NXP AUTOSAR OS/S32K v.4.0

User's Manual

Document Number: UM2OSASR4.0R1.0.0
Rev. 1.3

Contents

Section number **Title** **Page**

Chapter 1
Introduction

Chapter 2
Sample Application

Chapter 3
Usage for Derivatives

Chapter 4
Quick Reference

Section number	Title	Page

Chapter 1 Introduction

1.1 Introduction

This User's Manual describes NXP AUTOSAR OS/S32K, 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.

Usage for 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:

- NXP AUTOSAR OS/S32K Overview

1.2 NXP AUTOSAR OS/S32K Overview

AUTOSAR Operating System is a real-time operating system which conforms to the AUTOSAR OS v.4.0.3 specification.

NXP AUTOSAR OS/S32K supports SC1 class with addition of Service Protection in Extended Status for SC1 and Global Time Synchronization for ScheduleTable.

NXP AUTOSAR OS/S32K 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

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 user is granted the features described below.

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.

Chapter 2
Sample Application

2.1 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 first look at the OS.

2.2 Source Files

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

2.3 Building Sample

Take the following steps to build the sample application:

- Open the Windows command prompt window.
- Change the current directory to `sample\standard\sc` directory which contains sample files.
- To build the sample, execute the following command (assuming that GNU make 3.81 from redist is in the path):

Please see sample readme for more details.

