Freescale AUTOSAR OS/MPC56xxAM v.3.0

User's Manual

Revised: 26 March 2010

Legal Notices

Freescale reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Freescale does not assume any liability arising out of the application or use of any product described herein; neither does it convey any license under its patent rights nor the rights of others. Freescale 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 Freescale product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale 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 Freescale was negligent regarding the design or manufacture of the part.

The information in this document has been carefully checked and is believed to be entirely reliable, however, no responsibility is assumed for inaccuracies. Furthermore, Freescale reserves the right to make changes to any products herein to improve reliability, function or design. Freescale 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 Freescale Semiconductor, Inc. Permission is granted to reproduce and transmit the Problem Report Form and the Customer Satisfaction Survey to Freescale.

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Freescale 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. Freescale 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. Freescale disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

Contents

1 Introduction **5**

Freescale AUTOSAR OS/MPC56xxAM Overview 5

Technical Support Information 6

2 Sample Application **7**

Source Files. 7

Building Sample 8

3 Using an Unsupported Target Derivatives **11**

A Quick Reference **13**

System Services Quick Reference 13

OIL Language Quick Reference 27

OIL attributes names mapping to XML configuration 27

OS Object 27

APPLICATION Object 34

TASK Object. 36

ISR Object. 38

RESOURCE Object 39

EVENT Object 40

COUNTER Object 41

ALARM Object 42

MESSAGE Object 43

APPMODE Object 44

COM Object 45

NM Object 45

SCHEDULETABLE Object 46

1

Introduction

This User's Manual describes Freescale AUTOSAR OS/MPC56xxAM, how to build sample and user's applications. Information about OS services and OIL parameters is provided.

"Sample Application" chapter provides the user with definition of the sample application and instructions how to build the sample application.

"Using an Unsupported Target Derivatives" chapter contains recommendations about OS adaptation to other derivatives.

"Quick Reference" appendix lists OS run-time services with entry and exit conditions as well as OIL object parameters with their possible values and short descriptions.

This chapter consists of the following sections:

- Freescale AUTOSAR OS/MPC56xxAM Overview
- Technical Support Information

Freescale AUTOSAR OS/MPC56xxAM Overview

AUTOSAR Operating System is a real-time operating system which conforms to the AUTOSAR OS v.3.0 specification.

The AUTOSAR OS meets the following requirements:

- OS is fully configured and statically scaled;
- 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 AUTOSAR Operating System feasible for a broad spectrum of applications and hardware platforms.

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

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

Technical Support Information

How to Contact Us:

Corporate Headquarters

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

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

2

Sample Application

The chapter presents the sample application and describes how to build the sample application.

This chapter consists of the following sections:

- Source Files
- Building Sample

The samples provided with OS may be built without AUTOSAR framework and may be used for fist look at the OS.

Source Files

The are 4 sample applications for classes in correspondent directories. The samples root directory (sample\standard) contains command file for Lauterbach debugger ‘56xxAM.cmm’ with support for MPC5634M, MPC5644A and MPC5674F MCUs. Each sample directory (`sample\standard\sc*`) contains makefile for building the sample and following subdirectories:

The standard sample for SC1 is configured to run from internal flash memory, while other samples are configured for internal RAM.

- `src` - contains the sample configuration file and the source code:
  - `samplets<n>.c` – the application code (`TASKSND1`, `TASKSND2` and `TASKCNT`).
  - `samplerv<n>.c` – the application code (`TASKRCV1`, `TASKRCV2` and `TASKSTOP`).
  - `sample.h` – header file for the application code.
  - `sample<n>.oil` – configuration OIL file.
  - `sample<n>.epc` – configuration file in AUTOSAR XML.
- lcf - contains the linker command file ’sample(.lcf,.dld,.ld)’

<n> stands for 1, 2, 3 or 4 according to Scalability Class.

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

Building Sample

Take the following steps to build the sample application:

1. Open the Windows command prompt window.
2. Change the current directory to `sample\standard\sc<n>` directory which contains sample files.
3. To build the sample execute the following command (assuming that GNU make is in the path):

`"make clean"` and then
`"make"` (GrenHills is used by default) or
`"make compiler=diabppc"` for WindRiver compiler
`"make compiler=cwppc"` for CodeWarrior compiler

Please see sample readme for more details

NOTE If some of compiler, AUTOSAR OS or System Generator files are not found during building, check accuracy of the paths defined in the `sample\standard\common.mak` file.

4. After build completion the following subdirectories and files are created in the `sample\standard\sc<n>` directory:

- `output\obj` subdirectory contains object files and configuration files generated by SysGen.
- `output\bin` subdirectory contains the executable file, linker map and ORTI file.
- To execute the sample application load the executable file placed in the bin subdirectory to the SW simulator or evaluation board using the debugger.
- To clean all files generated during the sample building, execute the following command:

```
make clean
```

Please see sample readme file for more details

3

Using an Unsupported Target Derivatives

The chapter contains recommendations for the AUTOSAR OS adaptation to other derivatives.

The current version of the AUTOSAR OS supports MPC5644A, MPC5674F and MPC5634M MCU directly. To use OS on other members of MPC564xA family the *TargetMCU* shall be set to *MPC5644A.* For MPC563xM please use *TargetMCU = MPC5634M*. For MPC567xF please use *TargetMCU = MPC5674F*.

In order to use the OS on other derivative the user should check which timer hardware is available on the MCU if it does not match TargetMCU exactly. If the derivative timers and interrupts structure is similar to supported one, then it is possible to use it with a *TargetMCU* value.

It is not possible to use the OS on derivativies which has different timers configuration - i.e different type of timers, different addresses or different interrupt channels assigned to the timers.

A

Quick Reference

The appendix contains lists of AUTOSAR OS run-time services with entry and exit conditions as well as OIL object parameters with their possible values and short descriptions.

