
RTA-OSEK

Reference Guide

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.: +49 (711) 8 96 61-102
Fax: +49 (711) 8 96 61-106

www.etas.de

Japan

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

www.etas.co.jp

Korea

ETAS Korea Co., Ltd.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-899, Korea

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

www.etas.co.kr

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

www.etasinc.com

France

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

www.etas.fr

Great Britain

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

www.etas-uk.net

People's Republic of China

2404 Bank of China Tower
200 Yincheng Road Central
Shanghai 200120

Tel.: +86 21 5037 2220

Fax: +86 21 5037 2221

www.etas.cn

LiveDevices

LiveDevices Ltd.
Atlas House
Link Business Park
Osbalwick Link Road
Osbalwick
York, YO10 3JB

Tel.: +44 (0) 19 04 56 25 80

Fax: +44 (0) 19 04 56 25 81

www.livedevices.com



Copyright Notice

© 2001 - 2007 LiveDevices Ltd. All rights reserved.

Version: RTA-OSEK v5.0.2

No part of this document may be reproduced without the prior written consent of LiveDevices Ltd. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on any part of LiveDevices. While the information contained herein is assumed to be accurate, LiveDevices assumes no responsibility for any errors or omissions.

In no event shall LiveDevices, its employees, its contractors or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees or expenses of any nature or kind.

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

AUTOSAR is a trademark of AUTOSAR GdR.

All other product names are trademarks or registered trademarks of their respective owners.

Certain aspects of the technology described in this guide are the subject of the following patent applications:

UK - 0209479.5 and USA - 10/146,654,

UK - 0209800.2 and USA - 10/146,239,

UK - 0219936.2 and USA - 10/242,482.

Contents

- 1 About this Guide 1-1
 - 1.1 Who Should Read this Guide? 1-1
 - 1.2 Conventions 1-1
 - 1.3 References 1-1
- 2 RTA-OSEK Type Definitions 2-1
 - 2.1 Type Definition Notes 2-3
- 3 RTA-OSEK API Reference 3-1
 - 3.1 OSEK Conformance 3-1
 - 3.2 AUTOSAR and RTA-OSEK Features 3-1
 - 3.3 Static and Dynamic Interface 3-1
 - 3.4 The API Call Template 3-2
 - 3.5 ActivateTask() 3-4
 - 3.6 ActivateTaskset() 3-6
 - 3.7 AdvanceSchedule() 3-8

3.8	AssignTaskset()	3-10
3.9	CancelAlarm()	3-12
3.10	ChainTask()	3-14
3.11	ChainTaskset()	3-16
3.12	ClearEvent()	3-18
3.13	CloseCOM()	3-20
3.14	DisableAllInterrupts()	3-22
3.15	EnableAllInterrupts()	3-24
3.16	GetActiveApplicationMode()	3-25
3.17	GetAlarm()	3-26
3.18	GetAlarmBase()	3-28
3.19	GetArrivalpointDelay()	3-30
3.20	GetArrivalpointNext()	3-32
3.21	GetArrivalpointTasksetRef()	3-34
3.22	GetCounterValue()	3-36
3.23	GetEvent()	3-38
3.24	GetISRID()	3-40
3.25	GetMessageResource()	3-41
3.26	GetMessageStatus()	3-43
3.27	GetResource()	3-45
3.28	GetScheduleNext()	3-47
3.29	GetScheduleStatus()	3-49
3.30	GetScheduleTableStatus()	3-51
3.31	GetScheduleValue()	3-53
3.32	GetStackOffset()	3-55
3.33	GetTaskID()	3-56
3.34	GetTasksetRef()	3-58
3.35	GetTaskState()	3-60
3.36	IncrementCounter()	3-62
3.37	InitCOM()	3-64
3.38	InitCounter()	3-66
3.39	MergeTaskset()	3-68