NOTE

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

- After build completion the following subdirectories and files are created in the `sample\standard\sc1` directory:
  - `output_<compiler>_<cfg>\obj` subdirectory contains object files and configuration files generated by SysGen.
  - `output_<compiler>_<cfg>\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 compiler=<compiler>
  ```

  Please see sample readme file for more details.

Chapter 3
Usage for Derivatives

3.1 Usage for Derivatives

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

The current version of the AUTOSAR OS supports S32K MCU directly.

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=S32K.

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

Chapter 4
Quick Reference

4.1 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

4.2 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 otherwise. GetEvent has been updated for multicore configurations to work for cross core calls. GetEvent and SetEvent remote calls return error code (E_OS_CORE) when remote core is not active.

Table 4-1. AUTOSAR OS Services

Service	Input	Output	Allowed In
OS-Application management services			
AllowAccess	-	-	Task, ISR
	syntax: StatusType AllowAccess();		
GetApplicationState	Application Id	Application state	Task, ISR, application-specific ErrorHook
	syntax: StatusType GetApplicationState(ApplicationType Application, ApplicationStateRefType Value);		
GetApplicationID	-	Application Id	Task, ISR, all hooks
	syntax: ApplicationType GetApplicationID (void);		

Table continues on the next page...

Table 4-1. AUTOSAR OS Services (continued)

Service	Input	Output	Allowed In
CheckObjectAccess	Application Id, object type and Id	Access rights	Task, ISR, ErrorHook
	syntax: ObjectAccessType CheckObjectAccess (ApplicationType ApplID, ObjectTypeType ObjectType,OSObjectType objectId);		
CheckObjectOwnership	Object type and Id	Application Id	Task, ISR, ErrorHook
	syntax: ApplicationType CheckObjectOwnership (ObjectTypeType ObjectType, OSObjectType objectId);		
TerminateApplication	Restart option	-	Task, ISR, application-specific ErrorHook
	syntax: StatusType TerminateApplication(RestartType RestartOption);		
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);		
GetTaskID	-	Task name	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskID(TaskRefType <TaskIDRef>);		
GetTaskState	Task name	Task state	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskState(TaskType <TaskID>, TaskStateRefType <StateRef>);		
Schedule Table management services			
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
	StatusType StartScheduleTableAbs(ScheduleTableType sctId, TickType offset);		
StartScheduleTableSynchron	Schedule table ID	-	Task, ISR
	syntax: StatusType StartScheduleTableSynchron(ScheduleTableType sctId);		
StopScheduleTable	Schedule table ID	-	Task, ISR
	syntax: StatusType StopScheduleTable(ScheduleTableType sctId);		
NextScheduleTable	Current schedule table ID, next schedule table ID	-	Task, ISR
	syntax: StatusType NextScheduleTable(ScheduleTableType cursctId, ScheduleTableType nextsctId)		
SyncScheduleTable	Schedule table ID, current global time	-	Task, ISR

Table continues on the next page...

Table 4-1. AUTOSAR OS Services (continued)

Service	Input	Output	Allowed In
	syntax: StatusType SyncScheduleTable(ScheduleTableType sctId, GlobalTimeTickType time)		
GetScheduleTableStatus	Schedule table ID	Schedule table status	Task, ISR
	syntax: StatusType GetScheduleTableStatus(ScheduleTableType sctId, ScheduleTableStatusRefType status)		
SetScheduleTableAsync	Schedule table ID	-	Task, ISR
	syntax: StatusType SetScheduleTableAsync(ScheduleTableType sctId)		
Interrupt management services			
GetISRID	-	-	Task, ISR, error hook, protection hook
	syntax: ISRType 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, alarmcallbacks, ErrorHook,PreTaskHook, PostTaskHook
	syntax: void ResumeAllInterrupts(void);		
SuspendAllInterrupts	-	-	Task, ISR category 1 and 2, alarmcallbacks, 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	-	Task, ISR cat2
	syntax: void DisableInterruptSource(ISRType IsrId);		
EnableInterruptSource	ISR which might be enabled	-	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>);		

Table continues on the next page...

Table 4-1. AUTOSAR OS Services (continued)

Service	Input	Output	Allowed In
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(CounterType <CounterID>, TickType <Ticks>);		
IncrementCounter	Counter name	--	Task, ISR
	syntax: StatusType IncrementCounter(CounterType <CounterID>);		
GetCounterValue	Counter Id	Counter value	Task, ISR
	syntax: StatusType GetCounterValue(CounterType <CounterID>, TickRefType <TicksRef>);		
GetElapsedValue	Counter Id, previous value	elapsed time	Task, ISR
	syntax: StatusType GetElapsedValue(CounterType <CounterID>, TickRefType <valueRef>, TickRefType <tickRef>);		
GetCounterInfo	Counter name	Counter constants	All except ISR category 1
	syntax: StatusType GetCounterInfo(CounterType <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>[1]			
	syntax: ALARMCALLBACK(<CallbackName>);		
IOC management services: <IocId> is a unique identifier that references a unidirectional 1:1 or N:1 communication; <SenderId> is used only in N:1 communication.			
IocSend_<IocId>[_<SenderId>]	message data	-	Task, ISR
	syntax: Std_StatusType IocSend_<IocId>[_<SenderId>] (<Data> OUT);		

Table continues on the next page...

Table 4-1. AUTOSAR OS Services (continued)

Service	Input	Output	Allowed In
IocWrite_<IocId>[_<SenderId>]	message data	-	Task, ISR
	syntax: Std_StatusType IocWrite_<IocId>[_<SenderId>] (<Data> IN);		
IocReceive_<IocId>	-	Message data	Task, ISR cat. 2
	syntax: Std_StatusType IocReceive_<IocId> (<Data> OUT);		
IocRead_<IocId>	-	Message data	Task, ISR cat. 2
	syntax: Std_StatusType IocRead_<IocId> (<Data> OUT);		
syntax: Std_StatusType IocEmptyQueue_<IocId>(void);			Task, ISR cat. 2
<CallBackName>[2]	-	-	
	syntax: void <CallbackName> (void);		
Debugging services			
GetRunningStackUsage	-	-	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook; In SC3 and SC4 only from Task
	syntax: unsigned short GetRunningStackUsage(void);		
GetStackUsage	Task name	-	Only for SC1 and SC2; Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetStackUsage(TaskType <TaskID>);		
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>);		
PostIsrHook	-	-	-
	syntax: void PostIsrHook(void);		