This appendix consists of the following sections:

- System Services Quick Reference
- OIL Language Quick Reference

System Services Quick Reference

The list of all AUTOSAR Operating System run-time services is provided below. Input and output parameters, syntax and ability to use by AUTOSAR entities are shown. Note that ISR means ISR category 2 if not specified else

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
Task management services			
ActivateTask	Task name	–	Task, ISR
	syntax: StatusType ActivateTask(TaskType <TaskID>);		
TerminateTask	–	–	Task
	syntax: StatusType TerminateTask(void);		
ChainTask	Task name	–	Task
	syntax: StatusType ChainTask(TaskType <TaskID>);		
Schedule	–	–	Task
	syntax: StatusType Schedule(void);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
StartScheduleTableRel	Schedule table ID, relative time offset	–	Task, ISR
	syntax: StatusType StartScheduleTableRel(ScheduleTableType sctId, TickType offset);		
StartScheduleTableAbs	Schedule table ID, absolute time offset	–	Task, ISR
	syntax: StatusType StartScheduleTableRel(ScheduleTableType sctId, TickType offset);		
StartScheduleTableSynchron	Schedule table ID	–	Task, ISR
	syntax: StatusType StartScheduleTableRel(ScheduleTableType sctId, GlobalTimeTickType GlobalTime);		
StopScheduleTable	Schedule table ID		Task, ISR
	syntax: StatusType Schedule(ScheduleTableType sctId);		
NextScheduleTable	Current schedule table ID, next schedule table ID		Task, ISR
	syntax: StatusType Schedule(ScheduleTableType cursctId, ScheduleTableType nextsctId)		
SyncScheduleTable	Schedule table ID, current global time		Task, ISR
	syntax: StatusType Schedule(ScheduleTableType sctId, GlobalTimeTickType time)		
GetScheduleTableStatus	Schedule table ID,	Schedule table status	Task, ISR
	syntax: StatusType Schedule(ScheduleTableType sctId, ScheduleTableStatusRefType status)		
SetScheduleTableAsync	Schedule table ID		Task, ISR
	syntax: StatusType Schedule(ScheduleTableType sctId)		
GetTaskID	–	Task name	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskID(TaskRefType <TaskIDRef>);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
GetTaskState	Task name	Task state	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskState(TaskType <TaskID>, TaskStateRefType <StateRef>);		
Interrupt management services			
GetISRID	–	–	Task, ISR, error hook, protection hook
	syntax: void GetISRID(void);		
EnableAllInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void EnableAllInterrupts(void);		
DisableAllInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void DisableAllInterrupts(void);		
ResumeAllInterrupts	–	–	Task, ISR category 1 and 2, alarm-callbacks, ErrorHook, PreTaskHook, PostTaskHook
	syntax: void ResumeAllInterrupts(void);		
SuspendAllInterrupts	–	–	Task, ISR category 1 and 2, alarm-callbacks, ErrorHook, PreTaskHook, PostTaskHook
	syntax: void SuspendAllInterrupts(void);		
ResumeOSInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void ResumeOSInterrupts(void);		
SuspendOSInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void SuspendOSInterrupts(void);		
DisableInterruptSource	ISR which will be disabled	–	Only for SC2, SC3 and SC4 in Task, ISR cat2
	syntax: void DisableInterruptSource(ISRType IsrId);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
EnableInterruptSource	ISR which might be enabled	–	Only for SC2, SC3 and SC4 in Task, ISR cat2
	syntax: void EnableInterruptSource(ISRType IsrId);		
Resource management services			
GetResource	Resource name	–	Task, ISR
	syntax: StatusType GetResource(ResourceType <ResID>);		
ReleaseResource	Resource name	–	Task, ISR
	syntax: StatusType ReleaseResource(ResourceType <ResID>);		
Event control services			
SetEvent	Taks name, Event mask	–	Task, ISR
	syntax: StatusType SetEvent (TaskType <TaskID>, EventMaskType <Mask>);		
ClearEvent	Event mask	–	Extended task
	syntax: StatusType ClearEvent(EventMaskType <Mask>);		
GetEvent	Task name	Event mask	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetEvent(TaskType <TaskID>, EventMaskRefType <Event>);		
WaitEvent	Event mask	–	Extended task
	syntax: StatusType WaitEvent(EventMaskType <Mask>);		
Counter management services			
InitCounter	Counter name, initial value	–	Task, ISR
	syntax: StatusType InitCounter(CtrType <CounterID>, TickType <Ticks>);		
IncrementCounter	Counter name	–	Task, ISR
	syntax: StatusType IncrementCounter(CtrType <CounterID>);		
GetCounterValue	Counter Id	Counter value	Task, ISR,
	syntax: StatusType GetCounterValue(CtrType <CounterID>, TickRefType <TicksRef>);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
GetElapsedCounterValue	Counter Id, previous value	elapsed time	Task, ISR
	syntax: StatusType GetCounterValue(CtrType <CounterID>, TickRefType <previousValue>, TickRefType <TicksRef>);		
GetCounterInfo	Counter name	Counter constants	All except ISR category 1.
	syntax: StatusType GetCounterInfo(CtrType <CounterID>, CtrInfoRefType <InfoRef>);		
Alarm management services			
GetAlarmBase	Alarm name	Alarm constants	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetAlarmBase(AlarmType <AlarmID>, AlarmBaseRefType <InfoRef>);		
GetAlarm	Alarm name	Relative value in ticks before the alarm expires	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetAlarm(AlarmType <AlarmID>, TickRefType <TicksRef>);		
SetRelAlarm	Alarm name, Counter relative value, Cycle value	–	Task, ISR
	syntax: StatusType SetRelAlarm (AlarmType <AlarmID>, TickType <Increment>, TickType <Cycle>);		
SetAbsAlarm	Alarm name, Counter absolute value, Cycle value	–	Task, ISR
	syntax: StatusType SetAbsAlarm (AlarmType <AlarmID>, TickType <Start>, TickType <Cycle>);		
CancelAlarm	Alarm name	–	Task, ISR
	syntax: StatusType CancelAlarm(AlarmType <AlarmID>);		
<AlarmCallBack>[a]	–	–	–
	syntax: ALARMCALLBACK(<CallbackName>);		
Message management services			
SendMessage	Message name, message data	–	Task, ISR, COM callback
	syntax: StatusType SendMessage(MessageIdentifier <Message>, ApplicationDataRef <Data>);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
ReceiveMessage	Message name	Message data	Task, ISR, COM callback
	syntax: StatusType ReceiveMessage(MessageIdentifier <Message>, ApplicationDataRef <Data>);		
ReadFlag	–	–	Task, ISR, COM callback
	syntax: FlagValue ReadFlag_<FlagName>(void);		
ResetFlag	–	–	Task, ISR, COM callback
	syntax:StatusType ResetFlag_<FlagName>(void);		
GetMessageStatus	Message name	–	Task, ISR, COM callback
	syntax: StatusType GetMessageStatus (MessageIdentifier <Message>);		
StartCOM	–	–	Task, ISR
	syntax: StatusType StartCOM (COMApplicationModeType Mode);		
StopCOM	–	–	Task, ISR, COM callback
	syntax: StatusType StopCOM (COMShutdownModeType ShutdownMode);		
GetCOMApplicationMode	–	–	Task, ISR, COM callback
	syntax: COMApplicationModeType GetCOMApplicationMode (void);		
InitMessage	Message data	–	Task, ISR, COM callback
	StatusType InitMessage (MessageIdentifier Message, ApplicationDataRef DataRef		
<MessageCallBack>[b]	–	–	Task, ISR, COM callback
	syntax: void <CallbackName> (void);		
Debugging services			
GetRunningStackUsage	–	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetRunningStackUsage(void);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
GetStackUsage	Task name	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetStackUsage(TaskType <TaskID>);		
GetTimeStamp	–	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetTimeStamp (void);		
Execution control services			
GetActiveApplicationMode	–	Current application mode	Task, ISR, All hooks
	syntax: AppModeType GetActiveApplicationMode(void);		
StartOS	Application mode name	–	Outside of OS
	syntax: void StartOS(AppModeType <Mode>);		
ShutdownOS	Error code	–	Task, ISR, StartupHook, ErrorHook
	syntax: void ShutdownOS(StatusType <Error>);		
Hook Routines			
ErrorHook	Error code	–	–
	syntax: void ErrorHook(StatusType <Error>);		
PreTaskHook	–	–	–
	syntax: void PreTaskHook(void);		
PostTaskHook	–	–	–
	syntax: void PostTaskHook(void);		
StartupHook	–	–	–
	syntax: void StartupHook(void);		
ShutdownHook	Error code	–	–
	syntax: void ShutdownHook(StatusType <Error>);		
ProtectionHook	Error code	Action code	–
	syntax: ProtectionReturnType ProtectionHook(StatusType);		

