging ability of OS, defines whether additional run-time error checks are supported by OS for OSEK API calls or not. If the system has the *extended status* some 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. In the *standard status* OS performs only very limited set of checks, the performance is increased and the amount of consumed memory is decreased.

  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.

- **CC** (ENUM or AUTO)
  The 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)
  This OSEKturbo 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, static and run time ORTI (running task and system calls identification) modes will be used. If the attribute is set to 4, then dynamic and static ORTI and debugging services are implemented. See [“Debugging Application”](#) for details.

- **BuildNumber** (BOOLEAN)
  This OSEKturbo specific attribute specifies whether build number in ASCII form should be incorporated into OS binary image (ROM code) or not. If *BuildNumber* is set to FALSE then the user shall remove “OsBuildNumber” entry from linker command file (which is generated from makefile) for CodeWarrior compiler.
• **FastTerminate** (BOOLEAN)
  This OSEKturbo specific attribute specifies whether the fast versions of *Terminate/ChainTask* are used in BCC1 class.

• **FastScheduler** (BOOLEAN)
  This OSEKturbo specific attribute specifies whether the OS places global variabnles, used by scheduler, into the CPU general purpose registers.

• **MessageCopyAllocation** (ENUM)
  This OSEKturbo specific attribute specifies whether System Generator generates copies of messages in global memory or message copies are allocated by the user. The user can place copies of messages either in global memory or in local memory. This attribute affects all message copies in the application.

• **ResourceScheduler** (BOOLEAN)
  This OSEKturbo specific attribute specifies whether RES_SCHEDULER should be supported or not.

### 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 = <MPC MPC5500> {
  ClockFrequency = <integer />;
  SysTimer = <HWCOUNTER / SWCOUNTER / NONE> {
    COUNTER = <name of COUNTER>;
    ISRPRIORITY = <integer>;
    Period = <integer / AUTO>;
    TimerHardware = <name of hardware timer> {
      TimerModuloValue = <integer / AUTO>;
    };
  };
}
SecondTimer = <HWCOUNTER / SWCOUNTER / NONE> {
  COUNTER = <name of COUNTER>;
  ISRPRIORITY = <integer>;
  Period = <integer / AUTO>;
  TimerHardware = <name of hardware timer> {
    TimerModuloValue = <integer / AUTO>;
  };
};
```
HCLowPower = <TRUE / FALSE>;

• **TargetMCU** (ENUM)
  This OSEKturbo specific attribute defines the target for which
  the OS will be configured.

  It is highly recommended to use sample makefile as a basis
  for building application to provide consistency in compiler/
  linker settings.

  Value MPCMAC7100 corresponds to any MCU of this MCU
  family. Any target specific parts of code (such as timer
  support) are disabled in this case.

• **ClockFrequency** (UINT32)
  This OSEKturbo 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 AUTÔ. Otherwise this attribute is
  used to calculate OSTICKDURATION.

• **ClockDivider** (UINT32)
  This OSEKturbo 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)
  This OSEKturbo 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.

• **SysTimer** (ENUM)
  This OSEKturbo 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. The attribute can not be defined, if TargetMCU
  is set to MPC.

  If this attribute is SWCOUNTER or HWCOUNTER, then
  specific subattributes can be defined for the system timer.
**SecondTimer** (ENUM)
This OSEKturbo specific attribute specifies whether the internal OS second timer is used or not. The attribute can not be defined, if TargetMCU is set to MPC.

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)
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)
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)
This OSEKturbo 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/Value or/and TimerModuloValue is AUTO. This attribute is ignored if corresponded Prescaler/Value 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)
This OSEKturbo 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: FIT for MPC55xx; the possible values for HWCounter are DEC for MPC55xx;
The TimerHardware attributes in SysTimer and SecondTimer blocks can not have the same value. See “PowerPC Platform-Specific Features” for details about possible meanings of these parameters.

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

**TimerModuloValue** (UINT32)
This OSEKturbo specific attribute specifies timer modulo value. The value of this attribute is fully hardware-dependent. For more details see “PowerPC 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.

**HCLowPower** (BOOLEAN)
This OSEKturbo specific attribute defines that low power mode shall be used when there is no ready or running tasks and scheduler goes into idle loop.

In general, it is recommended to define this attribute to reduce power consumption.

**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):

\[
\text{IsrStackSize = \text{<integer>};} \\
\text{StackOverflowCheck = \text{TRUE / FALSE};}
\]

- **IsrStackSize** (UINT32)
  This OSEKturbo specific attribute specifies the ISR stack size in bytes. The stack is used in ISRs category 2. In this case the current status of the processor is saved onto the current stack, and stack is switched to the interrupt stack.

  This attribute shall be defined if any ISR category 2 (include system or second timer) are defined and Extended tasks are present in the configuration. Otherwise single stack is used for Basic tasks and ISRs and this attribute is ignored.

- **StackOverflowCheck** (BOOLEAN)
  This OSEKturbo specific attribute turns on stack overflow runtime checking and stack usage services. If this attribute is switched on, ErrorHook is defined and OS detects stack overflow, then the ErrorHook is called with appropriate error status code. This mode is intended for debugging.

**Task Related Attributes**

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):

\[
\text{TimeScale = \text{<TRUE / FALSE>}} \{ \\
\text{ScalePeriod = \text{<integer / AUTO>};} \\
\text{TimeUnit = \text{<ticks / ns / us /ms>};} \\
\text{Step = \text{<SET>}} \{ \\
\text{StepNumber = \text{<integer>};} \\
\text{StepTime = \text{<integer>};} \\
\text{TASK = \text{name of TASK>};} \\
\};
\]

- **TimeScale** (BOOLEAN)
  This OSEKturbo specific attribute specifies Time Scale mechanism.

For More Information: [www.freescale.com](http://www.freescale.com)
• **ScalePeriod** (UINT32)
  This OSEKturbo specific attribute specifies full period of time scale in chosen measurement units. It is used during system generation time to check that ScalePeriod is equal to sum of StepTime attributes of all steps. The attribute is ignored if it is AUTO.

• **TimeUnit** (ENUM)
  This OSEKturbo specific attribute specifies measurement units: ticks means ticks of System Timer, ns means nanoseconds, us - microseconds, and ms - milliseconds.

• **Step** (ENUM)
  This OSEKturbo specific attribute defines one of step elements in the Time Scale.

• **StepNumber** (UINT32)
  This OSEKturbo specific attribute specifies the order of steps. The numbers shall be unique.

• **StepTime** (UINT32)
  This OSEKturbo specific attribute specifies time to next task activation in measurement units chosen by means of the TimeUnit attribute.

• **TASK** (reference)
  The reference specifies the task to be activated.

**Interrupt Related Properties**

This group of attributes defines parameters of ISR execution. 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):

```
InterruptDispatcher = <Onelevel / MultiLevel>;
```

• **InterruptDispatcher** (ENUM)
  The attribute specifies the way of the external interrupts processing. It can have one of the following values:
  
  – OneLevel - the interrupt dispatcher is supported, but multilevel external interrupt emulation is not supported. In this case all external interrupts will be disabled during the interrupt processing.
  
  – MultiLevel - the interrupt dispatcher and multilevel external interrupt are supported. In this case interrupt processing can be
interrupted by the other interrupt of the higher priority. The priority of ISR is specified by its \textit{PRIORITY};

\textbf{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):

\begin{verbatim}
STARTUPHOOK = <TRUE / FALSE>;
SHUTDOWNHOOK = <TRUE / FALSE>;
PRETASKHOOK = <TRUE / FALSE>;
POSTTASKHOOK = <TRUE / FALSE>;
ERRORHOOK = <TRUE / FALSE>;
USEGETSERVICEID = <TRUE / FALSE>;
USEPARAMETERACCESS = <TRUE / FALSE>;
IdleLoopHook = <TRUE / FALSE>;
\end{verbatim}

- \textbf{STARTUPHOOK (BOOLEAN)}
  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 \textit{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.

- \textbf{SHUTDOWNHOOK (BOOLEAN)}
  The attribute defines whether the user’s-provided hook is called by the system during system shutdown or not (the \textit{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.

- \textbf{PRETASKHOOK (BOOLEAN)}
  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)**
  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.

- **ERRORHOOK (BOOLEAN)**
  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 or not (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)**
  This attribute specifies the possibility of usage the access macros to the service ID in the error hook.

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

- **IdleLoopHook (BOOLEAN)**
  This OSEKturbo specific attribute defines whether the user’s-
System Objects Definition

Task Definition

The provided hook is called by system from scheduler idle loop (when there are no tasks in ready or running state) or not (the IdleLoopHook hook routine).

This hook is intended for manipulation with hardware registers (like COP). It is not possible to call any OSEK OS services from this hook.

Floating Point Related Attributes

PowerPC Book E floating-point instructions are not supported in MPC5500 hardware therefore the OSEK OS/MPC55xx does not support floating point operations.

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 OSEK tasks. The syntax of the task object is as follows:

```plaintext
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>;
    ACCESSOR = <SENT / RECEIVED> {
        MESSAGE = <name of MESSAGE>;
        WITHOUTCOPY = <TRUE / FALSE>;
        ACCESSNAME = <string>;
    };
};
```

Attributes

The TASK object has the following attributes:

For More Information: www.freescale.com
System Objects Definition

Task Definition

- **PRIORITY** (UINT32)
  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. OSEK 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 0x7FFFFFFF.

- **SCHEDULE** (ENUM)
  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)
  The attribute determines whether the task is activated during the system start-up procedure or not.

- **APPMODE** (reference)
  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.

- **ACTIVATION** (UINT32)
  The attribute defines the maximum number of queued activation requests for the task. OSEKturbo OS does not support multiple activation, so this value is restricted to 1.

- **STACKSIZE** (UINT32)
  This OSEKturbo specific attribute defines the task stack size in bytes. It is applicable for extended tasks only.

- **RESOURCE** (reference)
  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)
  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
System Objects Definition

ISR Definition

pointed to define the event mask in case of auto-assignment (see section “Event Definition”).

- **ACCESSOR** (ENUM)
  The attribute is used to define type of usage for the message. The task uses the accessor for definition of references to sent or received messages.

- **MESSAGE** (reference)
  The reference specifies the message to be sent or received by the task. This parameter is a single reference, it has to be defined.

- **WITHOUTCOPY** (BOOLEAN)
  The attribute defines whether a local copy of the message is used or not. This attribute has to be defined.

- **ACCESSNAME** (STRING)
  The attribute defines the reference which can be used by an application to access the message data. This attribute has to be defined. The local copy of the message shall be defined with this name in an application code in case an access to the message is processed using local copy.

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:

```plaintext
ISR <name of ISR> {
    CATEGORY = <1 / 2>;
    PRIORITY = <integer>;
    IrqChannel = <enum>{;
        IrqNumber = <integer>;
    };
    RESOURCE = <name of RESOURCE>;
    ACCESSOR =<SENT / RECEIVED> {
        MESSAGE = <name of MESSAGE>;
        ACCESSNAME = <string>;
    };
}
```

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)**
  The attribute specifies category of the Interrupt Service Routine. (see “ISR Categories” for Interrupt Service Routine Categories details).

  Allowed values for MPC55xx applications are 1-15.

- **IrqChannel (ENUM)**
  The attribute specifies the interrupt channel. The available values of this attribute are DEC, FIT, EXTERNAL, INTERNAL.

- **IrqNumber (UINT32)**
  The attribute specifies the interrupt source for the EXTERNAL IrqChannel. The available values of this attribute are 0-300. (For the correspondence between IrqNumber value and external interrupt sources please the appropriate Hardware Technical References)

- **RESOURCE (reference)**
  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.

- **ACCESSOR (ENUM)**
  The attribute is used to define type of usage for the message. The ISR uses the accessor for definition of references to sent or received messages.

- **MESSAGE (reference)**
  The reference specifies the message to be sent or received by the task. This parameter is a single reference, it has to be defined.

- **ACCESSNAME (STRING)**
  The attribute defines the reference which can be used by an application to access the message data. This attribute has to be defined. The local copy of the message shall be defined with this name in an application code.

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

For More Information: www.freescale.com
System Objects Definition
Event Definition

"Management" describes resource concept in OSEK. The syntax of the resource object is as follows:

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

- **RESOURCEPROPERTY** (ENUM)

  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)

  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**.

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 OSEK. 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)
  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 OSEK 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. OSEK counters are described in section “Counters and Alarms”. The syntax of the counter object is:

```plaintext
COUNTER <name of COUNTER> {
    MINCYCLE = <integer>;
    MAXALLOWEDVALUE = <integer>;
    TICKSPERBASE = <integer>;
};
```

### Attributes

The object has the following standard attributes:

• **MINCYCLE** (UINT32)
  The attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter. (In fact, this parameter has a sense only for systems with extended OS status since it is checked in this case only.)

• **MAXALLOWEDVALUE** (UINT32)
  The attribute defines the maximum allowed counter value. After the counter reaches this value it rolls over and starts count again from zero.

• **TICKSPERBASE** (UINT32)
  The number of ticks that are required to reach a counter-specific value. This value cannot be derived automatically from other counter related attributes. The interpretation is up to the user.
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> {  
    TASK = <name of TASK>;  
    EVENT = <name of EVENT>;  
    ALARMCALLBACKNAME = <string>;  
  };  
  AUTOSTART = <TRUE / FALSE> {  
    ALARMTIME = <integer>;  
    CYCLETIME = <integer>;  
    APPMODE = <name of APPMODE>;  
  };  
};
```

Attributes

The object has the following attributes:

- **COUNTER** (reference)
  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)
  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.
• ALARMCALLBACKNAME (STRING)
  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.

• TASK (reference)
  The reference to a task which is to be notified via activation or event setting when the alarm expires.

• EVENT (reference)
  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)
  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)
  The attribute defines the time when the alarm shall expire first. The attribute should be be defined if the AUTOSTART attribute is set to TRUE.

• CYCLETIME (UINT32)
  The attribute defines the cycle time of a cyclic alarm. The attribute should be defined if the AUTOSTART attribute is set to TRUE.

• APPMODE (reference)
  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.
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> {  
    TYPE = <QUEUED / UNQUEUED>;  
    QUEUEDEPTH = <integer>;  
    CDATATYPE = <string>;  
    ACTION = <ACTIVATE TASK / SET EVENT / CALLBACK / FLAG / NONE> {  
        TASK = <name of TASK>;  
        EVENT = <name of EVENT>;  
        CALLBACKNAME = <string>;  
        FLAGNAME = <string>;  
    };  
};

Attributes

The object has the following standard attributes:

- **TYPE (ENUM)**
  The attribute specifies type of the message. In accordance with the OSEK specification there are two types of messages: unqueued and queued. Queued message data is buffered and consumed by receive operations. Unqueued message data is not consumed and is overwritten by each call to SendMessage.

- **QUEUEDEPTH (UINT64)**
  The attribute is specified if the message has a queue. If used for internal communication, the COM conformance class will be CCCB. The parameter can be defined only if the TYPE attribute is QUEUED.

- **CDATATYPE (STRING)**
  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.
System Objects Definition
Application Modes Definition

(signed, unsigned, short, long) and also structure or union specifier (starting strict 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.

• ACTION (ENUM)
The attribute defines which type of task notification is used when the message arrives. Only one action per message is provided.

• TASK (reference)
The reference specifies the task which shall be notified when the message arrives.

This reference shall be defined only if the value of the ACTION attribute is ACTIVATETASK or SETEVENT.

• EVENT (reference)
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 ACTION attribute is SETEVENT.

• CALLBACKNAME (STRING)
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 ACTION attribute is CALLBACK.

• FLAGNAME (STRING)
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 ACTION attribute is FLAG.

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 OSEK OS application mode.

For More Information: www.freescale.com
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>;
```

OSEKturbo 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> {
    USEMESSAGERESOURCE = <TRUE / FALSE>;
    USEMESSAGESTATUS = <TRUE / FALSE>;
};
```

**Attributes**

The object has the following standard attributes:

- **USEMESSAGERESOURCE** (BOOLEAN)
  The attribute specifies if the message resource mechanism is used.

- **USEMESSAGESTATUS** (BOOLEAN)
  The attribute specifies if the message status is available.

**NM Definition**

The NM object represents the local parameters of the network management subsystem on CPU. No attributes are defined for the NM objects.
OSEKturbo Performance Dependency

The following attributes of OS object directly affect OS performance:

- **STATUS** – if this attribute is set to EXTENDED, then the system performance is decreased because OS spends additional time for checking. See “System Services” for more detailed services description.

- **StackOverflowCheck** – if this attribute is TRUE, then the stack overflow conditions are checked when each rescheduling occurs and in each ISR, thus significantly decreasing OS performance.

- **DEBUG_LEVEL** – if this attribute is not equal 0, then **StackOverflowCheck** is (implicitly) turned on. Additional resource management is affected: resources state manipulations in code added.
  If this attribute is greater than 1 additional code to support tracing is added into all OS services.

- **InterruptDispatcher** – if this is not set to None, then additional code is executed before entering any ISR. The user should set **InterruptDispatcher** to OneLevel only when it is necessary.

- **FastTerminate** – if this attribute is FALSE, then the OS saves the state at the starting point of each Task and restores it in **TerminateTask** (or **ChainTask**). If **FastTerminate** is set to TRUE, then the OS performance is increased, but all calls to **TerminateTask** or **ChainTask** services must be done from Task body function level. This attribute is applicable only for BCC1.

- **FastScheduler** – if this attribute is set to TRUE, then the OS performance is increased because OS places some of it’s variables into the CPU registers. See “General and Special Purpose Registers Usage,” for details of registers usage.

- **PRETASKHOOK** and **POSTTASKHOOK** – if they are set to TRUE then the system performance is decreased even if corresponding routines (**PreTaskHook** and **PostTaskHook**) are empty because the OS calls them at each task switch.

- **ERRORHOOK** – if set to TRUE then the system performance is decreased if **STATUS** or **StackOverflowCheck** are also set to TRUE.