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

NOTE

InitCounter, GetCounterInfo, GetRunningStackUsage, GetStackUsage, PreIsrHook, PostIsrHook services are not defined in the AUTOSAR OS v.4.0.3 specification. It is NXP OS extension of the AUTOSAR OS.

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

Table 4-2. 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_GetAlarmBase_AlarmID()	Alarm identifier
OSError_GetAlarm_AlarmID()	Alarm identifier
OSError_SetRelAlarm_AlarmID()	Alarm identifier
OSError_SetAbsAlarm_AlarmID()	Alarm identifier
OSError_CancelAlarm_AlarmID()	Alarm identifier
OSError_InitCounter_CounterID()[1]	Counter identifier
OSError_IncrementCounter_CounterID()	Counter identifier
OSError_GetCounterValue_CounterID()	Counter identifier
OSError_GetElapsedValue_CounterID()	Counter identifier
OSError_GetCounterInfo_CounterID()	Counter identifier
OSError_StartScheduleTableAbs_ScheduleTableID()	Schedule Table identifier
OSError_StopScheduleTable_ScheduleTableID()	Schedule Table identifier
OSError_NextScheduleTable_ScheduleTableID()	Schedule Table identifier
OSError_StartScheduleTableSynchron_ScheduleTableID()	Schedule Table identifier
OSError_SyncScheduleTable_ScheduleTableID()	Schedule Table identifier
OSError_SetScheduleTableAsync_ScheduleTableID()	Schedule Table identifier
OSError_GetScheduleTableStatus_ScheduleTableID()	Schedule Table identifier
OSError_TerminateApplication_ApplicationID()	Application identifier
OSError_CheckObjectOwnership_ObjectID()	OS object identifier

1. Counter interface functions are not defined in AUTOSAR OS v.4.0.3 specification, this is NXP OS extension of the AUTOSAR OS.

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

Table 4-3. Data Types

Data Type	Description
ApplicationType	The data type for OS-Application identification
ObjectTypeType	The data type for object identification
RestartType	The argument type for TerminateApplication() service
ScheduleTableType	The data type identifies a ScheduleTable.
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*
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);
CounterType	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
ResourceType	The abstract data type for referencing a resource
StatusType	The data type for all status information the API services offer
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.4.0.3 specification. This is NXP OS extension of the AUTOSAR OS.

The DeclareISR(<ISR name>) constructional element is defined in NXP AUTOSAR OS.

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

Table 4-4. 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_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_SYS_FATAL	21	Fatal error in the OS code
E_OS_SYS_ORDER	23	Incorrect order of function calling
E_OS_PARAM_POINTER	29	A pointer argument to an API is null
IOC_E_OK	0	No error, successful completion
IOC_E_NOK	1	Error occurred. Used to identify error cases without error specification.
IOC_E_LIMIT	130	Overflow of FIFO associated with queued messages
IOC_E_LOST_DATA	64	The IOC service refused an IocSend request due to internal buffer overflow
IOC_E_NO_DATA	131	No data is available for reception in case of queued messages

The list of service identifiers for ErrorHook is provided below:

- identifiers for standard AUTOSAR services
  - OSServiceId_StartOS
  - OSServiceId_ShutdownOS
  - OSServiceId_GetActiveApplicationMode
  - OSServiceId_GetApplicationID
  - OSServiceId_AllowAccess
  - OSServiceId_GetApplicationState
  - OSServiceId_CheckObjectAccess
  - OSServiceId_CheckObjectOwnership
  - OSServiceId_TerminateApplication
  - 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_SetEvent
  - OSServiceId_ClearEvent
  - OSServiceId_GetEvent
  - OSServiceId_WaitEvent
  - OSServiceId_GetAlarmBase
  - OSServiceId_GetAlarm
  - OSServiceId_SetRelAlarm
  - OSServiceId_SetAbsAlarm
  - OSServiceId_CancelAlarm
  - OSServiceId_GetCounterValue
  - OSServiceId_GetCounterInfo
- identifiers for NXP AUTOSAR OS specific services
  - OSServiceId_InitCounter
- identifier returned if the error occurred 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 4-5. 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 4-6. 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 counterspecific value in the system counter (it is the user constant not used by OS)
OSTICKSPERBASE2		Number of ticks required to reach a counterspecific 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)

Table continues on the next page...

Table 4-6. AUTOSAR OS Constants (continued)

Constant	Value	Description
OSDEFAULTAPPMODE		Default application mode. This constant is always a valid parameter for *StartOS* service
INVALID_OSAPPLICATION		Constant of data type ApplicationType for not defined OS-Application
OSTPTICKDURATION		Duration of the TP counter tick in nanoseconds
<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 2.1.56

NOTE

OSMAXALLOWEDVALUE2, OSTICKSPERBASE2, OSTICKDURATION2, INVALID_OSAPPLICATION, OSTPTICKDURATION, OSMINCYCLE2, <IsrName>PRIORITY and OSBUILDNUMBER constants are not defined in the AUTOSAR OS v.4.0.3 specification. This is NXP OS extension of the AUTOSAR OS.

4.3 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. NXP 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.

4.3.1 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.

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

Table 4-7. 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>;`
`CC = <BCC1 / ECC1 / `**`AUTO`**`>;`
`DEBUG_LEVEL = <0 / 1 / 2 / 3 / 4>;`
`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>;`
`ISRFLOATINGPOINT = <TRUE / `**`FALSE`**`>;`		

CC $_{\text{specific}}$	BCC1, ECC1, **AUTO**	This specific attribute specifies the conformance class which is supported by the OS
SCALABILITYCLASS $_{\text{standard}}$	SC1	This standard attribute defines the scalability class of OS.
ISRFLOATINGPOINT $_{\text{specific}}$	TRUE, **FALSE**	This specific attribute specifies whether the Floating-point unit in the interrupt handler (IRQ only) should be supported or not
DEBUG_LEVEL $_{\text{specific}}$	**0,** 1, 2, 3, 4	This specific attribute specifies the ORTI support in OS
STATUS $_{\text{standard}}$	STANDARD, EXTENDED	This standard attribute specifies OS debug status
STARTUPHOOK $_{\text{standard}}$	TRUE, FALSE	This standard attribute defines whether StartupHook is called after the operating system starting up and before the dispatcher starting or not