Table A.1 AUTOSAR OS Services

Service	Input	Output	Allowed In
PreIsrHook	–	–	–
	syntax: void PreTaskHook(void);		
PostIsrHook	–	–	–
	syntax: void PreTaskHook(void);		

a. <AlarmCallBack> is the value of the ALARMCALLBACKNAME attribute defined in ALARM object. The user can have several alarm callback functions, one for each alarm defined in the OIL file.

b. <MessageCallBack> is the value of the CALLBACKROUTINENAME attribute defined in MESSAGE object. The user can have several message callback functions, one for each message defined in the OIL file.

NOTE InitCounter, GetCounterInfo, GetRunningStackUsage, GetStackUsage, PreIsrHook, PostIsrHook and GetTimeStamp services are not defined in the AUTOSAR OS v.3.0 specification. It is Freescale OS extension of the AUTOSAR OS.

The list of macros for parameter access from *ErrorHook* routine is provided below.

Table A.2 AUTOSAR Macros for ErrorHook

Macro	Return Value
OSErrorGetServiceId()	Service identifier
OSError_StartOS_Mode()	Application mode
OSError_ActivateTask_TaskID()	Task identifier
OSError_ChainTask_TaskID()	Task identifier
OSError_GetTaskState_TaskID()	Task identifier
OSError_GetResource_ResID()	Resource identifier
OSError_ReleaseResource_ResID()	Resource identifier
OSError_SetEvent_TaskID()	Task identifier
OSError_GetEvent_TaskID()	Task identifier
OSError_SendMessage_Message()	Message identifier
OSError_ReceiveMessage_Message()	Message identifier
OSError_GetMessageStatus_Message()	Message identifier
OSError_GetAlarmBase_AlarmID()	Alarm identifier
OSError_GetAlarm_AlarmID()	Alarm identifier

Table A.2 AUTOSAR Macros for ErrorHook

Macro	Return Value
OSErrorGetServiceId()	Service identifier
OSError_StartOS_Mode()	Application mode
OSError_SetRelAlarm_AlarmID()	Alarm identifier
OSError_SetAbsAlarm_AlarmID()	Alarm identifier
OSError_CancelAlarm_AlarmID()	Alarm identifier
OSError_InitCounter_CounterID()[a]	Counter identifier
OSError_CounterTrigger_CounterID()[a]	Counter identifier
OSError_GetCounterValue_CounterID()[a]	Counter identifier
OSError_GetCounterInfo_CounterID()[a]	Counter identifier

a. Counter interface functions are not defined in AUTOSAR OS v.3.0 specification, this is Freescale OS extension of the AUTOSAR OS.

The list of AUTOSAR Operating System Data Types is provided below.

Table A.3 Data Types

Data Type	Description
ScheduleTableType	- identifier of ScheduleTable. - the status of a schedule table; . ; GlobalTimeTickType - the global time source type.
ScheduleTableStatusType	The data type for schedule table status
ScheduleTableStatusRefType	The data type references a schedule table status
GlobalTimeTickType	The data type for global time source
AlarmBaseRefType	The data type references data corresponding to the data type *AlarmBaseType*
AlarmBaseType	The data type represents a structure for storage of alarm characteristics. It is the same as *CtrInfoType*
AlarmType	The data type represents an alarm element
AppModeType	This data type represents the operating mode
CtrInfoRefType	The data type references data corresponding to the data type *CtrInfoType*

Table A.3 Data Types

Data Type	Description
CtrInfoType	The data type represents a structure for storage of counter characteristics. This structure has the following fields: *maxallowedvalue* maximum possible allowed count value; *ticksperbase* number of ticks required to reach a counter-specific significant unit (it is the user constant not used by OS); *mincycle* minimum allowed number of ticks for a cyclic alarm (only for a system with Extended Status);
CtrType	The data type references a counter
PhysicalTimeType	The data type for value returned by OS_TICKS2<Unit>_<Counter> macro
EventMaskRefType	The data type to refer to an event mask
EventMaskType	The data type of an event mask
FlagType	The data type of a message flag
ResourceType	The abstract data type for referencing a resource
StatusType	The data type for all status information the API services offer
SymbolicName	A unique name representing a message
TaskRefType	The data type to refer variables of the *TaskType* data type
TaskStateRefType	The data type to refer variables of the *TaskStateType* data type
TaskStateType	The data type for variables to store the state of a task
TaskType	The abstract data type for task identification
TickRefType	The data type references data corresponding to the data type *TickType*
TickType	The data type represents a counter value in system ticks
ISRType	The abstract data type for ISR identification

NOTE CtrInfoType and CtrInfoRefType data types are not defined in the AUTOSAR OS v.3.0 specification. This is Freescale OS extension of the AUTOSAR OS.