3.40	NextScheduleTable()	3-70
3.41	osAdvanceCounter_<CounterID>()	3-72
3.42	osResetOS()	3-74
3.43	ReadFlag()	3-75
3.44	ReceiveMessage()	3-77
3.45	ReleaseMessageResource()	3-79
3.46	ReleaseResource()	3-81
3.47	RemoveTaskset()	3-83
3.48	ResetFlag()	3-85
3.49	ResumeAllInterrupts()	3-87
3.50	ResumeOSInterrupts()	3-89
3.51	Schedule()	3-91
3.52	SendMessage()	3-93
3.53	SetAbsAlarm()	3-95
3.54	SetArrivalpointDelay()	3-97
3.55	SetArrivalpointNext()	3-99
3.56	SetEvent()	3-101
3.57	SetRelAlarm()	3-103
3.58	SetScheduleNext()	3-106
3.59	ShutdownOS()	3-108
3.60	StartCOM()	3-109
3.61	StartOS()	3-111
3.62	StartSchedule()	3-112
3.63	StartScheduleTable()	3-114
3.64	StopCOM()	3-116
3.65	StopSchedule()	3-118
3.66	StopScheduleTable()	3-120
3.67	SuspendAllInterrupts()	3-122
3.68	SuspendOSInterrupts()	3-124
3.69	TerminateTask()	3-126

3.70	TestArrivalpointWritable()	3-128
3.71	TestEquivalentTaskset()	3-130
3.72	TestSubTaskset()	3-132
3.73	Tick_<CounterID>()	3-134
3.74	TickSchedule()	3-136
3.75	WaitEvent()	3-138
4	Constructional Elements	4-1
4.1	DeclareAccessor()	4-1
4.2	DeclareAlarm()	4-2
4.3	DeclareCounter()	4-3
4.4	DeclareEvent()	4-4
4.5	DeclareFlag()	4-5
4.6	DeclareISR()	4-6
4.7	DeclareMessage()	4-7
4.8	DeclareResource()	4-8
4.9	DeclareTask()	4-9
5	Advanced Counter & Schedule Driver Interface	5-1
5.1	Tick Source Semantics	5-1
5.2	Initialization	5-2
5.3	Cancel_<ScheduleID>() [User Provided]	5-3
5.4	Now_<ScheduleID>() [User Provided]	5-4
5.5	Set_<ScheduleID>() [User Provided]	5-5
5.6	State_<ScheduleID>() [User Provided]	5-6
5.7	Cancel_<CounterID> [User Provided]	5-7
5.8	Now_<CounterID>() [User Provided]	5-8
5.9	Set_<CounterID>() [User Provided]	5-9
5.10	State_<CounterID>() [User Provided]	5-10
6	Execution Time Monitoring	6-1
6.1	GetExecutionTime()	6-1
6.2	GetLargestExecutionTime()	6-3
6.3	GetStopwatch() [User Provided]	6-5

6.4	GetStopwatchUncertainty() [User Provided].....	6-6
6.5	ResetLargestExecutionTime()	6-7
7	Hook Routines	7-1
7.1	ErrorHook() [User Provided]	7-1
7.2	OverrunHook() [User Provided]	7-3
7.3	MessageInit() [User Provided].....	7-4
7.4	PostTaskHook() [User Provided].....	7-5
7.5	PreTaskHook() [User Provided]	7-6
7.6	ShutdownHook() [User Provided]	7-7
7.7	StackFaultHook() [User Provided]	7-8
7.8	StartupHook() [User Provided]	7-10
8	Callbacks.....	8-1
8.1	ALARMCALLBACK() [User Provided]	8-1
8.2	COMCALLBACK() [User Provided].....	8-2
9	Predefined Objects.....	9-1
9.1	OSEK Counter Attributes.....	9-1
9.2	OSEK Task States	9-1
9.3	OSEK Resources.....	9-2
9.4	OSEK Application Modes.....	9-2
9.5	RTA-OSEK Build Levels	9-2
9.6	RTA-OSEK Tasksets	9-2
9.7	RTA-OSEK Application Characteristics.....	9-3
10	Macro Definitions	10-1
11	Quick Reference Guide.....	11-1
11.1	Dynamic Interface RTA-OSEK API Calls.....	11-1
11.2	Static Interface RTA-OSEK API Calls	11-4
11.3	Constructional Elements.....	11-5
11.4	Advanced Schedule Driver Interface.....	11-5
11.5	Advanced Counter Driver Interface.....	11-6