- **USEGETSERVICEID** and **USEPARAMETERACCESS** - if any of them is set to TRUE and **ERRORHOOK** is set to TRUE too then system performance is decreased.

The following attributes of RESOURCE object directly affect OS performance:
**RESOURCEPROPERTY** – if this attribute is set to INTERNAL, then the system performance is decreased because OS spends additional time for setting and releasing priorities of tasks when task switch occurs.

The best system performance may be achieved by leaving all mentioned in this section attributes with their default values, except *FastTerminate* and *FastScheduler*, which shall be explicitly set to TRUE.
Building of Application

This chapter contains information on how to build a user’s application using OSEK OS. It also describes memory requirements.

This chapter consists of the following sections:

- Application Structure
- Action Sequence to Build Application
- Sample Application

Application Structure

The application developed on the OSEK 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 OSEK 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

For More Information: www.freescale.com
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 OSEK 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 OSEK OS code. This process is shown on **Figure 13.1**.
Application Configuration

Applications built using OSEK 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). This utility is delivered as one of the parts of the OSEK Operating System. This tool runs as a 32-bit
console application for Windows 2000/98 and produces header and source files.

The following command is used to run SG:

```
sysgen [-options] oil_file
```

The following command line options are intended to control SG:

### Table 13.1 System Generator Command Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b</td>
<td>Defines a format of messages output as tree-like form</td>
<td>No</td>
</tr>
<tr>
<td>-c&lt;name&gt;</td>
<td>Defines &lt;name&gt; as the output c-data file name</td>
<td>Input OIL file name with &quot;.c&quot; extension</td>
</tr>
<tr>
<td>-f&lt;name&gt;</td>
<td>Defines &lt;name&gt; of command file for SysGen command line parameters</td>
<td>No</td>
</tr>
<tr>
<td>-h&lt;name&gt;</td>
<td>Defines &lt;name&gt; as the output header file name</td>
<td>Input OIL file name with &quot;.h&quot; extension</td>
</tr>
<tr>
<td>-i&lt;path&gt;</td>
<td>Defines &lt;path&gt; as the path for include files</td>
<td>No</td>
</tr>
<tr>
<td>-n&lt;CPU&gt;</td>
<td>Defines &lt;CPU&gt; as the name of CPU for which output files are generated</td>
<td>First CPU in the file</td>
</tr>
<tr>
<td>-p&lt;name&gt;</td>
<td>Defines &lt;name&gt; as the OS property file name</td>
<td>&quot;osprop.h&quot; in the source OIL file directory</td>
</tr>
<tr>
<td>-O&lt;version&gt;</td>
<td>Specifies ORTI version</td>
<td>2.1</td>
</tr>
<tr>
<td>-o&lt;name&gt;</td>
<td>Defines &lt;name&gt; as the ORTI information file name</td>
<td>Input OIL file name with &quot;.ort&quot; extension</td>
</tr>
<tr>
<td>-t</td>
<td>Only verification for OIL input file. No output files shall be generated</td>
<td>No</td>
</tr>
<tr>
<td>-v</td>
<td>When this option is defined, SysGen shall output versions of all its components</td>
<td>No</td>
</tr>
<tr>
<td>-w</td>
<td>Suppresses all warning messages</td>
<td>No</td>
</tr>
<tr>
<td>-w&lt;message identifier&gt;</td>
<td>Suppresses warning message defined by &lt;message identifier&gt;</td>
<td>No</td>
</tr>
<tr>
<td>-A</td>
<td>Forces to generate absolute paths in include directives of configuration files instead of relative one</td>
<td>No</td>
</tr>
</tbody>
</table>

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.
Building of Application

Action Sequence to Build Application

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. By default, System Generator uses the input file name for this output file with “.h” extension.

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. By default, System Generator uses the input file name for this output file with “.c” extension.

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

OSEK Operating System 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 after OSEK OS installation. 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.

For More Information: www.freescale.com
This file must be included in all user’s and OS’ source files. Since the user can specify another name for this file the special macro OSPROPH is designed to substitute the name. After OS property file the file osapi.h shall be included to make OS services visible to the user application. Then OS configuration file (cfg.h), which contains declarations of all OS objects shall be included. Macro OSCFGH may be used to substitute a name of configuration header file. The following code could be used in all user’s files:

```c
#if !defined(OSPROPH)
#   include "osprop.h"
#else  /* !defined (OSPROPH) */
#   include OSPROPH
#endif    /* !defined (OSPROPH) */

#include "osapi.h"

#if !defined(OSCFGH)
#   include "cfg.h"
#else  /*!defined(OSCFGH)*/
#   include OSCFGH
#endif /*!defined(OSCFGH)*/
```

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.

But the user is allowed to use some other method to include the property definition and configuration header files in his/her source code.

Among other files SG generates configuration C-file containing definitions and initialization of OSEK OS configuration data declared in corresponding configuration header file. Configuration C-file is a separate module, however, in some particular OSEK 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 “cfg.c”. The data type is defined in the user’s file “user.h” and referenced in the produced file “cfg.c”. The user’s code should be the following:

USER.H file:

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

USER.C file:

```c
#include "user.h" /* include user defined data type */
...
MSGTYPE MsgA; /* user defined variables */
...```
The compiler command line has the following option:

-#APPTYPESH="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 OSEK 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 “PowerPC 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 are listed below.

OS attributes:

- *MessageCopyAllocation*
- *TimeScale* subattributes:
  - *ScalePeriod*
  - *TimeUnit*
  - *Step* subattributes: *StepNumber, StepTime, TASK*

TASK attributes:

- *PRIORITY*
- *AUTOSTART*
- *RESOURCE*
- Number of *ACCESSOR* definition and all subattributes of *ACCESSOR*: *MESSAGE, WITHOUTCOPY, ACCESSNAME*
- *STACKSIZE*

ISR attributes:

- Number of *ACCESSOR* definition and all subattributes of *ACCESSOR*: *MESSAGE, ACCESSNAME*

COUNTER attributes:

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

ALARM attributes:

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

MESSAGE attributes:

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

Sample Application

In “Sample Application” the code of an OSEK 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.
PowerPC Platform-Specific Features

This chapter discusses special OSEK 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
- General and Special Purpose Registers Usage
- Stack Size
- Features

Compiler-Specific Features

The following tools should be used to build OSEK OS applications:
- Metrowerks C/C++ Compiler for Embedded PowerPC version 3.0.4.

Used Library Functions

OSEKturbo OS itself does not use any functions from compilers runtime libraries except `memcpy` and `memset`.

Compiler Issues

Installation procedure defines environment variable values in batch files which are used for sample compilation. If they were not set during installation the user should do it manually to compile sample. These variables are the following:

OSEKDIR = [path] – path to the OSEK directory

For More Information: www.freescale.com
PowerPC Platform-Specific Features

Compiler-Specific Features

SYSGENDIR = [path] – path to the System Generator directory

CWDIR = [path] – path to the CodeWarrior compiler

See the makefile in the SAMPLE\STANDARD directory for additional information.

User should use special compiler options for compiling OSEK OS modules.

---

**NOTE**

It is recommended to use compiler/linker options list as it is set in the sample makefiles. Using of the other compiler options may lead to possible compiling/linking 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 Metrowerks Software**

**Options for Metrowerks C/C++ Compiler for Embedded PowerPC**

The following Metrowerks C/C++ compiler options should be used for compiling all OSEK OS C-source modules:

```bash
-c -fp <FPsupport> -proc Zen -nostdinc -nosyspath -ansi off -opt all -Cpp_exceptions off -DOSIVPR=<ivpr>
```

Where `<FPsupport>` is soft (software) or none (none); `<intmembase>` is the base address of the internal memory.

Please see Metrowerks compiler User’s Manual for details on compiler options.

**Options for Metrowerks Assembler for Embedded PowerPC**

The following compiler options should be used for Metrowerks assembler:

```bash
-proc Zen
```

**Options for Metrowerks Linker for Embedded PowerPC**

```bash
-fp <FPsupport>
```
Where <FPsupport> is soft (software) or none (none).

**Link OS Code into Special Segment**

There is possibility to redirect the OS code into special memory segment using linker features. This may be useful for more precise memory placement. The following fragments of the Linker Command Files contains strings for OS memory section definition. Section names and their addresses may be changed according to the user’s discretion.

**Linker Command File for Metrowerks Linker for Embedded PowerPC**

```c
MEMORY
{
    bam:   org = 0xffffffff000
    code:  org = 0x00001000
        /*address and size for OS segment */
    os_rom: org = 0x0000E000, len = 0xB000
    ram:    org = 0x40008000
}

SECTIONS
{
    /* The first group contains code and constant data exclude OS code and constants */
    GROUP : {
        /* First take all code from all objects and libraries */
        .text (TEXT) : {
            *(.fini) *(.text) *(.rodata) *(.rdata)
        }
        /* Next take all small CONST data */
        extab (CONST) : {}
        .dtors (CONST) : {}
        .ctors (CONST) : {}
        extabindex (CONST) : {}
        .rodata (CONST) : {}
    } > code
        /* The group contains OS code and constant data */
    GROUP : {
```
General and Special Purpose Registers Usage

The Special purpose register 2 (SPRG2) is used to save/restore the basic stack pointer.
If FastScheduler attribute is set to TRUE then the general purpose registers r14, r15, r16 and r17 are used by OSEKturbo OS/MPC55xx for global variables. The register r17 is used only when there are more than 32 tasks configured.

NOTE In case when FastScheduler is set to TRUE and some source files or libraries do not include osapi.h header file the compiler shall be prevented from the usage of these registers. It may be done by including the file hwspec.osregs.h into source files. If modification of source files is not possible then the compiler command line option -AddIncl="osregs.h" shall be used.
Stack Size

Generally, the recommended minimal task stack size equals:

- 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 given in the table were measured for the typical application configurations for OSEKturbo OS/MPC5xx v.2.2.1 compiled with CodeWarrior compiler, not for this ESS. OSEK is a highly scalable RTOS and most of these values depend on configuration.

Table 14.1 Stack Usage for OS Services

<table>
<thead>
<tr>
<th>System service</th>
<th>CC</th>
<th>Used bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateTask with rescheduling</td>
<td>ECC1</td>
<td>48</td>
</tr>
<tr>
<td>TerminateTask</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>TerminateTask</td>
<td>ECC1</td>
<td>32</td>
</tr>
<tr>
<td>ChainTask</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>ChainTask - Basic Task for ECC1</td>
<td>ECC1</td>
<td>32</td>
</tr>
<tr>
<td>Schedule with rescheduling</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>GetTaskId</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>GetTaskState</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>EnableAllInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>DisableAllInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>ResumeAllInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>SuspendAllInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>ResumeOSInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>SuspendOSInterrupts</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>GetResource</td>
<td>BCC1</td>
<td>16</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>BCC1</td>
<td>16</td>
</tr>
</tbody>
</table>
To estimate the approximate size of a task stack, list the function call chains, take the one with the largest stack usage and add amount used by empty task to the stack usage. If there are ISRs, add stack size needed for ISR.

**Table 14.1 Stack Usage for OS Services**

<table>
<thead>
<tr>
<th>Operation</th>
<th>CC</th>
<th>Used bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetEven with rescheduling</td>
<td>ECC1</td>
<td>32</td>
</tr>
<tr>
<td>ClearEvent</td>
<td>ECC1</td>
<td>0</td>
</tr>
<tr>
<td>GetEvent</td>
<td>ECC1</td>
<td>0</td>
</tr>
<tr>
<td>WaitEvent with rescheduling</td>
<td>ECC1</td>
<td>32</td>
</tr>
<tr>
<td>SendMessage</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>InitCounter</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>CounterTrigger with rescheduling</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>GetCounterValue</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>GetCounterInfo</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>GetAlarmBase</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>SetRelAlarm</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td>BCC1</td>
<td>32</td>
</tr>
<tr>
<td>CancelAlarm</td>
<td>BCC1</td>
<td>16</td>
</tr>
<tr>
<td>GetAlarm</td>
<td>BCC1</td>
<td>0</td>
</tr>
<tr>
<td>StartTimeScale with rescheduling</td>
<td>BCC1</td>
<td>80</td>
</tr>
<tr>
<td>StopTimeScale</td>
<td>BCC1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 14.2 Stack Usage for Empty Tasks/ISRs**

<table>
<thead>
<tr>
<th>Operation</th>
<th>CC</th>
<th>Used bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty BASIC task</td>
<td>BCC1 FastTerminate</td>
<td>80</td>
</tr>
<tr>
<td>EXTENDED task with WaitEvent inside endless loop</td>
<td>ECC1</td>
<td>54</td>
</tr>
<tr>
<td>Empty ISR category 1</td>
<td>BCC1, ECC1</td>
<td>48</td>
</tr>
<tr>
<td>ISR category 2, Task stack</td>
<td>ECC1</td>
<td>96</td>
</tr>
<tr>
<td>Empty ISR category 2, ISR stack,</td>
<td>ECC1</td>
<td>24</td>
</tr>
<tr>
<td>SystemTimer ISR, Alarm activated (ISR Stack)</td>
<td>ECC1</td>
<td>24</td>
</tr>
</tbody>
</table>

* Including ActivateTask stack usage
Features

Programming Model

MPC55xx can operate at either of two privilege levels. Supervisor level is more privileged than user level.

OSEK OS uses supervisor programming model, so start-up code should provide correct initialization. OSEK OS always works in the big-endian byte ordering mode.

Target Hardware Initialization

OSEK OS does not provide target hardware (except used by system timers) initialization possibilities, therefore target hardware initialization (memory used, PowerPC cache settings, etc.) should be performed before system startup. The OSEKturbo OS/MPC55xx package includes the files containing some hardware initializing code located in the HWSPEC directory: These files are the example of the HW initialization for work with MPC5554.

Interrupt Vector Table

The interrupt vector table is defined in the file “vector.c” which is delivered with the OSEK Operating System and located in the HWSPEC directory. This file contains entries for System and Second Timers ISRs and for the external ISR.

OS/MPC55xx provides the external interrupt (sharing INVOR4 interrupt vector) distinction and the appropriate interrupt handler call. In this case OS/MPC55xx snaps up the external interrupt exception, makes required save/restore operations and calls appropriated interrupt handler.

The user should copy this file into the project directory and then modify it as needed. This modification can be necessary in case when the users uses own ISR for DEC or FIT interrupts and “Onelevel” mode for the interrupt dispatcher. In this case the user has to place his interrupt handler on the appripriated position in the “vector.c” file.

For More Information: www.freescale.com
Let’s assume, that ISR is defined via the following statement in configuration OIL-file:

ISR <nameISR> {
    I rqChannel = DEC;
    ... // ISR category should be defined here
};

In this case, the string “// ba <Users's ISR for the DEC interrupt>” in the “vector.c” file shall be replaced onto the following: “ba <nameISR>”

And ISR handler definition should look like the following in the application source file:

ISR( <nameISR> ) {
    ... // ISR body should be defined here
};

The user shouldn’t remove references to System and Second Timer ISR’s if they are used by application.

Please see vector.c located in HWSPEC directory for details.

**Low-Power Mode**

The user can configure OSEK OS to use low-power mode when system runs idle – HCLowPower configuration option (see “Global System Attributes”). If the HCLowPower is set to TRUE the system enters low power mode in idle loop (when there is no running task).

The user is able to implement his own low-power mode logic in the framework of IdleLoopHook. In this case it is recommended to set HCLowPower to FALSE. (see “Hook Routines Related Attributes”).

**MSR Bits Manipulation**

The user should not manipulate with MSR general-purpose bits like FP, ME, FE0, FE1, IP during run-time. If it is necessary to set MSR ME bit (for example), then it should be set before OS start-up.

For More Information: www.freescale.com
Implementation Background

OSEK OS task context contains all information needed to provide both ordinary task scheduling and scheduling after ISR.

Using Floating Point

PowerPC Book E floating-point instructions are not supported in MPC5500 hardware therefore the OSEK OS/MPC55XX does not support floating point operations. Software floating port (compiler support and libraries) may be used as usual.

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

OSEKturbo OS/MPC55xx provides two levels of the interrupt dispatching:

- OneLevel, when one interrupt level for external interrupts is supported
- MultiLevel, when multilevel interrupt emulation mode is supported

NOTE
There are two interrupt levels even in case of OneLevel Interrupt dispatcher: maskable and unmaskable interrupts. The words “interrupts level” means maskable external interrupts level.

External Interrupts Processing Background

One Level Interrupt Dispatcher

When an interrupt(s) occur Interrupt Dispatcher in the One Level mode calls the interrupt handler which corresponds to the interrupt

For More Information: www.freescale.com
of highest level among pending interrupts. All interrupts are disabled during interrupt processing.

**Multilevel Interrupt Dispatcher**

OSEKturbo OS/MPC55xx supports multilevel interrupts provided by the hardware. *Multilevel Interrupt Dispatcher* works as one level dispatcher but interrupts are processed in accordance with its priority, so interrupts of higher levels are enabled while interrupts of the same and lower levels are disabled when the ISR code is executed.

**Stack Definition**

MPC r1 register (stack pointer) should always contain word-aligned address, therefore all task and ISR stack sizes in OSEK OS application should be word-aligned (should be multiple of 4).

**Timer Hardware**

The special OS attributes are introduced to define hardware interrupt source and desired parameters for counter considered to be System (Second) Timer.

Note that counter assigned to System (Second) Timer uses interrupts from corresponding timer channel. At run time OSEKturbo OS enables CPU interrupts and timers interrupts corresponding to System and Second Timers. User shall not directly manipulate with System/Second timer hardware when OSEK is running. However the timers hardware may be initialized prior to calling to StartOS in main function. In this case the *Prescaler* attribute may be set to *USER* thus disabling timer prescaler reinitialization at OSEK 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.

The following statement should be used for system timer definition:

```c
OS <name> {
...
    SysTimer = <SWCOUNTER / HW COUNTER>;
}
```
This OSEK OS version contains system timer code developed for MPC5500 target. The following hardware sources can be used for System and/or Second Timers:

- DEC (Decrementer Counter);
- FIT (Fixed Interval Timer)

### DEC-based System Timer

The decrementer, a counter that is updated at the same rate as the TB, provides a means of signaling an exception after a specified amount of time has elapsed.

The DEC timer is used as the Hardware timer.

The `ISRPRIORITY` attribute specifies the periodic interrupt request priority. This attribute shall have the value from the range 1..15.

### FIT-based System(Second) Timer

The fixed-interval timer is essentially a selected bit of the TB, which provides a means of signalling an exception whenever the selected bit transitions from 0 to 1, in a repetitive fashion. The fixed-interval timer is typically used to trigger periodic system maintenance functions.

The FIT timer is used as the Software timer.

The `ISRPRIORITY` attribute specifies the periodic interrupt request priority. This attribute shall have the value from the range 1..15.

The `TimerModuloValue` attribute provides the number of bit in the 64-bit timer that is used as modulo; bit 0 is the most significant. The period of timer is equal to $2^{(63-TimerModuloValue)}$ ticks of the FIT input frequency.
Example of Timers Configuration

For example, to configure a system timer as a Hardware counter and the second timer as a Software counter on an MPC5500 the following attributes may be set in the OIL file:

OS OsName {
...  
  SysTimer = HWCOUNTER {
    COUNTER = CounterName1;
    ISRPRIOITY = 10;
    TimerHardware = DEC;
  };
...  
...  
  SecondTimer = SWCOUNTER{
    COUNTER = CounterName2;
    ISRPRIOITY = 2;
    TimerHardware = FIT{
      TimerModuloValue = 51;
    };
  };
...  
};

OS/MPC55xx does not control the IP bus clock frequency therefore the system hardware timer Period is not configurable.

For the SecondTimer the TimerModuloValue = 51 causes an interrupt to occur every \(2^{(63-51)} = 4096\) Time Base ticks.
Application Troubleshooting

In this chapter some advice is given which may be useful for developers working with the OSEK Operating System.

This chapter consists of the following sections:

- System Generation
- Stack Errors
- Unregistered External Interrupts
- Known Problems

System Generation

The System Generator is used to generate the code for the OSEK Operating System 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.

Stack Errors

Stack errors may be due to the stack pointer being incorrect or to stack content being corrupted. Stack content problems are possible if pointers are used to access stack variables, but stack pointer problems seem to be more common. The symptom of either problem is usually a task or ISR executing normally, but then when
a return is performed, the program executes at some incorrect address.

Tasks should have enough stack for their execution, therefore it is recommended to pay attention in the task definition statements to providing each task with a needed amount of stack. See “PowerPC Platform-Specific Features”.

NOTE The optional stack overflow checking during run time are implemented in OSEKturbo OS for debugging application. See “Stack Overflow Checking”.

Unregistered External Interrupts

If interrupt dispatcher is used then external interrupts shall be registered in the system that is they shall be defined in the application oil file. System will stop on the 0xYYY1000 vector if unregistered interrupt occurs. OSEK OS does not support reset of all interrupt sources during initialization.

Known Problems

Troubleshooting

Problem A: Error while running SETUP.EXE from a network drive.

Resolving: Start SETUP.EXE from local drive.

Problem B: Installation program produces error message: “Cannot detect installation media. Installation failed”.

Reason: Installation files were copied into NTFS partition, archive file attribute flags are cleared for some of them and compress file attribute flags are set for some of them.

Workaround: Set archive file attribute flag for all the installation files.

Problem C: Development software does not work under MS Windows.

For More Information: www.freescale.com
Reason 1: Inappropriate development hardware or software is used.

Reason 2: OSEK OS/mpc55xx software is installed into the directory with spaces (like "C:\Program Files").

Workaround 1: Install the software into the directory without spaces (for example: "C:\metrowerks\osek").

Reason 3: Environment variables OSEKDIR, CWDIR are not correct.

Workaround 1: Set correct environment variables.

Problem D: The NMAKE or the MAKE generates a wrong output.

Reason 1: The needed directories and files are not installed properly.

Resolving 1: Check the integrity and consistency of the package using FILELIST.TXT.

Reason 2: The needed software is not installed or installed improperly.

Resolving 1: Check the installed software (listed above).

Resolving 2: Check the correctness of the software installation (environment variables, etc.)

Problem E: Shared files are not removed from BIN directory during uninstallation of OSEK OS.

Reason: Current version of uninstall program (file IsUninst.exe ver. 5.10.145.0 from Windows NT Service Pack 4) processes shared files with error.

Resolving: Get new version of IsUninst.exe program.

Workaround: Remove shared files manually after uninstallation.

Problem F: Error "The parameter is incorrect" during installation.

Reason: Too long full name (include parent directories) of the directory with installation media.

Workaround: Move installation media to directory with shorter full path.

Problem G: Some icon is not created in Start menu after installation.

Reason: There is not enough space on target drive.

For More Information: www.freescale.com
Workaround: Reinstall OSEK/OS to other drive.
System Services

This chapter provides a detailed description for all OSEK Operating System run-time services, with appropriate examples.

This chapter consists of the following sections:

- General
- Task Management Services
- ISR Management Services
- Resource Management Services
- Event Management Services
- Counter Management Services
- Alarm Management Services
- Communication Management Services
- Debugging Services
- Operating System Execution Control

General

This chapter provides detailed description of all OSEK 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:

**System Services**

**Task Management Services**

**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.

**Task Management Services**

**Data Types**

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

- **TaskType** – the abstract data type for task identification

For More Information: www.freescale.com
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 OSEK 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 OSEK OS to indicate task:

- INVALID_TASK – constant of data type TaskType for undefined task

Conventions

Within an application of the OSEK 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 previous OSEK versions. It may be omitted in application code.

Syntax: DeclareTask( <name of task> );
Input: <name of task> – a 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.
Conformance: BCC1, ECC1

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.
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_OSRESOURCE – the task still occupies resources.
  - E_OSCALLEVEL – a call at the interrupt level.

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_RESOURCE – the calling task still occupies resources.
  - E_OS_CallLevel – a call at the interrupt level.

Conformance: BCC1, ECC1

Schedule

Syntax: StatusType Schedule( void );

Input: None.

Output: None.
System Services
Task Management Services

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-preemptable 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_CALLEVEL – a call at the interrupt level.
  – E_OSRESOURCE - calling task occupies resources.

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.
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:
  - None.
Examples for Task Management Services

The example below shows the use of all OSEK OS task management services. PreTaskHook is used to count switches to tasks. Three tasks are declared in the OIL configuration file:

```c
...  
  OS os1{
    ...
    PRETASKHOOK = TRUE;
  };
...
  TASK TaskA {
    PRIORITY = 3;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
  };
  TASK TaskB {
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = EventTaskB;
  };
  TASK TaskC {
    PRIORITY = 1;
    SCHEDULE = NON;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = EventTaskC;
  };
...  
  int CntTaskA;
```
int CntTaskB;
int CntTaskC;

void PreTaskHook( void );
{
    GetTaskId( &task );

    switch( task ) /* increment corresponding counter */
    {
        case TaskA:
            CntTaskA++; break;
        case TaskB:
            CntTaskB++; break;
        case TaskC:
            CntTaskC++; break;
    }
}

TASK( TaskA )
{
    TaskType task;
    ... /* any user’s code */

    ActivateTask( TaskB ); /* activate TaskB */
    ... /* any user’s code */
    ChainTask( TaskC );
}

TASK( TaskB )
{
    TaskStateType state;
    ... /* any user code */

    GetTaskState( TaskC, &state ); /* check the state of TaskC */
    switch( state ) /* and perform appropriate actions */
    {
        case READY: break;
        case WAITING: SetEvent( TaskC, EventTaskC );
                        break;
        case SUSPENDED: ChainTask( TaskC );
                        break;
    }
}
ISR Management Services

Data Types
No special data types are defined for the OSEK interrupt handling functionality.

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

ISR( <name of ISR>)
{
... /* any user code */
... /* any user code */
}
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 previous OSEK versions. It may be useful in vector file if it does not include OS configuration file.

**Syntax:**
```c
DeclareISR( <name of ISR> );
```

**Input:**
`<name of ISR>` – a reference to the ISR.

**Description:**
This statement declares ISR function.

**Conformance:**
BCC1, ECC1

---

**EnableAllInterrupts**

**Syntax:**
```c
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 is intended to control only the “EE” bit in the Machine State Register (MSR). This service does not support nesting.

---

\(^1\) This declaration is not defined by OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.
System Services
ISR Management Services

Status:  
  • Standard:  
    – None.  
  • Extended:  
    – None.

Conformance:  BCC1, ECC1

**DisableAllInterrupts**

Syntax:  
  void DisableAllInterrupts ( void );

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 is intended to control only the “EE” bit in the Machine State Register (MSR).

This service does not support nesting.

Status:  
  • Standard:  
    – None.
  • Extended:  
    – None.

Conformance:  BCC1, ECC1

**ResumeAllInterrupts**

Syntax:  
  void ResumeAllInterrupts ( void );

Input:  None.

Output:  None.
**System Services**

**ISR Management Services**

**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 `ErrorHook` is defined and `STATUS` is set to EXTENDED this service calls ErrorHook with E_OS_SYS_ORDER status 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:**

```c
void SuspendAllInterrupts ( void );
```

**Input:** None.

**Output:** None.

---

1. this is OSEKturbo extension of OSEK OS
**System Services**

**ISR Management Services**

**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 OSEKturbo dispatcher calls\(^1\) ErrorHook, if it is defined, with parametr E_OS_SYS_ORDER.

**Status:**
- **Standard:**
  - None.
- **Extended:**
  - None.

**Conformance:**
BCC1, ECC1

---

**ResumeOSInterrupts**

**Syntax:**
```c
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

---

\(^1\) this is OSEKturbo extension of OSEK OS
SuspendOSInterruptions/ResumeOSInterruptions 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 SuspendOSInterruptions and ResumeOSInterruptions the interrupt recognition status saved by the first call of SuspendOSInterruptions is restored by the last call of the ResumeOSInterruptions service.

If ErrorHook is defined and STATUS is set to EXTENDED this service calls1 ErrorHook with E_OS_SYS_ORDER status 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

SuspendOSInterruptions

Syntax: void SuspendOSInterruptions ( 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 ResumeOSInterruptions service. No API service calls beside SuspendAllInterrupts/ResumeAllInterrupts pairs and

---

1. this is OSEKturbo extension of OSEK OS
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 OSEKturbo dispatcher calls ErrorHook, if it is defined, with parametr E_OS_SYS_ORDER.

If no ISRs of category 2 are defined, then this service does nothing.

Status:
• Standard:
  – None.
• Extended:
  – None.

Conformance: BCC1, ECC1

Examples for Interrupt Management Services

Below examples for ISR category 1 and 2 are presented.

The following definitions are made in the definition file:

```c
...
OS myOS {
  ...
  IsrStackSize = 256;
  MessageCopyAllocation = OS;
  ...
};
TASK TaskB {
  PRIORITY = 2;
  SCHEDULE = FULL;
}
```

1. this is OSEKturbo extension of OSEK OS
AUTOSTART = FALSE;
ACTIVATION = 1;
STACKSIZE = 256;
EVENT = EventTaskB;
}

TASK IndTask {
PRIORITY = 1;
SCHEDULE = FULL;
AUTOSTART = TRUE;
ACTIVATION = 1;
}

COUNTER Ctr1 {
MINCYCLE = 1;
MAXALLOWEDVALUE = 24;
TICKSPERBASE = 1;
}

MESSAGE Temp {
TYPE = UNQUEUED;
CDATATYPE = "char";
ACTION = NONE;
}

MESSAGE Wrn {
TYPE = UNQUEUED;
CDATATYPE = "MSGCTYPE";
ACTION = NONE;
}

ISR ISR1_handler {
CATEGORY = 1;
PRIORITY = 8;
}

ISR ISR2_handler {
CATEGORY = 2;
PRIORITY = 6;
}

... 

The C-language code is the following:

cchar CREG, DREG;
cchar data1;
...
System Services
Resource Management Services

/* ISR category 1: */
ISR( ISR1_handler )
{
    if( CREG != 0xC0 ) CREG |= 0x40;
    else CREG |= 0x03;
    DREG = data1;
}
TaskStateType stateB;
...
/* ISR category 2: */
ISR( ISR2_handler )
{
    CounterTrigger( Ctrl1 );
    GetTaskState( TaskB, &stateB );
    if( stateB == SUSPENDED ) ActivateTask( TaskB );
}

Resource Management Services

Data Types
The OSEK 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 previous OSEK versions. It may be omitted in application code.

For More Information: www.freescale.com
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.

Conformance: BCC1, ECC1

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 OSEKturbo extension of OSEK 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. The code supporting scheduler resource is excluded from application, if ResourceScheduler is defined as FALSE in OIL.

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.

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 OSEKturbo extension of OSEK OS because OSEK/VDX specifies full support only beginning 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. The code supporting scheduler resource is excluded from application, if scheduler resource is not defined on OIL.

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.

Conformance: BCC1, ECC1

Examples of Using Resources

The example below presents resource management directives.

The following definitions are made in the definition file:

```c

TASK TaskA {
  PRIORITY = 1;
  SCHEDULE = FULL;
  AUTOSTART = TRUE;
  ACTIVATION = 1;
  STACKSIZE = 256;
  EVENT = EventTaskA;
  RESOURCE = SCI_res;
  RESOURCE = TASKB_res;
};
TASK TaskB {
  PRIORITY = 2;
  SCHEDULE = FULL;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  RESOURCE = TASKB_res;
};
TASK TaskC {
  PRIORITY = 3;
  SCHEDULE = FULL;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
}
```

For More Information: www.freescale.com
The C-language code is the following:

```c
TASK( TaskA )
{
... /* user’s code */
    /* occupy the SCI resource to disable SCI_handler */
    GetResource( SCI_res );
... /* user’s code which configures SCI */
    /* release the SCI resource to enable SCI_handler */
    ReleaseResource( SCI_res );
...
    /* occupy the resource to avoid starting TaskB */
    GetResource( TASKB_res );
    ActivateTask( TaskB );
    /* occupy the scheduler to disable rescheduling */
    GetResource( RES_SCHEDULER );
    ActivateTask( TaskC );
... /* user’s code */
    /* release the scheduler resource */
    ReleaseResource( RES_SCHEDULER );
    /* TaskC is started here */
    ... /* user’s code */
    /* release the TaskB resource */
    ReleaseResource( TASKB_res );
    /* TaskB is started here */
    ... /* user’s code */
    TerminateTask();
```
Event Management Services

Data Types

The OSEK 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

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

Event Declaration

The declaration statement DeclareEvent may be used for compatibility with previous 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.

Conformance: ECC1

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.
          – E_OS_STATE – the referenced task is in the suspended state.

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 – a call at the interrupt level is not allowed.

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.
  - E_OS_STATE – the referenced task is in the suspended state.

Conformance: ECC1

**WaitEvent**

**Syntax:**

```c
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.
**Examples of Using Events**

The example below shows how events can be used in the OSEK Operating System.

The following definitions are made in the definition file:

```c
TASK TASK_A {
    PRIORITY = 3;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = ExtEvent1;
    EVENT = ExtEvent2;
    EVENT = XEvent;
    EVENT = YEvent;
    EVENT = Z1_FLG;
    EVENT = Z2_FLG;
};
TASK TASK_B {
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = DgrAlmEvent;
    EVENT = ExtEvent2;
};
TASK TASK_C {
    PRIORITY = 1;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
};
```
ACTIVATION = 1;
};
COUNTER DgrCnt {
    MAXALLOWEDVALUE = 150;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
};
ALARM AWAKE {
    COUNTER = DgrCnt;
    ACTION = SETEVENT {
        TASK = TASK_B;
        EVENT = DgrAlmEvent;
    };
};
/* 'external' events for TASK_A */
EVENT ExtEvent1 { MASK = 0x02 };
/* 'internal' events for TASK_A */
EVENT XEvent { MASK = 0x80 };
EVENT YEvent { MASK = 0x40 };
EVENT Z1_FLG { MASK = 0x20 };
EVENT Z2_FLG { MASK = 0x10 };
/* event for TASK_B */
EVENT DgrAlmEvent { MASK = 0x01 };
/* this EVENT object defines two different */
/* events for TASK_A and TASK_B (see references */
/* in the TASK objects), but these events */
/* have one and the same mask */
EVENT ExtEvent2 { MASK = 0x04 };

MESSAGE Norm {
    TYPE = UNQUEUED;
    CDATATYPE = "int";
};
...

The C-language file:

TASK( TASK_A ) /* Extended task TASK_A */
{
    /* 'external' events ExtEvent1 and ExtEvent2 */
    /* aa is an 'OR' of the masks of these events */
    EventMaskType aa = (ExtEvent1 | ExtEvent2);
System Services
Event Management Services

/* ‘internal’ events (flags) */
EventMaskType x, z1 = Z1_FLG, z2 = Z2_FLG;
int speed;
...
  /* Check the variable and set internal flag if needed */
  if (speed == LIMIT)
    { 
      x = XEvent;
      SetEvent( TASK_A, x );
    }
...
GetEventMask( TASK_A, &x ); /* check ‘internal’ flag */
  /* Perform some actions in accordance with */
  /* internal flag status */
  if ( (x & XEvent) != 0 ) ClearEvent( z1 );
  else SetEvent( TASK_A, z2 );
  if ( (x & YEvent) == 0 ) ChainTask( TASK_C );
...
  /* the task is stopped until one of three */
  /* ‘external’ events is set by another task */
WaitEvent( aa );
  /* clear all ‘external’ events after awakening */
ClearEvent( aa );
...