The DeclareISR(<ISR name>) constructional element is defined in Freescale AUTOSAR OS, it is intended to use in the vector.c file.

The table below contains all return values for the AUTOSAR Operating System run-time services and error values.

Table A.4 Services Return and Error Values

Name	Value	Type
E_OK	0	No error, successful completion
E_OS_ACCESS	1	Access to the service/object denied
E_OS_CALLEVEL	2	Access to the service from the ISR is not permitted
E_OS_ID	3	The object ID is invalid
E_OS_LIMIT	4	The limit of services/objects exceeded
E_OS_NOFUNC	5	The object is not used, the service is rejected
E_OS_RESOURCE	6	The task still occupies the resource
E_OS_STATE	7	The state of the object is not correct for the required service
E_OS_VALUE	8	A value outside of the admissible limit
E_OS_SERVICEID	9	Service can not be called
E_OS_RATE	10	Task activation rate exceeds limit
E_OS_ILLEGAL_ADDRESS	11	An invalid address is given as a parameter to a service
E_OS_MISSINGEND	12	Tasks terminates without a TerminateTask() or ChainTask() call
E_OS_DISABLEDINT	13	OS service is called inside an interrupt disable/enable pair
E_OS_STACKFAULT	14	Stack fault detected via stack monitoring by the OS
E_OS_PROTECTION_MEMORY	15	Memory access violation occurred
E_OS_PROTECTION_TIME	16	Task/Category 2 ISR exceeds its execution time budget
E_OS_PROTECTION_LOCKED	17	Task/Category 2 ISR exceeds Resource or ISRs lock time
E_OS_PROTECTION_EXCEPTION	18	Trap occurred
E_OS_PROTECTION_ARRIVAL	19	Task/Category 2 ISR arrives before its timeframe has expired
E_OS_SYS_FATAL	21	Fatal error in the OS code
E_OS_SYS_ORDER [a]	23	Incorrect order of function calling
E_COM_ID	35	Invalid message name passed as parameter
E_COM_LIMIT	36	Overflow of FIFO associated with queued messages
E_COM_NOMSG	41	No message available

[a.] E_OS_SYS_xxx codes are defined in the AUTOSAR OS v.3.0 specification as implementation specific. This is Freescale OS extension of the AUTOSAR OS.

The list of service identifiers for *ErrorHook* is provided below:

- identifieirs for standard AUTOSAR services

  OSServiceId_StartOS

  OSServiceId_ShutdownOS

  OSServiceId_GetActiveApplicationMode

  OSServiceId_ActivateTask

  OSServiceId_TerminateTask

  OSServiceId_ChainTask

  OSServiceId_Schedule

  OSServiceId_GetTaskID

  OSServiceId_GetTaskState

  OSServiceId_ResumeAllInterrupts

  OSServiceId_SuspendAllinterrupts

  OSServiceId_ResumeOSInterrupts

  OSServiceId_SuspendOSinterrupts

  OSServiceId_EnableAllInterrupts

  OSServiceId_DisableAllInterrupts

  OSServiceId_GetResource

  OSServiceId_ReleaseResource

  OSServiceId_SetEvent

  OSServiceId_ClearEvent

  OSServiceId_GetEvent

  OSServiceId_WaitEvent

  OSServiceId_SendMessage

  OSServiceId_ReceiveMessage

  OSServiceId_GetMessageStatus

  OSServiceId_StartCOM

  OSServiceId_StopCOM

  OSServiceId_GetAlarmBase

OSServiceId_GetAlarm

OSServiceId_SetRelAlarm

OSServiceId_SetAbsAlarm

OSServiceId_CancelAlarm

OSServiceId_GetCounterValue

OSServiceId_GetElapsedCounterValue

OSServiceId_GetCounterInfo

- identifiers for Freescale AUTOSAR OS specific services

  OSServiceId_InitCounter

- identifier returned if the error occured not in the OS service called by the user but inside OS dispatcher

  OSServiceId_NoService

The following table contains AUTOSAR Operating System constants with short descriptions.

Table A.5 AUTOSAR OS Constants

Constant	Value	Description
RUNNING	0	Constant of data type *TaskStateType* for task state *running*
WAITING	1	Constant of data type *TaskStateType* for task state *waiting*
READY	2	Constant of data type *TaskStateType* for task state *ready*
SUSPENDED	3	Constant of data type *TaskStateType* for task state *suspended*

Table A.5 AUTOSAR OS Constants

Constant	Value	Description
INVALID_TASK	Depends on the OS configuration.	Constant of data type *TaskType* for not defined Task
INVALID_ISR		Constant of data type *ISRType* for not defined ISR
RES_SCHEDULER		Constant of data type *ResourceType* for *Scheduler* as a resource
OSMAXALLOWEDVALUE		Maximum possible allowed system counter value
OSMAXALLOWEDVALUE2		Maximum possible allowed second counter value
OSTICKSPERBASE		Number of ticks required to reach a counter-specific value in the system counter (it is the user constant not used by OS)
OSTICKSPERBASE2		Number of ticks required to reach a counter-specific value in the second counter (it is the user constant not used by OS)
OSTICKDURATION		Duration of the system counter tick in nanoseconds
OSTICKDURATION2		Duration of the second counter tick in nanoseconds
OSMINCYCLE		Minimum allowed number of ticks for a cyclic alarm attached to the system counter (only for a system with Extended Status)
OSMINCYCLE2		Minimum allowed number of ticks for a cyclic alarm attached to the second counter (only for a system with Extended Status)
OSDEFAULTAPPMODE		Default application mode. This constant is always a valid parameter for *StartOS* service
<IsrName>PRIORITY		The value of this constant is equal to *PRIORITY* of ISR <IsrName>.
OSBUILDNUMBER	Current build number	Current build number in ASCII, for example 2.1.10

NOTE OSMAXALLOWEDVALUE2, OSTICKSPERBASE2, OSTICKDURATION2, OSMINCYCLE2, <IsrName>PRIORITY and OSBUILDNUMBER constants are not defined in the AUTOSAR OS v.3.0 specification. This is Freescale OS extension of the AUTOSAR OS.

OIL Language Quick Reference

The lists of all the OIL object parameters with their possible values and short descriptions are provided here. All standard object attributes must be always defined. Freescale AUTOSAR OS specific attributes can be defined in addition to standard ones. The value used by default is typed in boldface in the *Possible Values* cells.