Table continues on the next page...

Table 4-7. OS Parameters (continued)

Object Parameters	Possible Values	Description
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
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
PROTECTIONHOOK standard	TRUE, **FALSE**	This standard attribute defines whether or not PROTECTIONHOOK is called on protection error.
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

4.3.2.1 Specific OS Parameters

Table 4-8. OS Parameters

Object Parameters	Possible Values	Description
Global Multi-core Related Attributes		These attributes define single or multi core support in the system.
The attributes should be define inside the scope of the OS object in accordance with the following syntax: `NumberOfCores= <1>;`		
NumberOfCores specific	1	Defines maximum number of cores that are controlled by the OS
Stack Related Attributes		These attributes define stack support in the system.

Table continues on the next page...

Table 4-8. OS Parameters (continued)

Object Parameters	Possible Values	Description

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
 	
`STACKMONITORING  = <TRUE / FALSE>{`
`    Pattern = <0x55555555 / integer>;`
`    PatternSize = <1 / 2 / 4>;`
`};`
`CommonStackSize = <integer / AUTO>;`
`CommonStackSize2 = <integer / AUTO>;`
`IsrStackSize =  <integer / AUTO>;`
`IsrStackSize2 =  <integer / AUTO>;`		

STACKMONITORING $_{\text{standard}}$	TRUE, FALSE	Defines whether or not OS monitors stack
Pattern $_{\text{specific}}$	**0x55555555**, `integer`	Specifies a 32-bit pattern to fill stack
PatternSize $_{\text{specific}}$	**1**, 2, 4	Defines the size in 32-bit words of area to be checked
CommonStackSize $_{\text{specific}}$	`integer`	Defines the common stack size for basic tasks in bytes (in SC3 and SC4)
Interrupt Related Properties		This group of OS attributes defines parameters of ISR execution
IsrStackSize $_{\text{specific}}$	`integer`	The attribute specifies ISR stack size. It shall be defined if there are ISR category 2 in SC2 and in SC1 if CC = ECC1

CPU Related Attributes	This group of OS attributes provides possibility to tune the selected hardware
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>;
        Size = <integer>;
    };
    InternalRAM = <TRUE / FALSE> {
        Address = <integer / 0x1C000000>;
        Size = <integer / 0x00040000>;
    };
    InternalRAM2 = <TRUE / FALSE> {
        Address = <integer>;
        Size = <integer>;
    };
    ExternalROM = <TRUE / FALSE> {
        Address = <integer>;
        Size = <integer>;
    };
    ExternalRAM = <TRUE / FALSE> {
        Address = <integer>;
        Size = <integer>;
    };
  ClockFrequency = <integer / 12000>;
    ClockDivider = <integer / 1>;
    ClockMultiplier = <integer / 1>;

    SysTimer = <HWCOUNTER / SWCOUNTER / NONE> {
        COUNTER = <name of COUNTER>;
        ISRPRIORITY = <integer>;
        Period = <integer / AUTO>;
        TimerHardware = <name of timer hardware> {
            GlobalFTMPrescaler = <USER / OS> {
                Value = <integer / AUTO>;
            };
            Channel = <integer>;
            PeripheralClockDivider = <integer / 1>;
            TimerModuloValue = <integer / AUTO>;
            Freeze = <TRUE / FALSE>;
        };
    };

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

Object Parameters	Possible Values	Description
TargetMCU $_{\text{specific}}$	S32K	The attribute specifies target MCU type
ClockFrequency $_{\text{specific}}$	integer, **12000**	The attribute specifies oscillator frequency in kHz (used for calculating prescaler value and timer modulo value)
ClockDivider $_{\text{specific}}$	integer, **1**	The attribute specifies the divider value that is set by the User in PLL.
ClockMultiplier $_{\text{specific}}$	integer, **1**	The attribute specifies the multiplier value that is set by the User in PLL.
InternalROM $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies is internal ROM used by the OS or user application.
InternalRAM $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies is external RAM used by the OS or user application.
InternalRAM2 $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies is external RAM used by the OS or user application.
ExternalROM $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies is internal ROM used by the OS or user application.
ExternalRAM $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies is external RAM used by the OS or user application.
Address $_{\text{specific}}$	integer	The attribute specifies the address of memory area.
Size $_{\text{specific}}$	integer	The attribute specifies the size of memory area.
SysTimer $_{\text{specific}}$	HWCOUNTER, SWCOUNTER, NONE	The attribute defines whether the internal OS system timer is used or not.
SecondTimer $_{\text{specific}}$	HWCOUNTER, SWCOUNTER, NONE	The attribute defines whether the internal OS second timer is used or not.

Table continues on the next page...

Object Parameters	Possible Values	Description
COUNTER $_{\text{specific}}$	name of COUNTER	The attribute 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
ISRPRIORITY $_{\text{specific}}$	[1-14]	Specifies priority of system timer (second timer) interrupt handler
Period $_{\text{specific}}$	integer AUTO	The attribute specifies period of a tick of the system (second/TP timer) counter in nanoseconds.
TimerHardware $_{\text{specific}}$	FTM(0/1/2/3/4/5/6/7), SYSTICK	The attribute is intended to select the hardware interrupt source for the System and Second Timers.
GlobalFTMPrescaler $_{\text{specific}}$	USER, OS	The attribute specifies whether the FTM timer HW shall be initialized during OS startup. The value of the FTM prescaler that divides the FTM channel clock frequency is actually computed as being the power of 2 of the value given in the oil configuration file.
PeripheralClockDivider $_{\text{specific}}$	1 .. 16	The attribute specifies MCU peripheral clock divider value
Channel $_{\text{specific}}$	integer	The attribute specifies the timer HW channel to be used for OS timer
TimerModuloValue $_{\text{specific}}$	integer, **AUTO**	The attribute specifies timer hardware register value
Freeze $_{\text{specific}}$	TRUE, **FALSE**	The attribute specifies timer freeze bit value. If this attribute has value TRUE, the timer will be frozen while the MCU is in the debug mode

4.3.3 APPLICATION Object

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