TASK( TASK_B ) /* Extended task TASK_B */
{
  EventMaskType b_ev, a_ev;
  b_ev = DgrAlmEvent | ExtEvent2;
InitCounter( DgrCnt, 0 ); /* initialize the counter */
...
  /* this alarm will awake this task */
SetRelAlarm( AWAKE, 20, 0 );
WaitEvent( b_ev ); /* waiting for one of two events */

/* The task will be ready again when one of two */
/* events are set. One of them – DgrAlmEvent will */
/* be set by the alarm AWAKE after 20 ticks of the */
/* counter DgrCnt. Thus, the task can delay itself. */
ClearEvent( b_ev ); /* clear all events */
GetEvent( TASK_A, &a_ev ); /* get events of TASK_A */
if ( (a_ev & ExtEvent1) == 0)
{
    a_ev = ExtEvent2;
    SetEvent( TASK_A, a_ev );
} /* set the event ExtEvent2 for TASK_A */
...
}

TASK( TASK_C ) /* Basic task TASK_C */
{
EventMaskType bb, set;
set = ExtEvent2;
...
GetEvent( TASK_B, &bb );
    /* if the event ExtEvent2 for TASK_B */
    /* is clear, set it */
if ( (bb & ExtEvent2) == 0 ) SetEvent( TASK_B, set );
...
}

Counter Management Services

Data Types and Identifiers

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

- **TickType** – the data type represents count value in ticks
- **TickRefType** – the data type of a pointer to the variable of data type **TickType**
- **CtrRefType** – 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:
System Services
Counter Management Services

- `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 `CtrInfoType` structure have the data type `TickType`, and the structure looks like the following:

```c
typedef CtrInfoType tagCIT;
struct tagCIT
{
    TickType maxallowedvalue;
    TickType ticksperbase;
    TickType mincycle;
};
```

NOTE CtrRefType, CtrInfoType and CtrInfoRefType data types are not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. These are OSEKturbo extension of OSEK 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>`.
- **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** / **OSMAXALLOWEDVALUE**
  maximum possible allowed value of the system/second timer in ticks (see also “Counter Definition”)

- **OSTICKSPERBASE** / **OSTICKSPERBASE**
  number of ticks required to reach a counter-specific value in the system/second counter (see also “Counter Definition”)

- **OSTICKDURATION** / **OSTICKDURATION**
  duration of a tick of the system/second counter in nanoseconds (defined automatically by System Generator utility (see also “CPU Related Attributes”)

- **OSMINCYCLE** / **OSMINCYCLE**
  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 OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. These are OSEKturbo extension of OSEK OS.

**Counter Declaration**

The declaration statement DeclareCounter 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> – a 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.

**Conformance:**

BCC1, ECC1

For More Information: www.freescale.com
InitCounter

Syntax: StatusType InitCounter( CtrRefType <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 CounterTrigger. 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.

The service is not implemented if no counters are defined in the configuration file.

The InitCounter service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

Status:  • Standard:
         – E_OK – no error.

         • Extended:
         – E_OS_ID – the counter identifier is invalid.
         – E_OS_VALUE – the counter initialization value exceeds the maximum admissible value.
         – E_OS_CALLEVEL – a call at interrupt level (not allowed).

Conformance: BCC1, ECC1

CounterTrigger

Syntax: StatusType CounterTrigger( CtrRefType <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.

This service may be used only on software counters. It has no sense to use this service on a hardware counter.

This service is not implemented if no counters are defined in the configuration file.

The CounterTrigger service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

Status: • Standard:
  – E_OK – no error.
• Extended:
  – E_OS_ID – the counter identifier is invalid or belongs to hardware counter.

Conformance: BCC1, ECC1

GetCounterValue

Syntax: StatusType GetCounterValue( CtrRefType <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 system 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 level, ISR level and in ErrorHook, PreTaskHook and PostTaskHook hook routines.

This service is not implemented if no counters are defined in the configuration file.

The GetCounterValue service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

Status:

- Standard:
  - E_OK – no error.
- Extended:
  - E_OS_ID – the counter identifier is invalid.

Conformance: BCC1, ECC1

GetCounterInfo

Syntax: StatusType GetCounterInfo( CtrRefType <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 OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.
Status:

- Standard:
  - E_OK – no error.

- Extended:
  - E_OS_ID – the counter identifier is invalid.

Conformance: BCC1, ECC1

**Examples for Counter Management**

The example shows how CounterTrigger, GetCounterValue services can be used.

The following definitions are made in the definition file:

```c
...  
COUNTER FirstCnt {
    MAXALLOWEDVALUE = 127;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
};
COUNTER SecondCnt {
    MAXALLOWEDVALUE = 36;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
};
ISR First_Handler {
    CATEGORY = 2;
    PRIORITY = 1;
};
TASK TaskCnt {
    PRIORITY = 1;
    SCHEDULER = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
};
...
```

Following C-language code shows how CounterTrigger can be used to increment counter:
ISR First_Handler {
  ... /* User's code */
  CounterTrigger( FirstCnt );
  ... /* User's code */
}

TASK TaskCnt {
  TickType cnt1;
  ... /* User's code */
  CounterTrigger( FirstCnt );
  GetCounterValue( FirstCnt, &cnt1 );
  if( cnt1 == 0 )
    { CounterTrigger( SecondCnt ); }
  ... /* User's code */
}

Alarm Management Services

Data Types and Identifiers

The following data types are established by OSEK 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

*OSMICNCYCLE* – 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>` – a 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.

**Conformance:**

BCC1, ECC1

### 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.

---

For More Information: www.freescale.com
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.

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:
SetRelAlarm

Syntax:  

```c
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.
- Extended:
  - E_OS_ID – the alarm identifier is invalid.
- **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).

Conformance:  
- BCC1, ECC1.

### SetAbsAlarm

**Syntax:**

```c
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.

---

For More Information: [www.freescale.com](http://www.freescale.com)
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_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).

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.

Conformance: BCC1, ECC1
StartTimeScale

Syntax: `void StartTimeScale ( void );`

Input: None.

Output: None.

Description: The service starts a TimeScale processing. The first Task in TimeScale is activated immediately, the subsequent tasks are activated according with StepTime and StepNumber values.

Particularities: Allowed on task level only.

The `StartTimeScale` service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

Status:  
• Standard:  
  – None.
• Extended:  
  – None.

Conformance: BCC1, ECC1

StopTimeScale

Syntax: `void StopTimeScale ( void );`

Input: None.

Output: None.

Description: The service cancels a TimeScale processing by disabling interrupts from the system timer.

Particularities: Allowed on task level, on ISR level and in all hook routines.

The `StopTimeScale` service is not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.

Status:  
• Standard:  
  – None.
• Extended:  

For More Information: www.freescale.com
Examples for Alarm Management

The example shows how counters and alarms can be used. For examples of TimeScale using see OSEKturbo User’s Guide, 3.8 TimeScale. Also the sample application can be used as an example for TimeScale.

The following definitions are made in the definition file:

```
TASK TaskTime {
    PRIORITY = 3;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
};
TASK TASK_B {
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = DgrAlmEvent;
};
TASK TASK_X {
    PRIORITY = 1;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
};
COUNTER TimeCnt {
    MAXALLOWEDVALUE = 127;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
};
COUNTER DgrCnt {
    MAXALLOWEDVALUE = 36;
    TICKSPERBASE = 1;
    MINCYCLE = 1;
}
```
The alarm TimeAlm activates the task TASK_X when the counter TimeCnt expires. The alarm DgrAlm sets the specified event for the task TASK_B when the counter DgrCnt expires.

The C-language code is the following:

```c
OSMSGNorm _Norm;

TASK( TaskTime )
{
    TickType curTime;
    TickType ticksToExpire;
    OSBYTE i=0;

    InitCounter( TimeCnt, 0 ); /* init time counter with 0 value */
    CounterTrigger( TimeCnt ); /* increments counter */

    while (i != 1) {
```

For More Information: www.freescale.com
/ * read TimeCnt value */
GetCounterValue( TimeCnt, &curTime );
if( curTime == CONST )
{ /* if desired value, activate TaskB */
    ActivateTask( TASK_B );
    SetRelAlarm( TimeAlm, 20, 0 );
    /* activate TaskX when TimeCnt reaches */
    GetAlarm( TimeAlm, &ticksToExpire );
    /* just for example: TimeAlm will */
    /* expire after 'ticksToExpire' ticks of TimeCnt */
}
/* if more than desired value, terminate the task */
if( curTime > CONST ) TerminateTask();
}

TASK( Task_B )
{
OSMSGNorm _Norm;
EventMaskType evMask;

    evMask = DgrAlmEvent;
    /* init degree counter with 0 value */
    InitCounter( DgrCnt, 0 );
    SetAbsAlarm( DgrAlm, 75, 15 ); /* set cyclic alarm */
    WaitEvent( evMask );
    /* wait for event which must be set by the alarm */
    _Norm = 1;  /* wake up and send the message that all is OK */
    SendMesssage( Norm, _Norm);
    TerminateTask();
}

ISR( Timer_Isr )
{
    ... /* reset the hardware */
    /* increment the counter and process */
    /* alarms attached to the counter */
    CounterTrigger( TimeCnt );
}

ISR( Dgr_Isr )
{
Communication Management Services

Data Types and Identifiers

The following names are used in the OSEK Operating System to operate with messages:

- **SymbolicName** – a unique name representing a message. It only can be used in conjunction with calls of the message service. A SymbolicName need not be a data type. Variables or constants of SymbolicName can be declared or used

- **AccessName** – a unique name defining access to a message object. Depending on the chosen configuration, a distinction is made between the following AccessName scheme:
  
  **WithCopy** configuration:
  
  An application variable exists as a local copy of the message. The name of the variable is the AccessName. This variable contains a copy of the corresponding message object

  **WithoutCopy** configuration:

  The message object data is accessed via the AccessName. This AccessName is a static link: it refers directly to the message object data. The AccessName refers to the same data (RAM) as the message object

- **AccessNameRef** – the address of the message data field

- **FlagType** – the abstract data type for flag identification

SendMessage

**Syntax:**

```
StatusType SendMessage( SymbolicName <Message>,
    AccessNameRef <Data> );
```

**Input:**

- `<Message>` – symbolic name of the message object.
<Data> – the reference to the message data to be sent.

Output: None.

Description: **Unqueued and Queued WithCopy message:**

This service updates the message object <Message> with the Data given by <Data> and requests the transmission of the message object to the receiver. The message object is locked during this update and during the transmission of the data to the underlying communication layer. The message object is not updated if it is already locked. This service updates also the status information of the message object accordingly.

**Unqueued WithoutCopy message:**

This service requests the transmission of the already updated message object to the receiver.

Particularities: Allowed for all message types on task level and in message’s callback function. Unqueued messages in case of WithCopy only are allowed on ISR level and in ErrorHook routine.

SendMessage does not verify whether the message object has been initialized prior to sending it.

This service is not implemented if no messages are defined in the configuration file.

Status: • Standard:
  – E_OK – no error.
  – E_COM_LOCKED – message object locked (other task or ISR is sending or receiving the same message or this message resource is occupied).
  – E_OS_CALLEVEL - call at ISR level for queued message or with <Data> configured WithOutCopy; this status is OSEKturbo extension of OSEK OS.

• Extended:
  – E_COM_ID – invalid parameter <Message> or sending message other than unqueued one in WithCopy configuration from ISR level.

Conformance: BCC1, ECC1
ReceiveMessage

Syntax: StatusType ReceiveMessage ( SymbolicName <Message>, AccessNameRef <Data>);

Input: <Message> – symbolic name of the message object.

Output: <Data> – the reference to the message data to be received.

Description: Unqueued and Queued WithCopy message:

This service delivers the message data associated with the message object <Message> to the applications message copy referenced by <Data>. The message object is locked during data reception. This service also updates the status information of the message object accordingly.

Unqueued WithoutCopy message:

Returns Status only since application can access directly to the message object.

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.

Allowed for all message types on task level and in message’s callback function. Unqueued messages in case of WithCopy only are allowed on ISR level and in ErrorHook routine.

Status:
- Standard:
  - E_OK – no error.
  - E_COM_LOCKED – message object locked (other task or ISR is sending or receiving the same message or this message resource is occupied).
  - E_COM_NOMSG - the queued message identified by <Message> is empty.
  - E_COM_LIMIT - an overflow of the FIFO of the queued message identified by <Message> occurred since the last call to ReceiveMessage for that particular <Message>.
– E_OS_CALLEVEL - call at ISR level for queued message
or with <Data> configured WithOutCopy; this status is
OSEKturbo extension of OSEK OS.

• Extended:
– E_COM_ID – invalid parameter <Message> or receiving
message other than unqueued one in WithCopy
configuration from ISR level).

Conformance: BCC1, ECC1

GetMessageResource

Syntax: StatusType GetMessageResource ( 
SymbolicName <Message> )

Input: <Message> – symbolic name of the message object.

Output: None.

Description: The service GetMessageResource set the message object <Message>
status as busy.

Particularities: It is recommended that corresponding calls to Get- and
ReleaseMessageResource should appear within the same function on
the same function level. Before terminating the task or entering the
wait state the corresponding service ReleaseMessageResource shall be
called by the application layer.

This service can only be used to support the transfer of a message
identified by <Message> whose copy configuration is WithoutCopy.

The service is not implemented if no messages are defined in the
configuration file or if USEMESSAGERESOURCE is set to FALSE.

Status: • Standard:
– E_OK - the message has been set to BUSY successfully.
– E_COM_LOCKED - the message is locked.
– E_COM_BUSY - the message is already set to BUSY.
– E_OS_NOFUNC - the message is queued; this status is
OSEKturbo extension of OSEK OS.

• Extended:
– E_COM_ID - the message parameter is invalid.

For More Information: www.freescale.com
**ReleaseMessageResource**

**Syntax:**
```
StatusType ReleaseMessageResource ( 
    SymbolicName <Message> )
```

**Input:**
`<Message>` – symbolic name of the message object.

**Output:**
None.

**Description:**
The service `ReleaseMessageResource` shall unconditionally set the message object `<Message>` to NOT_BUSY.

**Particularities:**
It is recommended that corresponding calls to `Get-` and `ReleaseMessageResource` appear within the same function on the same function level. Before terminating the task or entering the wait state the corresponding service `ReleaseMessageResource` shall be used.

This service can only be used to support the transfer of a message identified by `<Message>` whose copy configuration is `WithoutCopy`.

The service is not implemented if no messages are defined in the configuration file or if `USEMESSAGERESOURCE` is set to FALSE.

**Status:**
- Standard:
  - E_OK - the message has been set to NOT_BUSY status.
- Extended:
  - E_COM_ID - the message parameter is invalid.
  - E_OS_NOFUNC - the message is queued; this status is OSEKturbo extension of OSEK OS.

**Conformance:**
BCC1, ECC1

---

**GetMessageStatus**

**Syntax:**
```
StatusType GetMessageStatus ( 
    SymbolicName <Message> )
```

**Input:**
`<Message>` – symbolic name of the message object.

**Output:**
None.
**Description:**
The service `GetMessageStatus` return the current status of the message object `<Message>`. If this service call fails, it return an implementation specific error code that shall be distinguishable from all other return values.

**Particularities:**
This service is not implemented if no messages are defined in the configuration file or if `USEMESSAGESTATUS` is set to FALSE.

**Status:**
- Standard:
  - `E_COM_LOCKED` - the message is locked.
  - `E_COM_BUSY` - the message is currently set to BUSY.
  - `E_COM_NOMSG` - the FIFO of the queued message identified by `<Message>` is empty.
  - `E_COM_LIMIT` - an overflow of the FIFO of the queued message identified by `<Message>` occurred.
  - `E_OK` - none of the above conditions is applicable and no indication of error is present.
- Extended:
  - `E_COM_ID` - the `<Message>` parameter is invalid.

**Conformance:**
BCC1, ECC1

---

**StartCOM**

**Syntax:**
```c
StatusType StartCOM (void);
```

**Input:**
None.

**Output:**
None.

**Description:**
This service calls message initialization routines.

**Particularities:**
This routine performs the initialization of the application specific message objects by calling the `MessageInit` function provided by the application programmer. `StartCOM` is called from within a task.

**Status:**
- `E_OK` – the initialization completed successfully.
- The service returns an implementation or application specific error code if the initialization did not complete successfully.

**Conformance:**
BCC1, ECC1

---

For More Information: www.freescale.com
**InitCOM**

Syntax: `StatusType InitCOM (void);`

Input: None.

Output: None.

Description: The service is intended for initialization of communication hardware.

Particularities: This service does nothing in OSEKturbo.

Status: – `E_OK` – the initialization completed successfully.

Conformance: BCC1, ECC1

**CloseCOM**

Syntax: `StatusType CloseCOM (void);`

Input: None.

Output: None.

Description: The service is intended for de-initialization of communication hardware.

Particularities: This service does nothing in OSEKturbo.

Status: – `E_OK` – the initialization completed successfully.

Conformance: BCC1, ECC1

**StopCOM**

Syntax: `StatusType StopCOM (Scalar <ShutdownMode>);`

Input: Input parameter is indifferent in case of local communication.

Output: None.

Description: This service is used to terminate a session of OSEK COM, release resources where applicable.


– `E_COM_BUSY` - application uses COM resources.
**MessageInit**

**Syntax:**
```
StatusType MessageInit (void);
```

**Input:** None.

**Output:** None.

**Description:** This routine initializes all application specific message objects.

**Particularities:** This function is provided by the application programmer and is called by the `StartCOM` routine only.

**Status:**
- `E_OK` – if the initialization of the application specific message object has completed successfully.
- The service returns an implementation or application specific error code if the initialization did not complete successfully.

**Conformance:** BCC1, ECC1

---

**ReadFlag**

**Syntax:**
```
FlagValue ReadFlag (FlagType <FlagName>);
```

**Input:** `<FlagName>` – message flag name.

**Output:** FlagValue – state of the flag `<FlagName>`.

**Description:** This service returns the value of the specified notification flag.

**Particularities:** This service returns 0 if the flag is not set, i.e. message has not arrived.

**Status:**
- Standard and Extended:
  - TRUE - the flag is set by arrived message.
  - FALSE - the flag is not set.

**Conformance:** BCC1, ECC1
ResetFlag

Syntax: StatusType ResetFlag (FlagType <FlagName>)

Input: <FlagName> – message flag name.

Output: None.

Description: This service sets the specified notification flag to FALSE.

Particularities: This service is provided to ensure flag data consistency thus enabling portable access to the message notification flag.