Memory consumption and performance trends based on influence of individual attributes are signed in the *Possible Values* cells. There are three signs put next to the attribute values (exclude default value). They display variation of RAM usage, ROM usage and execution TIME (first, second and third sign respectively) compared to the default attribute value. Symbol “+” corresponds to increasing RAM, ROM or TIME, Symbol “–” corresponds to decreasing RAM, ROM and TIME and symbol “±” designates “no change”.

OIL attributes names mapping to XML configuration

The AUTOSAR prescribes usage of XML notation for the OS and other modules configuration. In general, the OIL names are mapped to XML ones by de-capitalizing and adding a prefix "Os" to the attribute names. For exact mapping please see chapter "*System Objects Definition"* in Technical Reference.

OS Object

The OS object is the mandatory one for any application. It defines the OS and its properties for the application. The OS attributes exactly correspond to the system options and are divided into parts corresponding to appropriate system objects. The standard and Freescale AUTOSAR OS

specific attributes of the OS object are marked by the "standard" and "specific" respectively.

Table A.6 OS Parameters

Object Parameters	Possible Values	Description
Global System Attributes		This group of OS 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:
`STATUS = <STANDARD / EXTENDED>;`
`STARTUPHOOK = <TRUE / FALSE>;`
`ERRORHOOK = <TRUE / FALSE>;`
`SHUTDOWNHOOK = <TRUE / FALSE>;`
`PRETASKHOOK = <TRUE / FALSE>;`
`POSTTASKHOOK = <TRUE / FALSE>;`
`IsrHooks = <TRUE / **FALSE**>;`
`USEGETSERVICEID = <TRUE / FALSE>;`
`USERPARAMETERACCESS = <TRUE / FALSE>;`
`USERRESSCHEDULER = <TRUE / FALSE>;`
`PROTECTIONHOOK = <TRUE / FALSE>;`
`SCALABILITYCLASS = <SC1 / SC2 /SC3 /SC4>;`
`IsrStackSize = <integer>;`
`STACKMONITORING = = <TRUE / FALSE>{`
`Pattern = <**0x55555555** / integer>;`
`PatternSize = <**1** / 2 / 4>;`
`};`		

STATUS standard	STANDARD, EXTENDED (+,+,+)	This standard attribute specifies OS debug status
STARTUPHOOK standard	TRUE, FALSE	Defines whether or not StartupHook is called at the start of OS
ERRORHOOK standard	TRUE, FALSE	Defines whether or not ErrorHook is called when an error is detected
SHUTDOWNHOOK standard	TRUE, FALSE	Defines whether or not ShutdownHook is called when OS is shut down.
PRETASKHOOK standard	TRUE, FALSE	Defines whether or not PretaskHook is called before running a task
POSTTASKHOOK standard	TRUE, FALSE	Defines whether or not PosttaskHook is called after finishing a task

Table A.6 OS Parameters

Object Parameters	Possible Values	Description
IsrHooks specific	TRUE, **FALSE**	Defines whether or not Pre/PostIsrHook is called before and after the ISRs
USEGETSERVICEID standard	TRUE, FALSE	Defines whether or not macro to get service ID is provided for ErrorHook
USEPARAMETERACCESS standard	TRUE, FALSE	Defines whether or not macros to get the first parameter of the service is provided for ErrorHook.
USERESSCHEDULER standard	TRUE, FALSE	defines whether or not the RES_SCHEDULER is used in OS. If it is set to FALSE then RES_SCHEDULER support is excluded.
PROTECTIONHOOK standard	TRUE, FALSE	Defines whether or not PROTECTIONHOOK is called on protection error.
SCALABILITYCLASS standard	SC1, SC2, SC3, SC4	Defines the scalability class of OS
STACKMONITORING standard	TRUE, FALSE	Defines whether or not OS monitors stack
Pattern specific	**0x55555555,** integer	Specifies a 32-bit pattern to fill stack
PatternSize specific	**1**, 2, 4	defines the size in 32-bit words of area to be checked

Table A.6 OS Parameters

Object Parameters	Possible Values	Description
CPU Related Attributes		This group of OS attributes provides possibility to tune the selected hardware

Table A.6 OS Parameters

Object Parameters	Possible Values	Description

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:

```
TargetMCU = <name of MCU> {
    InternalROM = <TRUE / FALSE> {
      Address = <integer / 0x00000000>;
      Size    = <integer / 0x200000>;
    };
    InternalRAM = <TRUE / FALSE> {
      Address = <integer / 0x40000000>;
      Size    = <integer / 0x10000[a]>;
    };
    ExternalROM = <TRUE / FALSE> {
      Address = <integer>;
      Size    = <integer>;
    };
    ExternalRAM = <TRUE / FALSE> {
      Address = <integer>;
      Size    = <integer>;
    };
    ClockFrequency = <integer / 8000>;
    ClockDivider = <integer / 1>;
    ClockMultiplier = <integer / 1>;
    TPTimer = <TRUE / FALSE> {
        Period = <integer / AUTO>;
        TimerHardware = STM {
          Prescaler = <USER / OS> {
            Value = <integer / AUTO>;
        };
    };
    SysTimer = <HWCOUNTER / SWCOUNTER / NONE> {
        COUNTER = <name of COUNTER>;
        ISRPRIORITY = <integer>;
        Period = <integer / AUTO>;
        TimerHardware = <name of timer hardware> {
          Prescaler = <USER / OS> {
            Value = <integer / AUTO>;
          };
          Channel = <integer>;
          TimerModuloValue = <integer / AUTO>;
          PeripheralClockDivider = <integer / 1>;
        };
    };
```

Table A.6 OS Parameters

Object Parameters	Possible Values	Description

SecondTimer = <HWCOUNTER / SWCOUNTER / **NONE**> {
COUNTER = <name of COUNTER>;
ISRPRIORITY = <integer>;
Period = <integer / **AUTO**>;
TimerHardware = <name of timer hardware> {
Prescaler = <USER / OS> {
Value = <integer / **AUTO**>;
};
Channel = <integer>;
TimerModuloValue = <integer / **AUTO**>;
PeripheralClockDivider = <integer / 1>;
};
};
};		

TargetMCU specific	MPC5634M, MPC5644A, MPC5674F	Specifies target MCU type
ClockFrequency specific	integer	Specifies oscillator frequency in kHz (used for calculating prescaler value and timer modulo value)
InternalROM specific	**TRUE**, FALSE	specifies is internal ROM used by the OS or user application.
InternalRAM specific	**TRUE**, FALSE	specifies is internal RAM used by the OS or user application.
ExternalROM specific	TRUE, **FALSE**	specifies is external ROM used by the OS or user application.
ExternalRAM specific	TRUE, **FALSE**	specifies is external RAM used by the OS or user application.
Address specific	integer	specifies the address of memory area.
Size specific	integer	specifies the size of memory area.
ClockDivider specific	integer	Specifies the divider value that is set by the User in PLL.
ClockMultiplier specific	integer	Specifies the multiplier value that is set by the User in PLL.