```
APPLICATION <name of APPLICATION>{
    TASK = <name of TASK>;
    ISR = <name of ISR>;
    ALARM = <name of ALARM>;
    SCHEDULETABLE = <name of SCHEDULETABLE>;
    COUNTER = <name of COUNTER>;
    STARTUPHOOK = <TRUE / FALSE>;
    SHUTDOWNHOOK = <TRUE / FALSE>;
    ERRORHOOK = <TRUE / FALSE>;

    HAS_RESTARTTASK = <TRUE / FALSE>{
        RESTARTTASK = <name of TASK>
    };
    TRUSTED = <FALSE / TRUE>{
        TRUSTED_FUNCTION = <TRUE / FALSE>{
            NAME = <name of FUNCTION>;
        };
    };

};
```

The brief description of the OS_Application attributes is presented below.

Table 4-9. 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 which is called at the OS-Application restart.
RESTARTTASK standard	name of TASK	Defines the task to be started on 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.
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.
CORE standard		Defines ID of the core onto which the OS-Application is bound.
Address specific	integer	Specifies the address of peripheral memory area.
Size specific	integer	Specifies the size of peripheral memory area in bytes.

4.3.4 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> / AUTO;
    RESOURCE = <name of RESOURCE>;
    EVENT = <name of EVENT>;

    FLOATINGPOINT = <TRUE / FALSE>;
    ACCESSING_APPLICATION = <name of Application>;
};
```

The brief description of the task attributes is presented below.