11.6	Execution Time Monitoring Interface	11-7
11.7	Hooks.....	11-7
11.8	Other Callbacks	11-7
11.9	Predefined Objects.....	11-8
11.10	Macro Definitions	11-9
11.11	Error Codes	11-11
12	Application Build Reference.....	12-1
12.1	Command Line Options	12-1
12.1.1	General Options.....	12-1
12.1.2	Build Options	12-2
12.1.3	Analysis Options.....	12-3
12.2	Generated Files.....	12-4
12.2.1	Header Files	12-4
12.2.2	Header File Include Structure	12-5
12.3	RTA-OSEK Libraries	12-5
12.4	RTA-OSEK Builder	12-6
12.4.1	Environment Variables.....	12-6
12.4.2	Macros	12-8
13	Target .ini files	13-1
13.1	What is a target .ini file?	13-1
13.2	Naming the target .ini file	13-1
13.3	The format of a target .ini file	13-2
13.4	The [globals] section	13-2
13.4.1	CPU data	13-2
13.4.2	Custom build data	13-3
13.5	The [vectors] section	13-4
13.6	The ORTI debugger sections	13-5
13.6.1	Describing an ORTI debugger	13-5
13.6.2	Overriding an existing debugger description.....	13-9
13.6.3	Adding a new debugger description	13-9
13.6.4	Choosing new attribute values	13-10



1 About this Guide

This guide contains the complete technical details of RTA-OSEK Component. It describes the parts of RTA-OSEK that are common to all target hardware.

For each supported target, there is also an *RTA-OSEK Binding Manual* that provides target-specific information.

1.1 Who Should Read this Guide?


It is assumed that you are a developer who wants to know how to create predictable real-time systems using RTA-OSEK Component. Familiarity with the *RTA-OSEK User Guide* and the OSEK and OIL specifications is assumed.

1.2 Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running RTA-OSEK Component.

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the *courier* typeface. When the name of an object is made available to the programmer the name also appears in the *courier* typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

In the RTA-OSEK API Reference chapter, the  symbol is used in the Calling Environment tables. The symbol is used to indicate that support is provided in RTA-OSEK Component, but it may not be portable to other OSEK implementations.

1.3 References

OSEK is a European automotive industry standards effort to produce open systems interfaces for vehicle electronics. For details of the OSEK standards refer to <http://www.osek-vdx.org>.

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. For details of the AUTOSAR standards, please refer to <http://www.autosar.org>.

2 RTA-OSEK Type Definitions

RTA-OSEK provides support for 3 classes of operating system features:

1. OSEK OS
2. AUTOSAR (SC1) OS
3. Unique RTA-OSEK features.

The following table describes the C types that are provided by the classes. Portability of types is indicated in the "Portability" column. Types marked as OSEK in this column are also portable to AUTOSAR (SC1) OS.

Symbol	Description	Portability
AccessNameRef	Address of message data field.	OSEK
AlarmBaseRefType	Pointer to AlarmBaseType.	OSEK
AlarmBaseType	Structure for storage of counter characteristics (see <i>Section 2.1</i>).	OSEK
AlarmType	Alarm object.	OSEK
AppModeType	Application mode.	OSEK
ArrivalpointConstType	Read-only arrivalpoint object.	RTA-OSEK
ArrivalpointRefType	Pointer to ArrivalpointType.	RTA-OSEK
ArrivalpointType	Arrivalpoint object.	RTA-OSEK
BooleanRefType	Pointer to BooleanType.	RTA-OSEK
BooleanType	0 represents FALSE, non-zero represents TRUE.	RTA-OSEK
ByteType	Unsigned byte.	RTA-OSEK
CounterType	Counter object.	AUTOSAR
CycleType	Execution time measured in processor cycles. Unsigned integer and at least 16 bits.	RTA-OSEK
EventMaskRefType	Pointer to EventMaskType.	OSEK
EventMaskType	Mask of zero or more events.	OSEK

Symbol	Description	Portability
FlagType	Pointer to FlagValue.	OSEK
FlagValue	Flag object has the members TRUE & FALSE.	OSEK
ISRType	ISR Object	AUTOSAR
OSServiceIdType	Represents identification of system services.	OSEK
ResourceType	Resource object.	OSEK
ScheduleStatusRefType	Pointer to ScheduleStatusType.	RTA-OSEK
ScheduleStatusType	Structure for storage of RTA-OSEK schedule status information and advanced OSEK counter status information (see <i>Section 2.1</i>).	RTA-OSEK
ScheduleType	RTA-OSEK Schedule object	RTA-OSEK
ScheduleTableType	AUTOSAR Schedule Table object	AUTOSAR
ScheduleTableStatusType	Status for an AUTOSAR Schedule Table object	AUTOSAR
ScheduleTableStatusRefType	Pointer to ScheduleTableStatusType	AUTOSAR
SmallType	Memory efficient type. Unsigned integer and at least 8 bits.	RTA-OSEK
StackOffsetRefType	Pointer to StackOffsetType.	RTA-OSEK
StackOffsetType	Data type containing all stack offset values (often just one).	RTA-OSEK
StatusType	Status information API calls offer.	OSEK
StopwatchTickRefType	Pointer to StopwatchTickType.	RTA-OSEK