Table A.6 OS Parameters

Object Parameters	Possible Values	Description
TPTimer specific	TRUE, FALSE	Defines whether the Timer Protection Timer is used
SysTimer specific	HWCOUNTER, SWCOUNTER, NONE	Defines whether the internal OS system timer is used or not.
SecondTimer specific	HWCOUNTER, SWCOUNTER, NONE	Defines whether the internal OS second timer is used or not.
COUNTER specific	name of COUNTER	Specifies the COUNTER which shall be attached to the system or second timer. The same counter can not be attached to the System and Second timers
Period specific	integer AUTO	Specifies period of a tick of the system (second) counter in nanoseconds
TimerHardware specific	STM, PIT, RTI, eMIOS	The attribute is intended to select the hardware interrupt source for the TP timer, System and Second Timers.
Channel specific	integer	Specifies the timer HW channel to be used for OS timer
TimerModuloValue specific	integer, **AUTO**	Specifies timer hardware register value
PeripheralClockDivider specific	integer	Specifies the peripheral clock divider for peripheral frequency calculating. It is defined only for MPC5674F
Interrupt Related Properties		This group of OS attributes defines parameters of ISR execution
IsrStackSize specific	integer	Specifies ISR stack size. It shall be defined if there are ISR category 2 in SC2 and in SC1 if CC = ECC1
Hook Routines Related Attributes		This group of OS attributes defines additional hook routines support in the system

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`STARTUPHOOK = <TRUE / FALSE>;`
`SHUTDOWNHOOK = <TRUE / FALSE>;`
`PRETASKHOOK = <TRUE / FALSE>;`
`POSTTASKHOOK = <TRUE / FALSE>;`
`ERRORHOOK = <TRUE / FALSE>;`
`USEGETSERVICEID = <TRUE / FALSE>;`
`USEPARAMETERACCESS = <TRUE / FALSE>;`		

Table A.6 OS Parameters

Object Parameters	Possible Values	Description
STARTUPHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether StartupHook is called after the operating system starting up and before the dispatcher starting or not
SHUTDOWNHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether ShutdownHook is called during the system shutdown or not
PRETASKHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether PreTaskHook is called from the scheduler code before the operating system enters context of the task or not
POSTTASKHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether the PostTaskHook is called from the scheduler code after the operating system leaves the context of the task or not
ERRORHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether the ErrorHook is called by the system at the end of each system service which returns the status not equal to E_OK or not
USEGETSERVICEID standard	TRUE (+,+,+) FALSE	Specifies ability of usage the access macros to the service ID in the error hook
USEPARAMETERACCESS standard	TRUE (+,+,+) FALSE	Specifies ability of usage the access macros to the context related information in the error hook

[a.] default value depends on TargetMCU

APPLICATION Object

Parameters of APPLICATION object type define the task properties. The syntax of the application object definition is as follows:

```
APPLICATION <name of APPLICATION>{
  MemData0 = <TRUE / FALSE>;
  MemData1 = <TRUE / FALSE>;
  MemData2 = <TRUE / FALSE>;
  TRUSTED = <FALSE / TRUE>{
      TRUSTED_FUNCTION = <TRUE / FALSE>{
        NAME = <name of FUNCTION>;
      };
  };
  STARTUPHOOK = <TRUE / FALSE>;
  SHUTDOWNHOOK = <TRUE / FALSE>;
```

```
  ERRORHOOK = <TRUE / FALSE>;
  HAS_RESTARTTASK = <TRUE / FALSE>{
      RESTARTTASK = <name of TASK>
  };
  TASK = <name of TASK>;
  ISR = <name of ISR>;
  ALARM = <name of ALARM>;
  SCHEDULETABLE = <name of SCHEDULETABLE>;
  COUNTER = <name of COUNTER>;
  RESOURCE = <name of RESOURCE>;
  MESSAGE = <name of MESSAGE>;
};
```

The brief description of the OS_Apllication attributes is presented below.

Table A.7 APPLICATION Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
TRUSTED standard	TRUE, FALSE	Defines whether the OS-Application is trusted or not
TRUSTED_FUNCTION standard	TRUE, FALSE	Defines whether the trusted OS-Application has a trusted functions.
NAME standard	<function name>	Defines the name of the trusted function
STARTUPHOOK standard	TRUE, FALSE	Defines whether the application-specific hook (StartupHook_<App>) is called by the system.
SHUTDOWNHOOK standard	TRUE, FALSE	Defines whether the application-specific hook (ShutdownHook_<App>) is called by the system.
ERRORHOOK standard	TRUE, FALSE	Defines whether the application-specific hook (ErrorHook_<App>) is called by the system.
HAS_RESTARTTASK standard	TRUE, FALSE	Defines whether the OS-Application has a “restart” TASK wich is called at the OS-Application restart.
TASK standard	name of TASK	Reference a task owned by the OS-Application.
ISR standard	name of ISR	Reference an ISR owned by the OS-Application.

Table A.7 APPLICATION Parameters

Object Parameters	Possible Values	Description
ALARM standard	name of ALARM	Reference an alarm owned by the OS-Application.
SCHEDULETABLE standard	name of SCHEDULETABLE	Reference a scheduletable owned by the OS-Application.
COUNTER standard	name of COUNTER	Reference a counter owned by the OS-Application.
RESOURCE standard	name of RESOURCE	Reference a resource owned by the OS-Application.
MESSAGE standard	name of MESSAGE	Reference a message owned by the OS-Application.
Freescale AUTOSAR OS Specific Attribute		
MemData<0/1/2>	TRUE, FALSE	Defines which RAM areas are used by the OS-Application.

TASK Object

Parameters of TASK object type define the task properties. The syntax of the task object definition is as follows:

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

```
            LOCKINGTIME = <RESOURCELOCK>{
                RESOURCE = <name of RESOURCE>;
                MAXRESOURCELOCKTIME = <integer>;
            };
        ACCESSING_APPLICATION = <name of Application>;
};
```

The brief description of the task attributes is presented below.