Table 4-10. 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 NXP 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
FLOATINGPOINT	TRUE, FALSE	Specifies whether the Floating-point unit in task should be supported or not.
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this TASK
NXP AUTOSAR OS Specific Attribute		
STACKSIZE	integer	Defines the size of the Task's stack in bytes

4.3.5 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>;
    IsrFunction = <string / AUTO>;
    STACKSIZE = <integer> / AUTO;
    RESOURCE = <name of RESOURCE>;
};
```

The following parameters can be defined for the ISR object:

Table 4-11. 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
PRIORITY	[1..]	Specifies the priority of the interrupt service routine.
NXP AUTOSAR OS Specific Attributes		
STACKSIZE	integer	Defines the size of the ISR's stack in bytes. It is not required within SC1.
IsrFunction	string	Specifies the name of ISR handler function. By default it is the name of ISR object itself.

4.3.6 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 4-12. 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

4.3.7 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 4-13. EVENT Parameters

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

4.3.8 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>;
    SECONDSPERTICK = <integer> / AUTO;
    TICKSPERBASE = <integer>;
    TYPE = <SOFTWARE / HARDWARE>{
        DRIVER = OSINTERNAL/GPT {
            NS_PER_HW_TICK = <integer>;
            GPTCHANNELNAME = "name";
        };
        TIMECONSTANTS = TIMECONSTANT{
            NS = <integer>;
            CONSTNAME = "name";
        };
    };
    ACCESSING_APPLICATION = <name of Application>;
};
```

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

Table 4-14. COUNTER Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
MINCYCLE	integer [1.. 0xFFFFFFFF]	Specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter
SECONDSPERTICK	integer, **AUTO**	Specifies the length of tick of the counter
MAXALLOWEDVALUE	integer [1.. 0xFFFFFFFF]	Defines the maximum allowed counter value
TICKSPERBASE	integer [1.. 0xFFFFFFFF]	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, GPT	Defines timer driver; shall be OSINTERNAL, GPT is not supported
TIMECONSTANTS	TIMECONSTANT	Define constants which can be used to compare time values with timer tick values
NS	integer	Defines constant value
CONSTNAME	name of constant	Defines the name of constant with value in ticks
NS_PER_HW_TICK	integer	Defines the value in ticks; GPT is not supported
GPTCHANNELNAME	name of constant	Defines the name of GPT Channel; GPT is not supported
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this COUNTER

4.3.9 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>;
        TYPE = <ABSOLUTE / RELATIVE>;
    };
    ACCESSING_APPLICATION = <name of Application>;
};
```

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

Table 4-15. 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
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
TYPE	ABSOLUTE,RELATIVE	Defines the type of autostart for the alarm
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this ALARM

4.3.10 IOC Object

Parameters of this object type define the IOC properties. The syntax of the IOC object definition is presented below. Note that either ACTION (not equal NONE) or RECEIVER_PULL_CB attribute should be defined for the IOC object..