Status: • Standard and Extended:
   – E_OK - the flag reset completed successfully.

Conformance: BCC1, ECC1

Examples of Using Messages

The examples below present the usage of system services for communication. The following definitions are made in the definition file:

OS os1 {
...
MessageCopyAllocation = USER;
...}

TASK TASK_A {
  SCHEDULE = FULL;
  AUTOSTART = TRUE;
  ACTIVATION = 1;
  PRIORITY = 2;
  STACKSIZE = 256;
  EVENT = MYEVENTMSGA;
  ACCESSOR = RECEIVED {
    MESSAGE= MsgA;
    WITHOUTCOPY= FALSE;
    ACCESSNAME= "Msga";
  };
};

TASK TASK_B {
  SCHEDULE = FULL;
};
AUTOSTART = FALSE;
ACTIVATION = 1;
PRIORITY = 3;
STACKSIZE = 256;
ACCESSOR = RECEIVED {
    MESSAGE= MsgB;
    WITHOUTCOPY= FALSE;
    ACCESSNAME= "MessageBuffer";
};
);

TASK TASK_C {
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    PRIORITY = 1;
    STACKSIZE = 256;
    ACCESSOR = SENT {
        MESSAGE= MsgA;
        WITHOUTCOPY= TRUE;
        ACCESSNAME= "msgAbuf";
    };
    ACCESSOR = SENT {
        MESSAGE= MsgB;
        WITHOUTCOPY= FALSE;
        ACCESSNAME= "msgB";
    };
};

MESSAGE MsgA {
    TYPE = UNQUEUED;
    CDATATYPE = "MSGA";
    ACTION = SETEVENT{
        EVENT = MYEVENTMSGA;
        TASK  = TASKA;
    };
};

MESSAGE MsgB {
    TYPE = QUEUED {
        QUEUEDEPTH = 5;
    };
    CDATATYPE = "MSGB";
    ACTION = ACTIVATETASK{
        TASK = TASK_B;
    };
}
EVENT MYEVENTMSGA {
    MASK = AUTO;
};
...

The C-language code is the following:

typedef struct{
    int  msg;
} MSGA;            /* data type for MsgA */

typedef struct{
    int  num;
    char data;
} MSGB;            /* data type for MsgA */

TASK( TASK_A )
{
    MSGA msgA;
    ...
    ClearEvent( MYEVENTMSGA );
    WaitEvent( MYEVENTMSGA );
    ReceiveMessage( MsgA, &msgA ); /* with copy */
        /* received data in msgA may be freely used */
    ...
}

TASK( TASK_B )
{
    int rnum;
    char rdata;
    MSGB msgB;
    ...
    ReceiveMessage( MsgB, &msgB ); /* receive with copy */
    rnum = msgB.num;                /* get data from message */
    rdata = msgB.data;
    ...
    TerminateTask();
}
int msgNum = 0; /* number for MsgB */

TASK( TASK_C )
{
    MSGB msgB;
    ...
    msgAbuf.msg = MyIntData; /* put some data into message */
    msgB.num = msgNum; /* put number into message copy */
    msgNum++;
    msgB.data = MyCharData; /* put data into message copy */
    SendMessage( MsgA, &msgAbuf ); /* w/o copy */
    SendMessage( MsgB, &msgB ); /* with copy */
    ...
}

Debugging Services

These services are not defined by OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK 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 because single stack is used.

Particularities: The service is implemented if the value of the StackOverflowCheck 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:**

```c
unsigned short GetStackUsage ( TaskType <TaskID> );
```

**Input:**

`<TaskID>` - a reference to the task.

**Output:**

- amount of stack used by task `<TaskID>` in bytes.
- 0xFFFF 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 `StackOverflowCheck` 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:**

```c
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 call is allowed on task level, ISR level and in `ErrorHook`, `PreTaskHook` and `PostTaskHook` hook routines.

Conformance: BCC1, ECC1

For More Information: www.freescale.com
Operating System Execution Control

Data Types

The OSEK 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 OSEK 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 OSEK 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_COM_ID**
- **E_COM_LOCKED**
- **E_COM_BUSY**
- **E_COM_NOMSG**
System Services
Operating System Execution Control

- E_COM_LIMIT
- E_OS_SYS_STACK
- E_OS_SYS_ORDER
- E_OS_SYS_MAINSTACK
- E_OS_SYS_ISRSTACK

GetActiveApplicationMode
Syntax: AppModeType GetActiveApplicationMode( void );
Input: None.
Output: Current application mode.
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 OSEKturbo this service does not returns to the caller.
Conformance: BCC1, ECC1

ShutdownOS
Syntax: void ShutdownOS( StatusType <Error> );

1 E_OS_SYS_xxx are not defined in the OSEK/VDX Operating System, v.2.2, 10 September 2001 specification. This is OSEKturbo extension of OSEK OS.
**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` never returns.

`ShutdownOS` runs in connection with the currently active context, which may be unknown to the user. Thus, no API functions are admitted within the `ShutdownHook` routine.

Allowed on task level, on ISR level and in `StartupHook` and `ErrorHook` hook routines.

**Conformance:** BCC1, ECC1

## Hook Routines

### ErrorHook

**Syntax:**

```c
void ErrorHook( StatusType <Error> );
```

**Input:** `<Error>` – a code of the error occurred.

**Output:** None.

**Description:** To implement `ErrorHook` the routine with name ‘ErrorHook’ shall be defined in 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. It is called before returning to the task level. 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.
System Services
Operating System Execution Control

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.

See “Hook Routines” for general description of hook routines.

This hook is not called if the system configuration option ERRORHOOK is set to FALSE.

Interrupts category 2 (and interrupts category 1 if their priority is not higher than any ISR category 2 priority) are disabled in ErrorHook.

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: Interrupts category 2 (and interrupts category 1 if their priority is not higher than any ISR category 2 priority) are disabled in PreTaskHook.

See “Hook Routines” for general description of hook routines.

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: Interrupts category 2 (and interrupts category 1 if their priority is not higher than any ISR category 2 priority) are disabled in PostTaskHook.

PostTaskHook is not called if running task is exist and OS is aborted by ShutdownOS service.

See “Hook Routines” for general description of hook routines.

This hook is not called if the system configuration option POSTTASKHOOK 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: Interrupts category 2 (and interrupts category 1 if their priority is not higher than any ISR category 2 priority) are disabled in StartupHook.

See “Hook Routines” for general description of hook routines.

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 shutdown. User can avoid return from the hook to calling level. For example reset signal can be generated in the hook.

Particularities: Interrupts category 2 (and interrupts category 1 if their priority is not higher than any ISR category 2 priority) are disabled in ShutdownHook.

See “Hook Routines” for general description of hook routines. This hook is not called if the system configuration option ShutdownHook is turned off in the configuration file.

Status: None.

Conformance: BCC1, ECC1

IdleLoopHook

Syntax: void IdleLoopHook( void );

Input: None.

Output: None.
Description: To implement \textit{IdleLoopHook} the routine with name ‘IdleLoopHook’ shall be defined in user’s code. This hook is called by the operating system from scheduler idle loop. It is not possible to call OSEK OS services from this hook. Hardware dependent code like manipulation with COP registers may be placed here.

Particularities: Interrupts inside this routine are enabled.

\textit{IdleLoopHook} is executed periodically if \textit{HCLowPower} is set to FALSE. Otherwise \textit{IdleLoopHook} is executed one time only and CPU transfers itself to the low power mode.

It is recommended to set \textit{HCLowPower} to FALSE if the user enters one of low power modes inside \textit{IdleLoopHook}.

See “Hook Routines” for general description of hook routines.

This hook is not called if the system configuration option \textit{IdleLoopHook} is turned off in the configuration file.

\textit{IdleLoopHook} hook is not defined in the \textit{OSEK/VDX Operating System, v.2.2, 10 September 2001} specification. This is OSEKturbo extension of OSEK OS.

Conformance: BCC1, ECC1
Debugging Application

This chapter provides information about OSEK OS feature intended for debugging a user application with OS.

This chapter consists of the following sections:

- General
- Using OS Extended Status for Debugging
- Context Switch Monitoring
- Stack Debugging Support

General

OSEK 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 OSEK services.

ORTI

The purpose of OSEK Run Time Interface (ORTI) implementation is giving the user extended opportunities in debugging embedded OSEK 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 OSEKturbo supports 2.1 ORTI version.

The ORTI shall be supported from both sides: an OSEK 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. Option -O of the SysGen defines the version of ORTI specification to be supported in the output ORTI file. 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 OSEK OS. This process is depicted on Figure 17.1.
Figure 17.1  Application Building Process with ORTI Support

OSEK Operating System components:
- Programs
- Library

Data files
- Third-party tools
Stack Debugging Support

To provide stack usage control the bottom and top stack labels for Extended Tasks are realized in OSEK OS. These labels can be seen in symbol debugger during application execution. They can be used for dynamic control of task stacks usage.

Using OS Extended Status for Debugging

It is strongly 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 OSEK 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 OSEK 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, OSEK OS system calls identification. This interface is turned on, if DEBUG_LEVEL = 2 or greater.

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.
  - ISR_LEVEL - OS works on ISR level, but nested interrupt was finished.
  - SYSTEM_TIMER - System Timer ISR was activated.
  - SECOND_TIMER - Second Timer ISR was activated.

  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. If nested ISR terminates and OS returns to previous ISR, the running ISR is changed to ISR_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 OSEK OS service (Service) identifier with least significant bit set when CPU enters Service code.

  In case of leaving OSEK OS service the attribute value is set to service identifier with least significant bit cleared thus
indicating end of service. The attribute is updated with such value on the following events occurred:
- CPU leaves OSEK OS service code and starts to execute non OSEK OS service code - i.e. the application code in the same or another task;
- CPU leaves OSEK 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**

<table>
<thead>
<tr>
<th>Traced Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>SERVICE_1_ID</td>
<td>entering Service 1</td>
</tr>
<tr>
<td>SERVICE_1_EXIT</td>
<td>leaving Service 1 (possibly due to task switching)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>SERVICE_2_ID</td>
<td>entering Service 2</td>
</tr>
<tr>
<td>SERVICE_3_ID</td>
<td>entering nested Service 3 call (e.g. hook routine)</td>
</tr>
<tr>
<td>SERVICE_3_EXIT</td>
<td>leaving Service 3 and resuming Service 2</td>
</tr>
<tr>
<td>SERVICE_2_EXIT</td>
<td>leaving Service 2 (possibly due to task switching)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**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 referred 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 OSEK state the task is currently in.

- **task stack (in ECC1 class)**
  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.

To provide system with information about application main stack area make the following is accomplished:

- for Diab compiler: 2 variables are defined in Diab link command file: _OsOrtiStart (should be allocated at the same location as __SP_INIT Diab variable) and _OsOrtiStackStart (should be allocated at the same location as __SP_END Diab variable). In this case Diab link command file may look like the following:

```c
__HEAP_START= ADDR(.bss)+SIZEOF(.bss);
__SP_INIT= ADDR(ram)+SIZEOF(ram);
__HEAP_END= __SP_INIT-SIZEOF(stack);
__SP_END= __HEAP_END;
__DATA_RAM= ADDR(.data);
__DATA_ROM= __DATA_RAM;
__DATA_END= ADDR(.sdata)+SIZEOF(.sdata);
__BSS_START= ADDR(.sbss);
__BSS_END= ADDR(.bss)+SIZEOF(.bss);
```
/ * 2 symbols for ORTI support */
_OsOrtiStart= __SP_INIT;
_OsOrtiStackStart= __SP_END;

- for Metrowerks CodeWarrior compiler: 2 variables are defined in the OS code: _OsOrtiStart (contains the _stack_addr variable value) and _OsOrtiStackStart (contains the _stack_end variable value). These variables shall be pointed in the FORCEACTIVE statement in the link command file. In this case CodeWarrior link command file may look like the following:

```
MEMORY
{
  ...
}
FORCEACTIVE { _OsOrtiStart _OsOrtiStackStart}
SECTIONS
{
  ...
}
_heap_addr  = _e_bss;
_heap_end   = _e_bss+$(_heapsize);
_stack_end  = _heap_end;
_stack_addr = _stack_end+$(_stacksize);
```

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

**NOTE** The current counter value is applicable for software counters only. The access to special purpose register is required for OS/MPC55xx hardware counters, but this mechanism is not supported in ORTI.
The following alarm information is available to the user:

- alarm status
- assigned counter
- notified task and event
- time left to expire
- cycle value for cyclic alarm

**NOTE**
The time left to expire is applicable for alarms attached to software counters only. The access to special purpose register is required for OS/MPC55xx hardware counters, but this mechanism is not supported in ORTI.

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 the queue
- first valid message
- message state
- action to be performed when message arrives
Debugging Application
Stack Debugging Support

- message receiver
- message sender

Stack Debugging Support

Stack labels

Stack labels are provided to control OSEK OS stacks usage. These labels can be seen in symbol debugger and help to control stack usage. Stack labels define boundaries of the stack area for the Extended tasks. Bottom stack labels and top stack labels for tasks are provided.

Top of stack labels have the following format:

<task name>_TOS

Bottom of stack labels have the following format:

<task name>_BOS

Stack Overflow Checking

Optional stack overflow checking can be used during run time for check tasks, ISR and main stacks usage. Main and task stacks are checked during task switch. If stack overflow is detected the ErrorHook with E_OS_SYS_STACK, E_OS_SYS_MAINSTACK or E_OS_SYS_ISRSTACK status is called in case of Extended Task stack, Main stack or ISR stack overflow accordingly. Stack overflow control is turned on if StackOverflowCheck attribute value is TRUE and Error hook is defined. For main stack checking OS uses the label defined in linker command file, _stack_end for Codewarrior compiler.

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) is checked before leaving ISR of category 2.
Sample Application

This appendix contains the text and listing of the sample customer application developed using OSEK OS.

This appendix consists of the following sections:

- **Description**
- **Source Files**

**Description**

The Sample application delivered with the OSEK Operating System should help to learn how to use OSEK OS. The Sample’s source files are located in the SAMPLE directory – it contains all files needed to create an executable file.

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 OSEK OS mechanisms and allows the user to have the first look inside the OSEK OS.

The 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, TimeScale, two resources and two messages.

Generally, Sample tasks are divided into two parts. TASKSND1, TASKSND2 and TASKCNT compose the first part (samples.c file) and TASKRCV1, TASKRCV2 and TASKSTOP are the second part (samplepc.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),

For More Information: www.freescale.com
Sample Application

Description

- starts TimeScale,
- 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 MsgA message with copy,
- 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).

TimeScale has three steps:

1. On 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 without copy (MsgA message sets MSGAEVENT event for TASKRCV1 task),
   - releases MSGAACCESS resource,
   - terminates itself.

2. On second step TASKSND2 task is activated.
   Task TASKSND2:
   - adds 3 to “ind” variable value,
   - if “ind” variable value greater or equal 5 then subtracts 5 from “ind” value,
Sample Application

Source Files

- stores SYSTEMTIMER value to body of MsgB message copy,
- gets MSGBACCESS resource to prevent access to MsgB message,
- send MsgB message with copy (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 MsgB message without copy,
- releases MSGBACCESS resource,
- terminates itself.

TASKSTOP task:
- stops TimeScale (after a while TIMLIMITALARM expires)
- terminates itself.

The user can watch “ind” variable, messages content and so on.

Source Files

Source files for the Sample application are the following:
- samplets.c – the application code (TASKSND1, TASKSND2 and TASKCNT)
- samplerv.c – the application code (TASKRCV1, TASKRCV2 and TASKSTOP)
- sample.h – header file for the application code
- main.oin – OSEK Implementation Language file, platform independent part
Sample Application
Source Files

- `msmake.bat` – command file for compiling sample using Microsoft nmake utility
- `gnumake.bat` – command file for compiling sample using GNU make utility

The directory structure of the Sample application is described in the `readme.txt` file located in the `sample\standard` directory.

To build the executable file the user should make sure that OSEKturbo OS components are properly installed on the disk and paths for the OSEK directory and compiler software are known. The makefiles were written for Microsoft Visual C++ 5.0 or v.6.0 nmake utility and for GNU make (from Cygwin package v.1.3.9) utility. Run the `MSMAKE.BAT` or `GNUMAKE.BAT` from DOS prompt to build the executable file.

To build sample application for CodeWarrior compiler run batch file with `codewarrior` parameter:

```bash
>MSMAKE codewarrior
```

To build sample application for Diab Data compiler run batch file with `diab` parameter:

```bash
>MSMAKE diab
```

By default, Sample application exception table is located at 0x00000000 address. Therefore correct MSR value (MSR IP bit is cleared) should be set prior to initial StartOS() call. If exception table is located at 0xffffffff address, MSR IP bit should be set prior to initial StartOS() call and Sample makefile’s should be adjusted accordingly (the address of “vect” section in the linker script). (See the “samplerv.c” file for detail).

Makefile uses the system environment variables CWDIR to get compiler components, OSEKDIR to get OSEK components and SYSGENDIR to get SysGen shared components.

When all produced files are ready, the executable file can be load into the simulator and run.

For More Information: www.freescale.com
System Service Timing

This appendix provides information about OS services execution time.

This appendix consists of the following sections:

- General Notes
- Data Sheet

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;
STARTUPHOOK = FALSE;
SHUTDOWNHOOK = FALSE;
PRETASKHOOK = FALSE;
POSTTASKHOOK = FALSE;
ERRORHOOK = FALSE;
USEGETSERVICEID = FALSE;
USEPARAMETERACCESS = FALSE;
FastTerminate = TRUE;
FastScheduler = TRUE;
ResourceScheduler = FALSE;
ACTIVATION = 1; (for all TASK objects)
SCHEDULE = FULL; (for all TASK objects)
ISR category 2 defined
Four tasks present in the application

For More Information: www.freescale.com
The “Latency” column presents the time for which the interrupts are disabled in the service.

**Data Sheet**

The table below contains run-time services characteristics for OSEKturbo OS/MPC55xx. Used CPU clock frequency is 12 MHz.

**Table B.1 Run-time Services Timing Characteristics**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles</td>
<td>µs</td>
</tr>
<tr>
<td><strong>StartOS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial</strong></td>
<td></td>
<td>454</td>
<td>37.83</td>
</tr>
<tr>
<td><strong>ActivateTask</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial</strong></td>
<td>Task activated, no rescheduling</td>
<td>95</td>
<td>7.91</td>
</tr>
<tr>
<td><strong>Initial</strong></td>
<td>Task activated and rescheduled</td>
<td>143</td>
<td>11.91</td>
</tr>
<tr>
<td><strong>Internal resource defined</strong></td>
<td>Task activated and rescheduled</td>
<td>187</td>
<td>15.58</td>
</tr>
<tr>
<td><strong>SCHEDULE = NON for all tasks</strong></td>
<td>Task activated, no rescheduling</td>
<td>66</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>TerminateTask</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial</strong></td>
<td>Task terminated, return to lower prio task</td>
<td>86</td>
<td>7.16</td>
</tr>
<tr>
<td><strong>Initial</strong></td>
<td>Task terminated, new task started</td>
<td>85</td>
<td>7.083</td>
</tr>
<tr>
<td><strong>Internal resource defined</strong></td>
<td>Task terminated, return to lower prio task</td>
<td>161</td>
<td>13.41</td>
</tr>
<tr>
<td><strong>Internal resource defined</strong></td>
<td>Task terminated, new task started</td>
<td>171</td>
<td>14.25</td>
</tr>
<tr>
<td><strong>SCHEDULE = NON for all tasks</strong></td>
<td>Task terminated, return to lower prio task</td>
<td>86</td>
<td>7.16</td>
</tr>
<tr>
<td><strong>SCHEDULE = NON for all tasks</strong></td>
<td>Task terminated, new task started</td>
<td>85</td>
<td>7.08</td>
</tr>
</tbody>
</table>
### Table B.1  Run-time Services Timing Characteristics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles</td>
<td>µs</td>
</tr>
<tr>
<td>ChainTask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>Task terminated, next task started</td>
<td>86</td>
<td>7.16</td>
</tr>
<tr>
<td>Internal resource defined</td>
<td>Task terminated, next task started</td>
<td>172</td>
<td>14.33</td>
</tr>
<tr>
<td>SCHEDULE = NON for all tasks</td>
<td>Task terminated, next task started</td>
<td>82</td>
<td>6.83</td>
</tr>
<tr>
<td>Schedule</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>No rescheduling, all tasks are preemptable</td>
<td>71</td>
<td>5.91</td>
</tr>
<tr>
<td>Internal resource defined</td>
<td>No rescheduling, all tasks are preemptable</td>
<td>134</td>
<td>11.16</td>
</tr>
<tr>
<td>SCHEDULE = NON for all tasks</td>
<td>No rescheduling</td>
<td>71</td>
<td>5.91</td>
</tr>
<tr>
<td></td>
<td>Rescheduling, other task becomes running</td>
<td>105</td>
<td>8.75</td>
</tr>
<tr>
<td>GetTaskId</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td></td>
<td>21</td>
<td>1.75</td>
</tr>
<tr>
<td>GetTaskState</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>For running task</td>
<td>60</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>For ready task</td>
<td>78</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>For suspended task</td>
<td>74</td>
<td>6.16</td>
</tr>
<tr>
<td>EnableAllInterrupts</td>
<td></td>
<td>15</td>
<td>1.25</td>
</tr>
<tr>
<td>DisableAllInterrupts</td>
<td></td>
<td>13</td>
<td>1.08</td>
</tr>
<tr>
<td>ResumeAllInterrupts</td>
<td></td>
<td>45</td>
<td>3.75</td>
</tr>
<tr>
<td>SuspendAllInterrupts</td>
<td></td>
<td>49</td>
<td>4.08</td>
</tr>
</tbody>
</table>

For More Information: [www.freescale.com](http://www.freescale.com)
### Table B.1  Run-time Services Timing Characteristics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles µs</td>
<td>cycles µs</td>
</tr>
<tr>
<td>ResumeOSInterrupts</td>
<td>Initial</td>
<td>52</td>
<td>4.33</td>
</tr>
<tr>
<td>SuspendOSInterrupts</td>
<td>Initial</td>
<td>47</td>
<td>3.91</td>
</tr>
<tr>
<td>GetResource</td>
<td>a resource defined</td>
<td>102</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>call from ISR level</td>
<td>98</td>
<td>8.16</td>
</tr>
<tr>
<td>ReleaseResource</td>
<td>a resource defined</td>
<td>138</td>
<td>11.5</td>
</tr>
<tr>
<td></td>
<td>call from ISR level</td>
<td>124</td>
<td>10.33</td>
</tr>
<tr>
<td>SetEvent</td>
<td>CC = ECC1 SCHEDULE = NON for all tasks an event defined</td>
<td>119</td>
<td>9.91</td>
</tr>
<tr>
<td></td>
<td>Event set, task was not waiting it</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event set, task becomes ready</td>
<td>141</td>
<td>11.75</td>
</tr>
<tr>
<td></td>
<td>CC = ECC1 an event defined</td>
<td>126</td>
<td>10.5</td>
</tr>
<tr>
<td></td>
<td>Event set, task was not waiting it</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event set, task becomes ready</td>
<td>162</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td>Event set, task becomes running</td>
<td>378</td>
<td>31.5</td>
</tr>
<tr>
<td>ClearEvent</td>
<td>CC = ECC1 SCHEDULE = NON for all tasks an event defined</td>
<td>51</td>
<td>4.25</td>
</tr>
</tbody>
</table>

For More Information: www.freescale.com
### Table B.1  Run-time Services Timing Characteristics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles</td>
<td>µs</td>
</tr>
<tr>
<td>CC = ECC1</td>
<td>an event defined</td>
<td>50</td>
<td>4.16</td>
</tr>
<tr>
<td>GetEvent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC = ECC1</td>
<td>SCHEDULE = NON for all tasks</td>
<td>31</td>
<td>2.58</td>
</tr>
<tr>
<td>WaitEvent</td>
<td>an event defined</td>
<td>30</td>
<td>2.5</td>
</tr>
<tr>
<td>CC = ECC1</td>
<td>an event defined</td>
<td>76</td>
<td>6.33</td>
</tr>
<tr>
<td></td>
<td>Event was already set, task remains running</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task becomes waiting</td>
<td>300</td>
<td>25</td>
</tr>
<tr>
<td>CC = ECC1</td>
<td>an event defined</td>
<td>76</td>
<td>6.33</td>
</tr>
<tr>
<td></td>
<td>Event was already set, task remains running</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task becomes waiting</td>
<td>298</td>
<td>24.83</td>
</tr>
<tr>
<td>SendMessage</td>
<td>Message size 16 bytes</td>
<td>377</td>
<td>31.41</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>Message size 16 bytes</td>
<td>382</td>
<td>31.83</td>
</tr>
<tr>
<td>InitCounter</td>
<td>a counter defined</td>
<td>25</td>
<td>2.08</td>
</tr>
<tr>
<td>CounterTrigger</td>
<td>a counter defined</td>
<td>113</td>
<td>9.41</td>
</tr>
<tr>
<td></td>
<td>1 alarm is set to a counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alarm is not expired</td>
<td>215</td>
<td>17.91</td>
</tr>
<tr>
<td></td>
<td>Notified task becomes ready</td>
<td>278</td>
<td>23.16</td>
</tr>
<tr>
<td></td>
<td>Notified task becomes running</td>
<td>326</td>
<td>27.16</td>
</tr>
</tbody>
</table>

---

OSEKturbo OS/MPC55xx

For More Information: www.freescale.com
Table B.1 Run-time Services Timing Characteristics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles</td>
<td>μs</td>
</tr>
<tr>
<td>10 tasks, 10 alarms are set to a counter</td>
<td>Alarm is not expired</td>
<td>501</td>
<td>41.75</td>
</tr>
<tr>
<td></td>
<td>Notified tasks become ready</td>
<td>1206</td>
<td>100.5</td>
</tr>
<tr>
<td></td>
<td>High priority notified task becomes running</td>
<td>1259</td>
<td>104.91</td>
</tr>
<tr>
<td></td>
<td>GetCounterValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a counter defined</td>
<td>30</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>GetCounterInfo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a counter defined</td>
<td>33</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>GetAlarmBase</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>an alarm defined</td>
<td>63</td>
<td>5.25</td>
</tr>
<tr>
<td></td>
<td>SetRelAlarm</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>an alarm defined</td>
<td>Alarm set</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm expires immediately, notified task becomes ready</td>
<td>396</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm expires immediately, notified task becomes running</td>
<td>434</td>
</tr>
<tr>
<td></td>
<td>10 alarm are defined, 9 alarms are set on a counter</td>
<td>Alarm set</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm expires immediately, notified task becomes ready</td>
<td>690</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm expires immediately, notified task becomes running</td>
<td>720</td>
</tr>
</tbody>
</table>
Table B.1 Run-time Services Timing Characteristics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>Timing</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cycles</td>
<td>µs</td>
</tr>
<tr>
<td>SetAbsAlarm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 alarm are</td>
<td>Alarm set</td>
<td>186</td>
<td>15.5</td>
</tr>
<tr>
<td>defined,</td>
<td>Alarm expires immediately, notified task becomes ready</td>
<td>356</td>
<td>29.66</td>
</tr>
<tr>
<td>9 alarms are</td>
<td>Alarm expires immediately, notified task becomes running</td>
<td>394</td>
<td>32.83</td>
</tr>
<tr>
<td>on a counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 alarm is set</td>
<td></td>
<td>189</td>
<td>15.75</td>
</tr>
<tr>
<td>on a counter</td>
<td>Alarm expires immediately, notified task becomes ready</td>
<td>642</td>
<td>53.5</td>
</tr>
<tr>
<td>10 tasks, 10</td>
<td>Alarm expires immediately, notified task becomes running</td>
<td>674</td>
<td>56.16</td>
</tr>
<tr>
<td>alarms are set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 alarm is set</td>
<td></td>
<td>82</td>
<td>6.83</td>
</tr>
<tr>
<td>on a counter</td>
<td></td>
<td>86</td>
<td>7.16</td>
</tr>
<tr>
<td>10 tasks, 10</td>
<td></td>
<td>94</td>
<td>7.83</td>
</tr>
<tr>
<td>alarms are set</td>
<td></td>
<td>94</td>
<td>7.83</td>
</tr>
<tr>
<td>1 alarm is set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on a counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartTimeScale</td>
<td>TimeScale defined</td>
<td>163</td>
<td>13.58</td>
</tr>
<tr>
<td>StartTimeScale</td>
<td>Task activated, no rescheduling</td>
<td>230</td>
<td>19.16</td>
</tr>
<tr>
<td>StopTimeScale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TimeScale defined</td>
<td></td>
<td>37</td>
<td>3.08</td>
</tr>
</tbody>
</table>

The table below contains interrupt frame characteristics for OSEKturbo OS/MPC55xx.
### Table B.2  Interrupt Frame Characteristics for ISR Category 2\(^a\)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Conditions</th>
<th>cycles</th>
<th>µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>From interrupt until first ISR command</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td></td>
<td>164</td>
<td>13.66</td>
</tr>
<tr>
<td>From last ISR command until task/idle loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial</td>
<td>Return to interrupted task</td>
<td>129</td>
<td>10.75</td>
</tr>
<tr>
<td></td>
<td>Task rescheduling occurs</td>
<td>106</td>
<td>8.83</td>
</tr>
<tr>
<td></td>
<td>Return to Idle Loop</td>
<td>129</td>
<td>10.75</td>
</tr>
</tbody>
</table>

\(^a\) 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/idle loop for ISR category 2.
Memory Requirements

This appendix provides information about the amount of ROM and RAM directly used by various versions of the OSEK OS.

This appendix consists of the following sections:

- OSEK Operating System Memory Usage
- Data Sheet for CodeWarrior

OSEK Operating System Memory Usage

The table below contains the data about ROM and RAM needed for the OSEK 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 reflect 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:

\[
\begin{align*}
CC & = \text{BCC1;} \\
\text{STATUS} & = \text{STANDARD;} \\
\text{STARTUPHOOK} & = \text{FALSE;} \\
\text{SHUTDOWNHOOK} & = \text{FALSE;} \\
\text{PRETASKHOOK} & = \text{FALSE;} \\
\text{POSTTASKHOOK} & = \text{FALSE;} \\
\text{ERRORHOOK} & = \text{FALSE;} \\
\text{USEGETSERVICEID} & = \text{FALSE;} \\
\text{USEPARAMETERACCESS} & = \text{FALSE;} \\
\text{ResourceScheduler} & = \text{FALSE;} \\
\text{FastTerminate} & = \text{TRUE;} \\
\text{FastScheduler} & = \text{TRUE;} \\
\text{MessageCopyAllocation} & = \text{USER;} \\
\text{ACTIVATION} & = 1; \text{ (for all TASK objects)}
\end{align*}
\]

For More Information: www.freescale.com
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 BCC1 Conformance Class without any additional mechanisms and this is the minimal OSEK 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. Thus, the system functionality grows up.

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 vector table for ROM.

In the formulas in the table the following symbols are used:

- **T** is the number of Tasks
- **R** is the number of Resources (without RES_SCHEDULER)
- **IR** is the number of Resources with ISR priority
- **C** is the number of Counters
- **A** is the number of Alarms
- **M** is the number of Messages

For More Information: www.freescale.com
## Data Sheet for CodeWarrior

### Table C.1  Operating System Memory Requirements (CodeWarrior)

<table>
<thead>
<tr>
<th>Number</th>
<th>System Properties (configuration)</th>
<th>Conformance Class</th>
<th>RAM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial</td>
<td>BCC1</td>
<td>1+1*T</td>
<td>652+5*T</td>
</tr>
<tr>
<td>1</td>
<td>Initial SCHEDULE = NON for all tasks</td>
<td></td>
<td>1+1*T</td>
<td>608+5*T</td>
</tr>
<tr>
<td>2</td>
<td>Initial SCHEDULE = NON for one task and FULL for other one</td>
<td></td>
<td>1+1*T</td>
<td>672+5*T</td>
</tr>
<tr>
<td>3</td>
<td>Changed: STATUS = EXTENDED</td>
<td></td>
<td>89+88*T</td>
<td>1352+8*T</td>
</tr>
<tr>
<td>4</td>
<td>Changed: ERRORHOOK = TRUE</td>
<td></td>
<td>90+88*T</td>
<td>1796+8*T</td>
</tr>
<tr>
<td>5</td>
<td>Changed: USEGETSERVICEID = TRUE USEPARAMETERACCESS = TRUE</td>
<td></td>
<td>91+88*T</td>
<td>2000+8*T</td>
</tr>
<tr>
<td>6</td>
<td>Initial: Counter defined</td>
<td></td>
<td>1+1<em>T+8</em>C</td>
<td>892+5<em>T+8</em>C</td>
</tr>
<tr>
<td>7</td>
<td>Changed: Alarm with ACTION=ACTIVATETASK defined</td>
<td></td>
<td>1+1<em>T+12</em>C+20*A</td>
<td>1844+5<em>T+8</em>C+2*A</td>
</tr>
<tr>
<td>8</td>
<td>Initial: Message with ACTION=NONE defined</td>
<td></td>
<td>1+1<em>T+16</em>M</td>
<td>1028+5<em>T+12</em>M</td>
</tr>
<tr>
<td>9</td>
<td>Initial: Message with ACTION=ACTIVATETASK defined</td>
<td></td>
<td>1+132<em>T+20</em>M</td>
<td>1820+12<em>T+16</em>M</td>
</tr>
<tr>
<td>10</td>
<td>Initial: Message with ACTION=CALLBACK defined</td>
<td></td>
<td>1+1<em>T+20</em>M</td>
<td>1136+5<em>T+16</em>M</td>
</tr>
<tr>
<td>11</td>
<td>Initial: Resource for task defined</td>
<td></td>
<td>93+92<em>T+8</em>R</td>
<td>1100+8<em>T+1</em>R</td>
</tr>
<tr>
<td>12</td>
<td>Changed: ISR category 2 defined</td>
<td></td>
<td>98+92<em>T+8</em>R</td>
<td>4060+8<em>T+1</em>R</td>
</tr>
<tr>
<td>13</td>
<td>Changed: InterruptDispatcher = OneLevel</td>
<td></td>
<td>98+92<em>T+8</em>R</td>
<td>4728+8<em>T+1</em>R</td>
</tr>
<tr>
<td>14</td>
<td>Changed: Resource referenced by ISR</td>
<td></td>
<td>102+92<em>T+12</em>R</td>
<td>4888+8<em>T+1</em>R</td>
</tr>
</tbody>
</table>

OSEKturbo OS/MPC55xx

For More Information: www.freescale.com
### Table C.1 Operating System Memory Requirements (CodeWarrior)

<table>
<thead>
<tr>
<th>Number</th>
<th>System Properties (configuration)</th>
<th>Conformance Class</th>
<th>RAM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Initial:</td>
<td>ECC1</td>
<td>93+92*T</td>
<td>1336+12*T</td>
</tr>
<tr>
<td>16</td>
<td>Initial: an event defined</td>
<td></td>
<td>101+100*T</td>
<td>1576+12*T</td>
</tr>
<tr>
<td>17</td>
<td>Changed SCHEDULE = NON for all tasks</td>
<td></td>
<td>101+100*T</td>
<td>1532+12*T</td>
</tr>
<tr>
<td>18</td>
<td>Initial an event defined, SCHEDULE = NON for one task and FULL for other one</td>
<td></td>
<td>101+100*T</td>
<td>1592+12*T</td>
</tr>
<tr>
<td>19</td>
<td>Changed: STATUS = EXTENDED</td>
<td></td>
<td>101+100*T</td>
<td>1640+12*T</td>
</tr>
<tr>
<td>20</td>
<td>Changed: ERRORHOOK = TRUE</td>
<td></td>
<td>102+100*T</td>
<td>2084+12*T</td>
</tr>
<tr>
<td>21</td>
<td>Changed: USEGETSERVICEID = TRUE USEPARAMETERACCESS = TRUE</td>
<td></td>
<td>103+100*T</td>
<td>2288+12*T</td>
</tr>
<tr>
<td>22</td>
<td>Initial an event defined, Counter defined</td>
<td></td>
<td>101+100<em>T+8</em>C</td>
<td>1824+12<em>T+8</em>C</td>
</tr>
<tr>
<td>23</td>
<td>Changed: Alarm with ACTION=SETEVENT defined</td>
<td></td>
<td>101+100<em>T+12</em>C+24*A</td>
<td>2952+12<em>T+8</em>C+12*A</td>
</tr>
<tr>
<td>24</td>
<td>Initial an event defined, Message with ACTION=NONE defined</td>
<td></td>
<td>101+100<em>T+16</em>M</td>
<td>1952+12<em>T+12</em>M</td>
</tr>
<tr>
<td>25</td>
<td>Initial an event defined, Message with ACTION=ACTIVATE TASK defined</td>
<td></td>
<td>101+100<em>T+20</em>M</td>
<td>2152+12<em>T+16</em>M</td>
</tr>
<tr>
<td>26</td>
<td>Initial an event defined, Message with ACTION=SETEVENT defined</td>
<td></td>
<td>101+100<em>T+24</em>M</td>
<td>2124+12<em>T+20</em>M</td>
</tr>
<tr>
<td>27</td>
<td>Initial an event defined, Resource for task defined</td>
<td></td>
<td>105+104<em>T+8</em>R</td>
<td>1848+12<em>T+1</em>R</td>
</tr>
<tr>
<td>28</td>
<td>Changed: ISR category 2 defined</td>
<td></td>
<td>110+104<em>T+8</em>R</td>
<td>5004+12<em>T+1</em>R</td>
</tr>
<tr>
<td>29</td>
<td>Changed: InterruptDispatcher = OneLevel</td>
<td></td>
<td>110+104<em>T+8</em>R</td>
<td>5760+12<em>T+1</em>R</td>
</tr>
<tr>
<td>30</td>
<td>Changed: Resource referenced by ISR</td>
<td></td>
<td>114+104<em>T+12</em>R</td>
<td>5216+12<em>T+1</em>R</td>
</tr>
</tbody>
</table>

For More Information: www.freescale.com
System Generation Error Messages

This appendix explains OSEK 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**
- **Error Message Format**
- **List of Messages**

**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.

---