Symbol	Description	Portability
StopwatchTickType	Execution time measured in 'stopwatch' counter ticks. Unsigned integer and at least 16 bits.	RTA-OSEK
SymbolicName	Message object identifier.	OSEK
TaskRefType	Pointer to TaskType.	OSEK
TasksetConstType	RTA-OSEK read-only taskset.	RTA-OSEK
TasksetRefType	Pointer to TasksetType.	RTA-OSEK
TasksetType	RTA-OSEK taskset.	RTA-OSEK
TaskStateRefType	Pointer to TaskStateType.	OSEK
TaskStateType	State of a task. Includes members READY, RUNNING, SUSPENDED and WAITING.	OSEK
TaskType	Task object.	OSEK
TickRefType	Pointer to TickType.	OSEK
TickType	Count values in ticks. Unsigned integer. At least 16 bits.	OSEK
UInt16Type	Unsigned integer. Exactly 16 bits.	RTA-OSEK
UInt32Type	Unsigned integer. Exactly 32 bits (not provided on all RTA-OSEK targets).	RTA-OSEK
UIntType	Unsigned integer. At least 16 bits.	RTA-OSEK

2.1 Type Definition Notes

AlarmBaseType is used with OSEK alarms. It is a structure with the form:

```
struct {
```

```

TickType maxallowedvalue;
TickType ticksperbase;
TickType mincycle;
}

```

Where `maxallowedvalue` is the maximum allowed count value in ticks, `ticksperbase` is the number of ticks required to reach a counter-specific (significant) unit (not used by RTA-OSEK) and `mincycle` is the smallest allowed value for the cycle parameter of `SetRelAlarm()` or `SetAbsAlarm()`.

`ScheduleStatusType` is used with RTA-OSEK schedules and Advanced OSEK Counters. It is a C structure with the form:

```

struct {
    SmallType status;
    TickType expiry;
}

```

Where `status` encodes two bit fields, as shown in Figure 2:1.

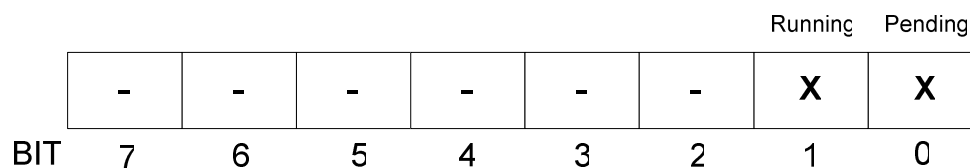


Figure 2:1 - `ScheduleStatusType()` Bit Fields

The value of the bit fields determines whether the value of `expiry` is defined. This is shown in the following table:

OS_STATUS_RUNNING	OS_STATUS_PENDING	status	expiry defined	Description
Not Set	Not Set	0	No	Stopped and not pending.
Not Set	Set	1	No	<i>Pending, but stopped (not possible).</i>
Set	Not Set	2	Yes	Running, but not pending.
Set	Set	3	No	Running and pending.

When `expiry` is defined it takes a value equal to the number of ticks before:

- the RTA-OSEK Schedule is due to process the next arrivalpoint for a schedule table
- the next advanced OSEK Alarm or AUTOSAR ScheduleTable Expiry Point due to expire on the associated counter

Important: The `expiry` is the number of ticks relative to the last expiry at which the next expiry is due. A value of 0 means that the number of ticks before the schedule is due is equal to the modulus of the RTA-OSEK Schedule or `maxallowedvalue` of the advanced OSEK Counter.

3 RTA-OSEK API Reference

This chapter gives a detailed description of the RTA-OSEK API calls, listed in alphabetical order.