Table A.8 TASK Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
PRIORITY	integer [0..0xFFFFFFFF]	Defines the priority of the task. The lowest priority has value 0
SCHEDULE	FULL, NON	Defines the run-time behavior of the task
AUTOSTART	TRUE, FALSE	Defines whether the task is activated during the system start-up procedure or not
APPMODE	name of APPMODE	Defines an application mode in which the task is auto-started
ACTIVATION	1	Specifies the maximum number of queued activation requests for the task. The Freescale AUTOSAR OS OS does not support multiple activation, so this value is restricted to 1
RESOURCE	name of RESOURCE	Resources accessed by the task. There can be several resource references
EVENT	name of EVENT	Events owned by the task. There can be several event references
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the task
TIMING_PROTECTION	TRUE, FALSE	Defines if timing protection is enabled for the task
EXECUTIONBUDGET	integer [0..0xFFFFFFFFFFFFFFFF]]	Contains the worst case execution time in nanoseconds for each task on a given TIMEFRAME
TIMEFRAME,	integer [0..0xFFFFFFFFFFFFFFFF]]	Defines a length of time frame in nanoseconds.
LOCKINGTIME	RESOURCELOCK	Contains a RESOURCE and its MAXRESOURCELOCKTIME attributes.
MAXRESOURCELOCKTIME	integer [0..0xFFFFFFFFFFFFFFFF]	Defines the worst case time between getting and releasing a given resource

Table A.8 TASK Parameters

Object Parameters	Possible Values	Description
MAXOSINTERRUPTLOCKTIME	integer [0..0xFFFFFFFFFFFFFFFF]]	Defines the worst time (in nanoseconds) the task locks out interrupts of category 2
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access thisTask
Freescale AUTOSAR OS Specific Attribute		
STACKSIZE	integer	Defines the size of the Task's stack in bytes

ISR Object

This object represents an Interrupt Service Routine. Parameters of this object type define ISR properties. The syntax of the ISR object is as follows:

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

The following parameters can be defined for the ISR object:

Table A.9 ISR Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
CATEGORY	1, 2	Specifies the category of interrupt service routine
RESOURCE	name of RESOURCE	Specifies the list of resources accessed by the task. The reference can not be defined if *CATEGORY* is 1.There can be several resource references
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the ISR
PRIORITY	[1..15]	Specifies the priority of the interrupt service routine
TIMING_PROTECTION	TRUE, FALSE	Defines if timing protection is enabled for the ISR
EXECUTIONTIME	integer [0..0xFFFFFFFFFFFFFFFF]]	Contains the worst case execution time in nanoseconds for each ISR
TIMEFRAME	integer [0..0xFFFFFFFFFFFFFFFF]]	Defines a length of time frame in nanoseconds.
LOCKINGTIME	RESOURCELOCK	Contains a RESOURCE and its MAXRESOURCELOCKTIME attributes.
MAXRESOURCELOCKTIME	integer [0..0xFFFFFFFFFFFFFFFF]	Defines the worst case time between getting and releasing a given resource
MAXOSINTERRUPTLOCKTIME	integer [0..0xFFFFFFFFFFFFFFFF]]	Defines the worst time (in nanoseconds) the ISR locks out interrupts of category 2
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this ISR
Freescale AUTOSAR OS Specific Attributes		
STACKSIZE	integer	Defines the size of the ISR's stack in bytes
FunctionName	string	Specifies the name of ISR handler function. By default it is the name of ISR object itself.
IrqChannel	EXTERNAL	Specifies the hardware interrupt channel
IrqNumber	[0..500]	Specifies the interrupt source for the EXTERNAL *IrqChannel*.

RESOURCE Object

The RESOURCE object is intended for the resource management. The syntax of the resource object is as follows:

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

The following standard parameters can be defined for the RESOURCE object:

Table A.10 RESOURCE Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
RESOURCEPROPERTY	STANDARD, LINKED, INTERNAL	Specifies a property of the resource. Performance decreases if RESOURCE with RESOURCEPROPERTY = INTERNAL defined
LINKEDRESOURCE	name of RESOURCE	Specifies the resource to which the linking shall be performed
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this Resource

EVENT Object

The EVENT object is intended for the event management. The syntax of the event object is as follows:

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

The following standard parameters can be defined for the EVENT object:

Table A.11 EVENT Parameters

Object Parameters	Possible Values	Description
Standard Attribute		
MASK	integer, **AUTO**	Represents the event

COUNTER Object

Attributes of this object type define counter properties. The syntax of the counter object is:

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

The following standard parameters can be defined for the COUNTER object:

Table A.12 COUNTER Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
MINCYCLE	integer	Specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter
MAXALLOWEDVALUE	integer	Defines the maximum allowed counter value
TICKSPERBASE	integer	Specifies the number of ticks required to reach a counter-specific value (it is the user constant not used by OS)
TYPE	SOFTWARE, HARDWARE	Defines the type of the counter
DRIVER	OSINTERNAL	Defines timer driver
NS	integer	Defines constant value
CONSTNAME	name of constant	Defines the name of constant with value in ticks
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this COUNTER

ALARM Object

This object presents OS alarms. The syntax of an alarm object is as follows.

```
ALARM <name of ALARM> {
        COUNTER = <name of COUNTER>;
        ACTION = <SETEVENT / ACTIVATETASK / ALARMCALLBACK /
INCREMENTCOUNTER> {
            TASK = <name of TASK>;
            EVENT = <name of EVENT>;
            ALARMCALLBACKNAME = <string>;
            COUNTER = <name of COUNTER>
        };
        AUTOSTART = <TRUE / FALSE> {
            ALARMTIME = <integer>;
            CYCLETIME = <integer>;
            APPMODE = <name of APPMODE>;
        };
        ACCESSING_APPLICATION = <name of Application>;
};
```

The following standard parameters can be defined for the ALARM object:

Table A.13 ALARM Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
COUNTER	name of COUNTER	Specifies the assigned counter
ACTION	ACTIVATETASK, SETEVENT, ALARMCALLBACK, INCREMENTCOUNTER	Defines the method of notification used when the alarm expires
TASK	name of TASK	Specifies the task being notified through activation or event setting when the alarm expires
EVENT	name of EVENT	Specifies the event mask to be set when the alarm expires. It shall be defined if ACTION is SETEVENT only
ALARMCALLBACKNAME	string	Specifies the name of the callback routine called when the alarm expires

Table A.13 ALARM Parameters

Object Parameters	Possible Values	Description
COUNTER (inside ACTION)	name of COUNTER	Specifies the name of the COUNTER to be incremented when the alarm expires
AUTOSTART	TRUE, FALSE	Defines whether an alarm is started automatically at system start-up depending on the application mode
ALARMTIME	integer	Defines the time when the alarm shall expire first
CYCLETIME	integer	Defines the cycle time of a cyclic alarm
APPMODE	name of APPMODE	Defines an application mode in which the alarm will be started automatically at system start-up
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this ALARM

MESSAGE Object

Parameters of this object type define the message properties. The syntax of the message object definition is presented below. Note that only one *ACTION* attribute should be defined for the MESSAGE object.