**NOTE**

All warning messages or some types of them can be suppressed by using the `-w` option.

---
System Generation Error Messages

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:

\[
\text{[<filename>(<line_number>) : ]information <prefix>####:<message>}
\]
\[
\text{[<filename>(<line_number>) : ]warning <prefix>####:<message>}
\]
\[
\text{[<filename>(<line_number>) : ]error <prefix>####:<message>}
\]
\[
\text{[<filename>(<line_number>) : ]Fatal Error <prefix>####:<message>}
\]

where:

- `<filename>` file name
- `<line_number>` line number
- `<prefix>` component specific prefix
- `####` number of message
- `<message>` short description of the error

The message format can be set to a tree-like form (if possible, for example if an error is encountered while the project verifying) by means `-b` option. The message includes the object type, the object name, the error code and a short error description. In this case the error message has the following format:

\[
\text{CPU<name>,<OBJECT><name>,<attribute>:information<prefix>####:<message>}
\]
\[
\text{CPU<name>,<OBJECT><name>,<attribute>:warning<prefix>####:<message>}
\]
\[
\text{CPU<name>,<OBJECT><name>,<attribute>:error<prefix>####:<message>}
\]
\[
\text{CPU<name>,<OBJECT><name>,<attribute>:Fatal Error<prefix>####:<message>}
\]

where:

- `<name>` name of corresponding object - CPU or object
- `<OBJECT>` keyword used to identify object type in OIL
- `<attribute>` name of outermost attribute (in some cases all nested attributes are presented through dots). Optional, present, if the message is associated with a particular attribute
System Generation Error Messages
List of Messages

<prefix>  component specific prefix
####   number of message
<message>  short description of the error

Below error and warning messages generated by SysGen components are described.

List of Messages

SysGen consists of several components, which work on different stages of OIL file processing. Each component generates messages with specific prefix before the number. The following prefixes are used: SG - for SysGen Engine main component, OR - for OIL Reader component, which reads specified OIL file into internal repository, TD - for Target-Specific DLL, which process stored OIL file with target platform awareness.

SysGen Engine Messages

SG0001: Option '-v' is used, ignoring other options
Information
When -v option is defined, System Generator is run only to output versions of all its components. So all other options are ignored.

SG0002: Option <option> has been redefined; effective value is <value>
Warning
Only one value can be used for the following options: -c, -h, -n, -o, -p. Being defined twice, their last value will be used as SysGen argument.

SG0003: Option <option> requires argument
Error
The following options should be defined with an argument: -c, -f, -h, -i, -n, -o, -p, -s.

SG0004: Option <option> does not accept arguments
Error
The following options should not be defined with an argument: -b, -t, -v.
System Generation Error Messages

List of Messages

SG0005: Unexpected token <token>
Error
An unrecognizable token was found in the input stream. The SysGen option or OIL file name is expected.

SG0006: Closing quotation mark is expected
Fatal Error
The quotation mark is missed. The command line contains odd number of quotes.

SG0007: Cannot open command argument file
Fatal Error
The command argument file cannot be open if it does not exist or there are no permissions to open the file or directory.

SG0008: Cyclic reference in the command argument file
Fatal Error
The command file contains -f option that has a reference to the same file as an argument. Also some command files can have number of recursive references.

SG0009: Input OIL file must be defined
Fatal Error
The input OIL file cannot be found if it does not exist or there are no permissions to open the file or directory.

SG0010: Processing halted; possibly there is not enough memory
Fatal Error
The message is generated by the SysGen either there is no enough free RAM memory on users' PC or some internal error is occurred and it would be resolved with the help of support team.
Please, report to osek@helpline.sps.mot.com.

SG0201: Error loading Target DLL <name>
Fatal Error
Target DLL with the specified name is absent.
SG0202: Wrong Target DLL <name> version
   Fatal Error
   Target DLL version shall be 1.x or 2.x. Other versions of Target DLL are not supported.

SG0203: Target DLL <name> failed
   Fatal Error
   Target DLL is corrupted.

SG0204: Target DLL <name> cannot be found
   Fatal Error
   The Target DLL name is not found in the Windows Registry.

SG0301: Implementation definition database in Windows Registry is invalid or does not exist
   Fatal Error
   Implementation definition database defined by installation program, was corrupted or removed.

SG0303: Container <name> does not exist
   Fatal Error
   Implementation definition database does not contain information about specified container.

SG0401: Wrong license
   Fatal Error
   The error message is generated if OSEKTURBO or FULL feature is not defined in the OSEK OS license. Also license file can be corrupted. For more detailed information see Flex License Manager documentation.

SG5001: Cannot open template file <filename>
   Fatal Error
   The System Generator cannot open template file for use if the template file which name defined in the Windows Registry, does not exist or there are no permissions to open the file or directory.
System Generation Error Messages
List of Messages

SG5002: Cannot open output file <filename>
Fatal 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.

SG5003: Template file <filename> is corrupt or invalid
Fatal Error
The contents of the template file are corrupt.

SG5004: Cannot write output file <filename>. Possibly not enough space on drive
Fatal Error
The output file cannot be written if there is no enough free disk space or there is no appropriate permission to write the output file.

OIL Reader Messages

OR0000: Cannot open input file
Fatal Error
The file does not exist, or there are no permissions to open the file or directory.

OR0002: Unterminated file path
Fatal Error
The incorrect syntax of an include directive.

OR0003: <file path> or "file path" expected
Fatal Error
The incorrect syntax of an include directive.

OR0004: Wrong #include directive format
Fatal Error
The incorrect syntax of an include directive.
System Generation Error Messages
List of Messages

OR0053: OIL version expected
Fatal Error
OIL version should be defined as value for the OIL_VERSION attribute at
the beginning of OIL file.

OR0005: Unclosed comment
Error
The OIL file may contain C++-style comments (/ * */ and //). If the comment
is started be /*, then the end of the comment is defined by */.

OR0006: Unknown directive
Error
Unknown preprocessing directive was detected.

OR0007: Unterminated string
Error
The attribute's value of STRING type should be completed by double-quote.

OR0008: Wrong number format
Error
The number format on any unsigned integer number (possibly restricted to
a range of numbers).

OR0009: Floating point is not supported
Error
Floating point is not supported by the 2.0 and 2.0e OIL versions. The
message is generated if an OIL file of 2.0 or 2.0e format contains an attribute
of FLOAT type.

OR0010: Syntax error
Error
A syntax error was found in the input stream. Problems of this type can
sometimes be attributed to a syntactical or clerical error. For example:
TASK taskA
{  
   PRIORITY = 5 // No closing semicolon
   SCHEDULE = FULL;
};
In the preceding example, the error message will be generated for the line which contains the definition of $SCHEDULE$ attribute, although the true source of the error appears on the line just above. As a general rule, make sure to also examine the lines above the line listed in the error message when trying to determine the cause.

**OR0011: Attribute value does not match type**
Error
The value does not correspond to attribute type.

**OR0012: Attribute value is out of range**
Error
The attribute value is greater or less than range defined.

**OR0013: Attribute range does not match type**
Error
The value range does not correspond to attribute type.

**OR0014: Range bounds are wrong**
Error
The first bound of the value range defines the minimal value of the attribute, and the second bound specifies the maximal value, i.e. the first bound has to be less than second one. For example, [1..256].

**OR0015: Range is not allowed**
Error
Object attributes of some types (ENUM, BOOLEAN, STRING) and object references have not to be defined with value range in the implementation definition part.

**OR0016: ENUM attribute requires value list**
Error
The attributes of ENUM type have to be defined with one or more values allowed for the attribute.
OR0017: AUTO is not allowed
Error
If the implementation definition part has an attribute with the WITH_AUTO specifier, then the AUTO value can be set for the attribute.

OR0018: Duplicated values in ENUM attribute
Error
Values listed for an attribute of ENUM type, have to differ each other.

OR0019: Range of BOOLEAN attribute must be [TRUE, FALSE]
Error
The attribute of BOOLEAN type can have either TRUE or FALSE value.

OR0020: Reference type must be appended with "_TYPE"
Error
The reference type is taken from the referenced object and _TYPE particle, e.g. a reference to a task shall use the TASK_TYPE keyword as reference type.

OR0021: Attribute type doesn’t match the type of one previously defined
Error
The same subattributes cannot be defined with different types.

OR0022: Duplicated attribute definition
Error
The implementation part of OIL file contains two definition of the same attribute.

OR0023: Wrong reference type
Error
According to OIL standard the reference type is taken from the referenced object and _TYPE particle, e.g. a reference to a task shall use the TASK_TYPE keyword as reference type.

OR0024: Duplicated predefined attribute
Error
The predefined attributes CONTAINER and SECTION are already defined (for OIL2.0e).
OR0025: Standard object <name> is not defined
Error
The implementation definition must contain all standard objects.

OR0026: Standard attribute <name> is not defined
Error
The implementation definition must contain all standard attributes.

OR0027: Standard attribute must have WITH_AUTO specifier
Error
This attribute definition does not correspond to OIL standard.

OR0028: Standard attribute must have [ ] specifier
Error
This attribute definition does not correspond to OIL standard.

OR0029: Standard attribute must not have [ ] specifier
Error
This attribute definition does not correspond to OIL standard.

OR0030: Standard attribute must have range
Error
This attribute definition does not correspond to OIL standard.

OR0031: Standard attribute range cannot be expanded
Error
This attribute definition does not correspond to OIL standard. The value range for standard attributes may be restricted but not expanded.

OR0032: Standard attribute must have ENUM value list
Error
This attribute definition does not correspond to OIL standard.

OR0033: Standard dependent attributes are expected
Error
According to OIL standard the attribute has to contain subattributes' definition.
OR0034: Standard attribute ENUM value list cannot be expanded
    Error
    This attribute definition does not correspond to OIL standard. The value
    range of standard attributes may be restricted but not expanded.

OR0035: Standard reference type mismatch
    Error
    The type of standard reference does not correspond to OIL standard.

OR0036: Standard attribute type mismatch
    Error
    The type of standard attribute does not correspond to OIL standard.

OR0037: Duplicated object
    Error
    The object has been already defined in the implementation definition.

OR0038: CFG <name> is not defined
    Error
    CFG statements are not supported by OIL standard.

OR0039: No implementation is found
    Error
    No implementation definition was found in the input file.

OR0040: Container type does not match implementation
    Error
    The OIL file contains a container type that is not defined by implementation.

OR0041: Section name does not match implementation
    Error
    The OIL file contains a section type that is not defined by implementation.

OR0042: 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.
OR0043: Attribute not defined in the implementation
Error
This attribute is not specified in the implementation definition.

OR0044: Attribute is already defined
Error
The attribute with specified name has been already defined.

OR0045: Single value is expected
Error
The attribute requires single value (for OIL 2.0 and 2.0e).

OR0046: Dependent attributes corresponded to wrong value not allowed
Error
The subattributes' values in the application do not correspond to them in the implementation definition.

OR0047: Reference is expected
Error
Invalid name of the referenced object.

OR0048: Referenced object is not found
Error
The referenced object with specified name is not found.

OR0049: Reference type mismatch
Error
The referenced object type does not correspond to reference type.

OR0050: Container attributes are not allowed
Error
The message is generated if CPU container is defined by attributes. Container attributes are allowed only for OIL 2.0e format.
OR0051: 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).

OR0052: Dependent NO_DEFAULT attribute is defined in the default branch of his parent
Error
The implementation definition must not have attributes with the default value that contain subattributes with NO_DEFAULT specifier.

OR0054: Standard attribute must not have WITH_AUTO specifier
Error
This attribute definition does not correspond to OIL standard.

OR0056: Object is already defined with different type
Error
The given name has already been used for a system object of the other type.

OR0057: <type> object must be defined
Error
The OS or APPMODE object cannot be missed in OIL file.

OR0058: Only one <type> object can be defined
Error
Only one OS object can be defined.

OR0059: Implementation is already defined
Error
Only one implementation can be defined in OIL file.

OR0060: CPU is already defined
Error
Only one CPU container can be defined in OIL file.
System Generation Error Messages

List of Messages

OR0061: No CPU in the file
Error
CPU container must be defined in OIL file.

OR0062: Include directive within object is not allowed
Error
The #include directive(s) cannot be placed inside object definition.

OR0063: Related item location
Information
The message refers to the item in implementation or application definition that is relevant to previous error message.

OR0064: Only CPU container allowed
Error
Only CPU container is supported by OIL format.

OR0065: Nested objects are not allowed
Error
The message is generated when OIL file is read. Subobjects cannot be defined in the object definition according to OIL standard.

OR0066: Unsupported OIL Version
Fatal Error
OSEK SysGen supports OIL 2.1, OIL 2.2 and OIL2.3.
OIL version should be defined as value for the OIL_VERSION attribute at the beginning of OIL file.

OR0067: <character> expected
Information
This character is expected according to OIL file syntax.

OR0068: String expected
Information
The string is expected according to OIL file syntax.
OR0069: Implementation name expected
Information
The implementation name has to be defined according to OIL file syntax.

OR0070: Value expected
Information
The attribute value has to be defined according to OIL file syntax.

OR0071: Attribute name expected
Information
The attribute name has to be defined according to OIL file syntax.

OR0072: Unknown character <char>
Information
This character has not to be used in the OIL file. The symbol code is presented.

OR0073: Cyclic #include reference
Information
Cyclic #include references are not allowed.

OR0074: Object <type> is not standard
Warning
Nonstandard object types have not to be defined in the implementation definition.

OR5011: Unknown object in implementation part
Error
The object defined in OIL file absents in the corresponding implementation definition file.

OR5012: Unknown attribute in implementation part
Error
The attribute defined in OIL file absents in the corresponding implementation definition file.
System Generation Error Messages
List of Messages

OR5013: Type of attribute is invalid
Error
The attribute type defined in OIL file does not correspond to the registered implementation definition file.

OR5014: Type of reference is invalid
Error
The attribute type defined in OIL file does not correspond to the registered implementation definition file.

OR5017: Single value <name> attribute is declared as multiple value
Error
The attribute defined in OIL file as multiple has single value in the registered implementation definition file.

OR5018: Multiple value <name> attribute is declared as single value
Error
The attribute defined in OIL file as single has multiple value in the registered implementation definition file.

OR5019: Attribute must not have WITH_AUTO specifier
Error
The attribute is defined in OIL file with WITH_AUTO specifier, but this is inconsistent with the corresponding implementation definition file.

OR5020: Attribute must have WITH_AUTO specifier
Error
The attribute is defined in OIL file without WITH_AUTO specifier, but this is inconsistent with the corresponding implementation definition file.

OR5021: Attribute must not have default value
Error
The default value is defined for the attribute in OIL file, but the corresponding implementation definition file does not specify default value for this attribute.

For More Information: www.freescale.com
OR5022: Wrong attribute default value
Error
The attribute default value defined in OIL file does not correspond to it in the appropriate implementation definition file.

OR5023: Attribute must have default value
Error
The default value is not defined for the attribute in OIL file, but the corresponding implementation definition file specifies the default value for this attribute.

OR5024: Attribute must not have range
Error
The value range is defined for the attribute in OIL file, but the corresponding implementation definition file does not assign default value for this attribute.

OR5025: Wrong minimum value
Error
The minimum bound in the value range defined for the attribute in OIL file and in the corresponding implementation definition file is inconsistent.

OR5026: Wrong maximum value
Error
The maximum bound in the value range defined for the attribute in OIL file and in the corresponding implementation definition file is inconsistent.

OR5027: Attribute must have range
Error
The value range defined for the attribute in the implementation definition file, is absent for this attribute in the appropriate OIL file.

OR5028: Wrong ENUM value
Error
The value list defined for the attribute in OIL file and in the corresponding implementation definition file is inconsistent.
OR5029: Some ENUM values are not defined for attribute

Error
Some of the values defined for the attribute in the implementation definition file, is absent for this attribute in the appropriate OIL file.

OR5030: Unknown container type

Error
The application definition specifies a container type that was not defined.

OR5031: Unknown section

Error
The application definition specifies a section that was not defined (for OIL 2.0e).