3.1 OSEK Conformance

RTA-OSEK is certified to the requirements of OSEK OS conformance classes BCC1, BCC2, ECC1 and ECC2, and to CCCA and CCCB for OSEK COM.

3.2 AUTOSAR and RTA-OSEK Features

RTA-OSEK also provides features from in AUTOSAR OS Scalability Class 1 (Release 1.0) and unique RTA-OSEK features that are not part of the OSEK OS standard. These are documented explicitly in the following chapters.

3.3 Static and Dynamic Interface

Portability: The static interface is exclusive to RTA-OSEK.

The static interface comprises of a set of RTA-OSEK calls that provide the same operations as their dynamic counterparts, but at lower or no execution time cost. The static versions of the API calls may be more efficient and provide static error checking, as determined by RTA-OSEK. The static versions of the API calls are derived from the dynamic calls in the following ways:

<pre> APIName(param1) → APIName_param1() APIName(param1, param2) → APIName_param1_param2() APIName(param1, param2) → APIName_param1(param2) </pre>
--

Some of the API calls listed in Chapter 3 have static versions. The static versions are shown for each applicable call. You can find a summary of these calls in Section 11.2 of this guide.

3.4 The API Call Template

Each API call is described in this guide using the following standard format:

The title gives the name of the API call.

A brief description of the API call is provided.

Function declaration:

Interface in C syntax.

Static interface: The static version of the API call is listed (if applicable).

Parameters:

Parameter	Input/Output	Description
Parameter Name	Input/Output	Description.

Description:

Explanation of the functionality of the API call.

Error codes:

Build	Code	Description
Build level of RTA-OSEK	Return values.	Description of return value.

Calling environment:

Environment	Valid
Idle task	
Task	
Category interrupts 1	
Category interrupts 2	

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	
Posttask	
Error	
Overrun	

Symbols used are:

✓ = Valid

✗ = Invalid

🌀 = Supported in RTA-OSEK, but may not be portable to other OSEK implementations.

Portability:

Specifies the portability between other implementation of the same version of the relevant OS standard as follows:

OSEK = API call is portable between OSEK OS implementations.

AUTOSAR = API call is portable between AUTOSAR OS implementations.

RTA-OSEK = API call is only portable between implementations of RTA-OSEK.

Notes:

Usage restrictions and notes for the API call.

See also:

List of related API calls.

3.5 ActivateTask()

Activates a task.

Function declaration:

```
StatusType ActivateTask(TaskType TaskID)
```

Static version: `ActivateTask_TaskID(TaskID)`

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task to be activated.


Description:


If task `TaskID` is in the *suspended* state, it is transferred to the *ready* state. If it is already in the *ready* or *running* state and the total number of activations is less than the activation limit (for a BCC2 task), the current activation is placed in the *ready* state on the activation queue.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_LIMIT	Too many task activations of the multiply activated task <code>TaskID</code> . As a consequence the task activation is ignored.
Extended	E_OS_ID	<code>TaskID</code> is not a valid task type.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR

OSEK

Notes:

For correct timing analysis, lower priority tasks must not activate higher priority tasks.

Rescheduling after a call to `ActivateTask()` depends on the calling environment:

- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`.
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Hooks: Rescheduling will not take place until the hook terminates.

See also:

`ChainTask()`
`DeclareTask()`
`GetTaskID()`
`GetTaskState()`
`TerminateTask()`

3.6 ActivateTaskset()

Activate a set of tasks.

Function declaration:

```
StatusType ActivateTaskset(TasksetType TasksetID)
```

Static version: `ActivateTaskset_TasksetID()`

Parameters:

Parameter	Input/Output	Description
TasksetID	Input	Name of the taskset.

Description:

The tasks in taskset `TasksetID` are transferred from the *suspended* state into the *ready* state. If any task in the taskset is already in the *ready* or *running* state, and the task does not support queued activations (i.e the task is not BCC2) or does support queued activations but the queue is full, then current taskset activation is lost.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	TasksetID is not a valid taskset type.
Extended	E_OS_LIMIT	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

For correct timing analysis, lower priority tasks must not activate tasksets containing higher priority tasks.

Rescheduling after a call to `ActivateTaskset()` depends on the calling environment:

- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`.
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Hooks: Rescheduling will not take place until the hook terminates.