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

The following standard parameters can be defined for the MESSAGE object:

Table A.14 MESSAGE Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
MESSAGEPROPERTY	SEND_STATIC_INTERNAL, RECEIVE_UNQUEUED_INTERNAL, RECEIVE_QUEUED_INTERNAL	Specifies the message type
SEMDINGMESSAGE	name of MESSAGE	Specifies which message transfers data to this (received) one
QUEUESIZE	integer	Specified if the message has a queue
CDATATYPE	string	Defines the data type of a message item
NOTIFICATION	ACTIVATETASK, SETEVENT, COMCALLBACK, FLAG, NONE	Defines the type of task notification used when the message has arrived
TASK	name of TASK	Specifies the task which shall be notified when the message has arrived. It shall be defined if *ACTION* is *ACTIVATETASK* or *SETEVENT* only
EVENT	name of EVENT	Specifies the event to be set when the message has arrived. It shall be defined if *ACTION* is *SETEVENT* only
CALLBACKROUTINE NAME	string	Defines the name of a function to call as an action when the message has been sent. It shall be defined if *ACTION* is *CALLBACK* only. It is valid only for SC1
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the callback function
FLAGNAME	string	Defines the name of the flag that is set when the message is sent. It shall be defined if *ACTION* is *FLAG* only
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this ScheduleTable

APPMODE Object

The APPMODE object is intended for the application mode management. This object has no standard parameters.

COM Object

The COM object represents the OS communication subsystem properties on CPU. Only one COM object must be defined on the local CPU. The syntax scheme of a COM object is as follows:

```
COM <name of COM> {
    COMERRORHOOK = TRUE/FALSE;
    COMUSEGETSERVICEID = TRUE/FALSE;
    COMUSEPARAMETERACCESS = TRUE/FALSE;
    COMSTARTCOMEXTENSION = TRUE/FALSE;
    COMAPPMODE = "<string>";
    COMSTATUS = COMEXTENDED/COMSTANDARD;
};
```

The object has the following standard attributes:

Table A.15 COM Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
COMERRORHOOK	TRUE, **FALSE**	Specifies if the COM ErrorHook mechanism is used
COMUSEGETSERVICEID	TRUE, **FALSE**	Defines whether the macro to get service ID is provided for COMErrorHook
COMUSEPARAMETERACCESS	TRUE, **FALSE**	Defines whether macros to get the first parameter of the service is provided for COMErrorHook
COMSTARTCOMEXTENSION	TRUE, **FALSE**	Specifies if the COM startup extention mechanism is used
COMAPPMODE	string	Specifies the COM mode name(s)
COMSTATUS	COMEXTENDED, **COMSTANDARD**	Specifies the COM error checking status

NM Object

The NM object represents the local parameters of the network management subsystem on CPU. This object has no standard parameters.

SCHEDULETABLE Object

The SCHEDULETABLE object provides a user with possibility to implement a statically defined task activation. The syntax scheme of a SCHEDULETABLE object is as follows:

```
SCHEDULETABLE = <name of SCHEDULETABLE> {
        COUNTER = <name of COUNTER;
        AUTOSTART = <TRUE / FALSE>{
            TYPE = <ABSOLUTE / RELATIVE / SYNCHRON>{
              ABSVALUE = <intenger>;
              RELOFFSET = <integer>};
            };
            APPMODE = <name of APPMODE>;
        }
        SYNC = <TRUE / FALSE>{
            SYNCSTRATEGY = <EXPLICIT / IMPLICIT>{
              EXPLICITPRECISION = <integer>;
          };
        };
        REPEATING = <TRUE / FALSE>;
        DURATION = <integer>;
        EXPIRYPOINTS = EXPIRYPOINT {
          OFFSET = <integer>;
          ACTION = <TASKACTIVATION / EVENTSETTING>{
            TASK = <name of TASK>;
            EVENT = <name of EVENT>;
          };
          ADJUSTABLEEXPPOINT = <TRUE / FALSE>{
            MAXADVANCE = <integer>;
            MAXRETARD = <integer>;
          };
        };
        ACCESSING_APPLICATION = <name of Application>;
};
```

Table A.16 SCHEDULETABLE Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
COUNTER	name of COUNTER	Specifies the assigned counter
AUTOSTART	TRUE, FALSE	Defines whether or not the SCHEDULETABLE is activated during the system start-up procedure.
SYNC	TRUE, **FALSE**	Defines whether of not the start of shedule table is synchronous
SYNCSTRATEGY	EXPLICIT, IMPLICIT	Defines the type of synchronization
EXPLICITPRECISION	integer (0..0xFFFFFFFF)	Defines the precision of synchronization in ticks
REPEATING	TRUE, FALSE	Defines whether or not the schedule table is periodic
DURATION	integer (0..0xFFFFFFFF)	Defines the length of the schedule table period in ticks
EXPIRYPOINTS	EXPIRYPOINT	Container(s) for the expire points.
ACTION	TASKACTIVATION / EVENTSETTING	Defines the action(s) inside expire point
OFFSET	integer (0..0xFFFFFFFF)	Defines the distance from the start of ScheduleTable to given ExpirePoint in ticks
TASK	name of TASK	Defines the TASK to run at the expiry point
EVENT	name of EVENT	Defines the EVENT to set at the expity point
ADJUSTABLEEXPPOINT	TRUE, **FALSE**	Defines that the adjustment of this point for synchronization is allowed
MAXADVANCE	integer (0..0xFFFFFFFF)	Defines the maximum allowed addition to offset while synchronizing; in ticks
MAXRETARD	integer (0..0xFFFFFFFF)	Defines the maximum allowed substraction to offset while synchronizing; in ticks
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this ScheduleTable