OR5032: Unknown implementation

Error
The application definition specifies an implementation that was not defined (for OIL 2.0e).

OR5050: Attribute must not have NO_DEFAULT value

Error
The NO_DEFAULT specifier is defined for the attribute in OIL file, but in the corresponding implementation this attribute does not have this specifier.

OR5051: Attribute must have NO_DEFAULT value

Error
The NO_DEFAULT specifier is not defined for the attribute in OIL file, but the corresponding implementation definition file specify it for this attribute.

OR5060: Object not defined

Error
The object is defined in the implementation definition file, but it is absent in the appropriate OIL file.

OR5061: Attribute not defined

Error
The attribute is defined in the implementation definition file, but it is absent in the appropriate OIL file.
System Generation Error Messages
List of Messages

OR5062:  Depended attribute group for value: <value> is absent
Error
The values of subattributes shall be defined if these parameters do not have
default values in the implementation definition.

OR5101:  Can't create mode <name>, ignored
Warning
The message is generated when OIL file was converted from 2.0e OIL
format to 2.1 format. Application mode in 2.1 OIL format contains only
references to tasks defined in the application. APPMODE object is not
created.

Target-Specific DLL Messages

TD0001:  Wrong license
Fatal Error
The error message is generated if required features are not defined in license
file or defined for other host id or expired.

TD0006:  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.

TD0007:  EVENT is defined in <class>
Information
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.

TD0023:  IsrStackSize shall be defined
Error
The IsrStackSize attribute must be defined if the application contains
extended tasks and ISR category 2 or system timer.

For More Information: www.freescale.com
TD0038: At least one OS/TASK/APPMODE shall be defined

Error

At least one object of OS, TASK, and APPMODE types should to be
defined in an application. Only one OS object has to be defined.

TD0045: COUNTER for SysTimer / SecondTimer shall be defined

Error

If the SysTimer/SecondTimer attribute is SWCOUNTER or
HWCOUNTER, then the COUNTER reference shall be defined. The same
counter cannot be attached to both system and second timers.

TD0046: Same COUNTER cannot be used for SysTimer and SecondTimer

Error

The same counter cannot be attached to both system and second timers.

TD0051: 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.

TD0056: ISR category 1 cannot have RESOURCE reference

Error

The RESOURCE reference cannot be defined for ISR object if its
CATEGORY attribute is 1.

TD0057: ISR category 1 cannot use MESSAGE, ACCESSOR attribute shall be
undefined

Error

The message is generated if the CATEGORY attribute of ISR object is set to
1 and ACCESSOR attribute is defined.

TD0058: 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.
TD0059: **ISR cannot send/receive QUEUED MESSAGE**

Error

The *MESSAGE* reference of the ISR object cannot refer to the MESSAGE object with the QUEUED value of the *TYPE* attribute.

TD0060: **Accessor for QUEUED MESSAGE must be with copy only**

Error

If the *MESSAGE* reference of the TASK object refers to the MESSAGE object with the QUEUED value of the *TYPE* attribute, then the *WITHOUTCOPY* attribute of the TASK object cannot be TRUE.

TD0065: **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.

TD0066: **TASK has no EVENT reference**

Information

The referenced task is a basic one.

TD0070: **TASK to be notified shall be defined**

Error

The TASK reference is not defined.

TD0071: **EVENT shall be defined for SETEVENT notification**

Error

The EVENT reference shall be specified only if the *ACTION* attribute of the ALARM and MESSAGE objects is set as SETEVENT.

TD0072: **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.

TD0073: **More than one TASK per priority is defined for <class>**

Warning

Only one task per priority may be used in BCC1 and ECC1 classes, i.e., each task should have unique priority.
TD0085: Static stack size must be defined for Extended task
Error
The task is considered as extended, if any event is referenced. For extended tasks the stack size should be defined. The \texttt{STACKSIZE} attribute of the \texttt{TASK} object should be defined.

TD0086: Basic task doesn't require stack size
Warning
The value of the \texttt{STACKSIZE} attribute is ignored for basic tasks.

TD0098: ACTION = SETEVENT cannot be defined in <class> class
Error
The event setting task notification method cannot be used in Basic conformance classes.

TD0101: Name or ACCESSNAME shall be C-identifier
Error
Only C-identifier can be used in the \textit{Name} or \textit{ACCESSNAME} attributes’ names.

TD0107: 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.

TD0109: 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.

TD0110: MESSAGE is declared but never used
Warning
This message is generated when the MESSAGE object is declared but TASK or ISR objects don't declare accessors to this MESSAGE.
Freescale Semiconductor, Inc.

System Generation Error Messages

List of Messages

TD0111: Accessor name <name> is not unique
Error
The ACCESSNAME attributes should have the different values.

TD0112: Step ordering number <number> is not unique
Error
Two or more StepNumbers share the same number. The StepNumber attributes shall have the unique values.

TD0113: One action has to be specified
Error
Only one ACTION attribute has to be defined for the MESSAGE object in CCCB class.

TD0115: Sum of step times is not equal to ScalePeriod
Warning
The value of the ScalePeriod attribute does not correspond to sum of values of StepTime attributes.

TD0116: One OS/APPMODE shall be defined
Error
One object of OS and at least one APPMODE types shall be defined in an application.

TD0117: TimeScale works only when SysTimer is HWCOUNTER
Error
The TimeScale attribute and its subattributes can be defined only if the SysTimer attribute has HWCOUNTER value.

TD0118: TimeScale cannot work on COUNTER which is referenced by ALARM
Error
Since the system timer and alarm cannot use the same counter in case of TimeScale mechanism using, the SysTimer attribute of OS objects and ALARM objects cannot have the COUNTER attribute with the same value.

For More Information: www.freescale.com
TD0120: ResourceScheduler is FALSE but RES_SCHEDULER is defined. Definition ignored
Warning
RES_SCHEDULER is supported if the ResourceScheduler attribute is TRUE.

TD0123: TASK can have one INTERNAL RESOURCE
Error
The TASK object can have only one reference to the RESOURCE object which has the RESOURCEPROPERTY subattribute with the INTERNAL value.

TD0125: More than one ACCESSOR = SENT is defined for message
Error
The message is generated if there is more than one TASK or ISR defines an accessor to write the same message. By other words, every message might be referred only by one MESSAGE subattribute within ACCESSOR = SENT defined for TASK or ISR object.

TD0126: An ACCESSOR = SENT shall be defined for message
Warning
The message is generated if there are neither TASK nor ISR defines an accessor to write the message. By other words, every message must be referred by one MESSAGE subattribute within the ACCESSOR = SENT defined for TASK or ISR object.

TD0134: At least one StepTime shall be bigger than 0
Error
The message is generated if all Steps in the TimeScale definition has the StepTime values equal to 0.

TD0135: IrqNumber <number> already used
Error
The message is generated if the IRQ <number> is used for more than one ISR definition.
TD0136: IrqNumber <number> is occupied for <TimerName> timer
Error
The message is generated if there is an attempt to use for ISR the IRQ
<number> that is used by System or Second timer.

TD0137: <IrqChannel> hardware is occupied for System/Second timer
Error
The message is generated if there is an attempt to use for ISR the <IrqChannel>
hardware that is used by System or Second timer.

TD0200: This Target-Specific DLL is not intended to work with this platform
Fatal Error
The target-specific DLL does not correspond to target platform.

TD0201: <Target Name> target is not supported by your license!
Fatal Error
The license for target platform is not provided.

TD0202: License <OSEKTURBO> expires in <number> days
Warning
The message warns when the license expires in less than 30 days. The
warning appears every work session.

TD0203: License <OSEKTURBO> will expire tonight at midnight
Warning
This warning appears when the license expires at this day.

TD0250: Attribute <name> is not converted
Warning
The message is generated if the attribute defined for implementation
according to OIL 2.0/2.1 standard, cannot be converted to an attribute
applicable to OIL 2.2 implementation. This attribute does not exist in the
new implementation, and source and target implementations have not
equivalent attribute.
System Generation Error Messages
List of Messages

TD0251: Cannot set to value <value>
Warning
The message is generated if the attribute defined for implementation according to OIL 2.0/2.1 standard, being converted to an attribute applicable to OIL 2.2/2.3 implementation, cannot accept old value. It can be caused out of range for new attribute.

TD0500: ORTIFULL license not found. ORTI file will not be generated
Warning
ORTIFULL license shall be installed for ORTI files generation.

TD0501: 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.

TD0507: RESOURCE is declared but never used
Error
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.

TD0509: 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.

TD0513: <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.
TD0515: 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.

TD0516: 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.

TD0517: Period shall be defined because of Prescaler.Value is AUTO

Error
The Period attribute shall be defined if the Value or/and TimerModuloValue attributes inside SysTimer attribute is AUTO.

TD0518: Period shall be defined because of TimerModuloValue is AUTO

Error
The Period attribute shall be defined if the Value or/and TimerModuloValue attributes inside SysTimer attribute is AUTO.

TD0519: Cannot calculate with defined TimerModuloValue

Error
If prescaler value is AUTO and timer modulo value is defined by the user, then prescaler cannot be calculated during system generation.

TD0520: Prescaler.Values shall be identical for both timers

Error
The Prescaler attribute in the SysTimer attribute scope and the Prescaler attribute inside the SecondTimer attribute shall have the same value.

TD0521: Modulo value cannot fit in range. Period is too long

Error
The message is generated if the modulo value is more than hardware allows for any prescaler value.
TD0522: Modulo value cannot fit in range. Period is too short
   Error
   The message is generated if the modulo value is less than hardware allows
   for any prescaler value.

TD0523: StepTime value is too big
   Error
   Specified value of the StepTime attribute cannot be initialized in hardware.

TD0524: StepTime value cannot be represented precisely in hardware settings
   Error
   The value of StepTime attribute differs from time that can be represented in
   hardware.

TD0525: Specify StepTime value in multiples of <number> nanoseconds
   Information
   The value of the StepTime attribute should be a multiple of number in
   nanoseconds, that is specified by the value of the Period attribute defined
   for the SysTimer attribute with HWCOUNTER value.

TD0526: StepTime value exceeds maximal possible hardware counter period
   Error
   The message is generated when calculated StepTime attribute's value is
   greater than COUNTER/MAXALLOWEDVALUE value which is
   referenced with OS/SysTimer/COUNTER.

TD0527: Hardware counter period shall be at least <number> times greater
   Information
   The message is generated after group of TD0526 error messages and
   specifies required increase of SysTimer/Period or Prescaler/Value
   attribute's value to eliminate TD0526 errors.

TD0740: Cannot find registry key
   Fatal Error
   The Windows Registry is inconsistent.
TD0741:  Cannot find registry value
Fatal Error
The Windows Registry is inconsistent.

TD0750:  EVENT <name> and EVENT <name> share the same bit and used by TASK all together
Warning
Sharing the same bit by event masks inside the task can cause that the task will obtain the unexpected event notification.

TD0752:  EVENT is declared but never used
Warning
The EVENT object is defined in OIL file, but other OIL objects (task, alarms) have no references to this event.

TD0753:  Period is out of representable range
Warning
The message is generated if the period size is more than 32 bits.

TD0754:  Period is defined by prescaler and timer modulo. It will be ignored
Warning
The Period attribute is ignored if Prescaler/Value and TimerModuloValue attributes don't have AUTO value.

TD0755:  The name of OSEK OS property file is not defined explicitly
Warning
The -p option is used to define location of OS property file.

TD0756:  The name of OSEK OS header configuration file is not defined explicitly
Warning
The -h option is used to define the name of OS header file.

TD0757:  The name of OSEK OS source configuration file is not defined explicitly
Warning
The -c option is used to define the name of OS source file.

For More Information: www.freescale.com
TD0758:  The name of OSEK ORTI file is not defined explicitly  
     Warning  
     The -o option is used to define the name of ORTI file.

TD0759:  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.

TD0760:  <filename> shall be used for OSEK OS property file name  
     Information  
     If the location of osprop.h file is not defined explicitly, then by default this file is located in the source OIL file directory.  
     The -p option is used to define the location of OS property file.

TD0761:  <filename> shall be used for OSEK OS header configuration file name  
     Information  
     If the name of header file is not defined explicitly, then by default this file has the same name as the source OIL file, .h extension and located in the same directory.
     The -h option is used to define the name of OS header file.

TD0762:  <filename> shall be used for OSEK OS source configuration file name  
     Information  
     If the name of source file is not defined explicitly, then by default this file has the same name as the source OIL file, .c extension and located in the same directory.
     The -c option is used to define the name of OS source file.

TD0763:  <filename> shall be used for OSEK ORTI file name  
     Information  
     If the name of ORTI file is not defined explicitly, then by default this file has the same name as the source OIL file, .ort extension and located in the same directory.
     The -o option is used to define the name of ORTI file.
TD0765: EVENT is treated as it is used by all the TASKs

Information

The EVENT object is defined in OIL file, but task objects have no references to this event. This leads for allocation of the bit in event mask which cannot be used by other event masks.

TD0766: EVENT is declared but never used and there are no Extended TASKs

Error

The EVENT object is defined in OIL file, but task have no references to this event. The task is considered as extended, if any event is referenced.

TD0767: The version of ORTI specification is not defined explicitly, version 2.1 will be used

Warning

If the version of ORTI file format is not defined explicitly, then by default this file will be generated in format corresponding to version 2.1.

The -O option is used to define the version of ORTI file format.

TD0768: -O option requires either 2.0 or 2.1 version of ORTI specification as a parameter

Error

The current OSEKturbo version supports ORTI specification version 2.0 and 2.1.

TD0775: 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, Target DLL calculates closest period to it.

TD0800: Number of priorities exceeds <number>

Error

For OS/MPC55xx number of priorities is limited by 64.

TD0801: Number of RESOURCEs exceeds <number>

Error

Number of RESOURCE objects defined in system is restricted by 254 (including RES_SCHEDULER).
TD0802: Number of ALARMs exceeds 254
Error
Number of ALARM objects defined in system is restricted by 254.

TD0803: Number of MESSAGEs exceeds 254
Error
Number of MESSAGE objects defined in system is restricted by 254.

TD0804: Number of COUNTERs exceeds 254
Error
Number of COUNTER objects defined in system is restricted by 254.

TD805: 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.

TD806: 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.

TD0807: Number of APPMODEs exceeds 8
Error
Number of APPMODE objects defined in system is restricted by 8.

TD0808: 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.
TD1500:  Prescaler value of <name> timer for <name> MCU cannot be more than <number>
Error
MCU manufacturer reserves values of the prescaler.

TD1501:  Prescaler value 0 means RTI timer is OFF
Error
The timer parameters could not be calculated if there is no clock input to the timer.

TD2251:  InterruptDispatcher can be defined only for derivative
Error
The InterruptDispatcher attribute has to be defined as None if the TargetMCU value corresponds CPU family name - MPC.
Index

A
ACCESSNAME 124, 125
ACCESSOR 124, 125
ACTION 128, 131
ACTIVATION 123
ALARM 128
alarm 68
ALARMCALLBACKNAME 129
ALARMTIME 129
Application configuration file 104
Application Modes 131
APPMODE 123, 129, 131
AUTOSTART 123, 129

B
Basic Task 22, 32
BuildNumber 113

C
CALLBACKNAME 131
CATEGORY 125
CC 113
CDATATYPE 130
Ceiling Priority 59
ClockFrequency 115
Conformance Class 22
conversion constant 66
COUNTER 116, 127, 128
CPU 108
CYCLETIME 129

D
DEBUG_LEVEL 113

E
ERRORHOOK 121
EVENT 123, 126, 129, 131
event 77
Extended Status 26, 98, 232
Extended Task 22, 30

F
FastScheduler 114
FastTerminate 114
FLAGNAME 131

H
HCLowPower 117
hook routines 91

I
IdleLoopHook 121
interrupt dispatcher 119, 153
Interrupt Service Routine (ISR) 47
interrupt stack frame 51
InterruptDispatcher 119
IrqNumber 125
ISR 124
ISRPRIORITY 116
IsrStackSize 118

L
LINKEDRESOURCE 126

M
MASK 127
MAXALLOWEDVALUE 127
MESSAGE 124, 125, 130
MessageCopyAllocation 114
MINCYCLE 127
multilevel interrupt dispatcher 51

O
OIL 104
OS 112
OSEK 15
OSEK Implementation Language 104
OSEK Run Time Interface 229

P
Period 116
POSTTASKHOOK 121
Prescaler 117
PRETASKHOOK 120
PRIORITY 123

OSEKturbo OS/MPC55xx

For More Information: www.freescale.com
Index

Q
QUEUEDEPTH 130

R
ready state 30, 32
RESOURCE 123, 125
RESOURCEPROPERTY 126
ResourceScheduler 114
run time context 30
running state 30, 32

S
ScalePeriod 119
SCHEDULE 123
scheduler 41
SecondTimer 116
SHUTDOWNHOOK 120
Stack Errors 157
StackTraceCheck 118
STACKSIZE 123
Standard OIL version 105
STARTUPHOOK 120
Start-up Routine 99
STATUS 113
Step 119
StepNumber 119
StepTime 119
suspended state 30, 32
System Generator 26, 103
system timer 67
SysTimer 115

T
TargetMCU 115
TASK 119, 122, 129, 131
TICKSPERBASE 127
TimerHardware 117
TimerModuloValue 117
TimeScale 118
TimeUnit 119
TYPE 130

U
Unqueued Messages 83

USEGETSERVICEID 121
USEMESSAGERESOURCE 132
USEMESSAGESTATUS 132
USEPARAMETERACCESS 121

W
waiting state 22, 30, 77
WITHOUTCOPY 124