```
IOC <name of IOC> {
    DATA_PROPERTIES = DATA_PROPERTY{
        DATA_PROPERTY_INDEX = <integer / AUTO>
        DataTypeName = <string>;
        INIT_VALUE = <string> / AUTO;
        DataTypeProperty = <REFERENCE / DATA>;
    };
    BUFFER_LENGTH = <integer> / 0;
    RECEIVER = RCV {
        FUNCTION_IMPLEMENTATION_KIND=<DO_NOT_CARE / FUNCTION /MACRO>;
        RCV_OSAPPLICATION = <name of APPLICATION>;
        RECEIVER_PULL_CB = <string> / AUTO;
        ACTION = <NONE / ACTIVATETASK / SETEVENT> {
            TASK = <name of TASK>;
            EVENT = <name of EVENT>;
        };
    };
    SENDER = SND {
        FUNCTION_IMPLEMENTATION_KIND=<DO_NOT_CARE / FUNCTION /MACRO>;
        SENDER_ID = <integer> / AUTO;
        SND_OSAPPLICATION = <name of APPLICATION>;
    };
};
```

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

Table 4-16. IOC Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
DATA_PROPERTIES	DATA_PROPERTY	Specifies container(s) for data properties
DataTypeName	string	Specifies the type of the data to be transferred on the IOC communication channel.
DataTypeProperty	DATA, REFERENCE	Specifies how the data is passed to sending functions (IOCSend, IOCWrite) - by reference or by value
DATA_PROPERTY_INDEX	integer [0..0xFF], **AUTO**	Specifies the order the data is send, e.g. whether IocSendGroup(A,B) or IocSendGroup(B,A) shall be used.
INIT_VALUE	string, **AUTO**	Specifies Initial Value for the data to be transferred on the IOC communication channel
BUFFER_LENGTH	integer, **0**	Specifies the size of the IOC internal queue to be allocated for a queued communication. If it is set to 0 or undefined, the IOC is not QUEUED
RECEIVER	RCV	Specifies the receiver of the message
FUNCTION_IMPLEMENTATI ON_KIND	DO_NOT_CARE, FUNCTION, MACRO	Specifies whether this communication is implemented as a macro or as a function. IOC is implemented as "MACRO". The rest implementation kind values are ignored.
RCV_OSAPPLICATION	name of APPLICATION	Specifies the receiving OS-Application.
RECEIVER_PULL_CB	string, **AUTO**	Defines the name of a callback function
ACTION	ACTIVATETASK, SETEVENT, 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

Table continues on the next page...

Table 4-16. IOC Parameters (continued)

Object Parameters	Possible Values	Description
EVENT	name of EVENT	Specifies the event to be set when the message has arrived. It shall be defined if ACTION is SETEVENT only
SENDER	SND	Specifies the sender of the message
SND_OSAPPLICATION	name of APPLICATION	Specifies the sending OS-Application.
SENDER_ID	[0..255], **AUTO**	Defines a sender in a N:1 communication to distinguish between senders.

4.3.11 APPMODE Object

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

4.3.12 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;
    REPEATING = <TRUE / FALSE>;
    DURATION = <integer>;
    ACCESSING_APPLICATION = <name of Application>;
    AUTOSTART = <TRUE / FALSE>{
        APPMODE = <name of APPMODE>;
        TYPE = <ABSOLUTE / RELATIVE / SYNCHRON>;
        STARTVALUE = <integer / AUTO>;
    };
    SYNC = <TRUE / FALSE>{
        SYNCSTRATEGY = <EXPLICIT / IMPLICIT>{
            EXPLICITPRECISION = <integer>;
        };
    };
    EXPIRYPOINTS = EXPIRYPOINT {
        OFFSET = <integer>;
        ACTION = <TASKACTIVATION / EVENTSETTING>{
            TASK = <name of TASK>;
            EVENT = <name of EVENT>;
        };
        ADJUSTABLEEXPPOINT = <TRUE / FALSE>{
            MAXADVANCE = <integer>;
            MAXRETARD = <integer>;
        };
    };
};
```

The scheduletable object has the following standard attributes:

Table 4-17. 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
TYPE	ABSOLUTE / RELATIVE / SYNCHRON	Defines whether the Schedule Table is automatically started at system start-up synchronously or with given offset or absolute value of the assigned counter
STARTVALUE	integer (0..0xFFFFFFFF) / **AUTO**	Defines the value of assigned counter at which the Schedule Table will start.
APPMODE	name of APPMODE	Defines an application mode in which the schedule table will be started automatically
ABSVALUE	integer (0..0xFFFFFFFF)	Defines the value of assigned counter at which the Schedule Table will start.
RELOFFSET	integer (0..0xFFFFFFFF)	Defines the offset from OS startup when Schedule Table will start in counter ticks.
SYNC	TRUE, **FALSE**	Defines whether of not the start of schedule 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.
OFFSET	integer (0..0xFFFFFFFF)	Defines the distance from the start of ScheduleTable to given ExpirePoint in ticks
ACTION	TASKACTIVATION / EVENTSETTING	Defines the action(s) inside expire point
TASK	name of TASK	Defines the TASK to run at the expiry point
EVENT	name of EVENT	Defines the EVENT to set at the expiry 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 subtraction to offset while synchronizing; in ticks
ACCESSING_APPLICATION	name of Application	Defines which application(s) may access this SCHEDULETABLE

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support