See also:

```
AssignTaskset()  
ChainTaskset()  
GetTasksetRef()  
MergeTaskset()  
RemoveTaskset()  
TestEquivalentTaskset()  
TestSubTaskset()
```

3.7 AdvanceSchedule()

Process the next arrivalpoint on the advanced schedule.

Function declaration:

```
StatusType AdvanceSchedule(
    ScheduleType          ScheduleID,
    ScheduleStatusRefType ScheduleStatus)
```

```
Static version: AdvanceSchedule_ScheduleID(
    ScheduleStatusRefType ScheduleStatus)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Name of the schedule.
ScheduleStatus	Output	Reference to schedule status.

Description:

This call indicates to advanced schedule `ScheduleID` that the next arrivalpoint must be processed. The tasks for that arrivalpoint are then activated.

If the final arrivalpoint is processed by this call, the advanced schedule is stopped and a call is made to the advanced schedule callback `Cancel_<ScheduleID>()` device driver function.



Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	ScheduleID is not valid.
Extended	E_OS_LIMIT	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	E_OS_STATE	ScheduleID is not running.
Extended	E_OS_SYS_S_MISMATCH	Schedule contains an advanced counter (call only permitted for ticked).
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	

Environment (Hooks)	Valid
Startup	*

Environment	Valid
Task	
Category 1 interrupts	✘
Category 2 interrupts	

Environment (Hooks)	Valid
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

RTA-OSEK

Notes:

Rescheduling after a call to `AdvanceSchedule()` depends on the calling environment:

- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`.
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Hooks: Rescheduling will not take place until the hook terminates.

If the returned status is *running*, the call must become due again 'expire' ticks after this instance of the call became due. It is the responsibility of the application programmer to ensure that this happens.

For correct timing analysis, lower priority tasks must not advance schedules that result in the activation of higher priority tasks.

Further information can be found in Section 2.1.1.

See also:

```

GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()

```

TickSchedule()

3.8 AssignTaskset()

Assign the members of a taskset to another taskset.

Function declaration:

```
StatusType AssignTaskset(TasksetType DestTaskset,
                          TasksetType SourceTaskset)
```

Parameters:

Parameter	Input/Output	Description
DestTaskset	Output	Name of a writable taskset.
SourceTaskset	Input	Taskset containing tasks to be assigned.

Description:

The `DestTaskset` taskset is updated to contain exactly the tasks in the `SourceTaskset` taskset.

If the `SourceTaskset` and `DestTaskset` are the same taskset, then the membership of the taskset is unchanged as a result of the call.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_TS_INVALID	Either taskset is invalid.
Extended	E_OS_SYS_TS_READONLY	<code>DestTaskset</code> is not writable.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
ChainTaskset()
GetTasksetRef()
MergeTaskset()
RemoveTaskset()
TestEquivalentTaskset()
TestSubTaskset()

3.9 CancelAlarm()

Cancel an alarm.

Function declaration:

```
StatusType CancelAlarm(AlarmType AlarmID)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Alarm name.


Description:

Cancels the alarm `AlarmID`.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_NOFUNC	Alarm <code>AlarmID</code> not in use.
Extended	E_OS_ID	Alarm <code>AlarmID</code> is not valid.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:`DeclareAlarm()``GetAlarm()``GetAlarmBase()``SetAbsAlarm()``SetRelAlarm()`

3.10 ChainTask()

Terminate the calling task and activate a task.

Function declaration:

```
StatusType ChainTask(TaskType TaskID)
```

Static version: ChainTask_TaskID()

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task to activate after terminating the current task.

Description:

This call terminates the calling task. After termination, the specified `TaskID` is activated. Using this call ensures that the succeeding task starts to run at the earliest possible point after the calling task has been terminated.

Internal resources are released automatically.

A task can chain itself without affecting the activation count.

Error codes:

Build	Code	Description
All	E_OS_LIMIT	Too many activations of task <code>TaskID</code> . As a consequence the task activation is ignored.
Extended	E_OS_ID	<code>TaskID</code> is not a valid task type.
Extended	E_OS_CALLEVEL	Call at interrupt level.
Extended	E_OS_RESOURCE	Calling task still occupies resource.
Extended	E_OS_SYS_IDLE	Called from idle task.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	✘
Task	✓
Category 1 interrupts	✘
Category 2 interrupts	✘

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

AUTOSAR
OSEK

Notes:

`ChainTask()` must be called as the last statement in a task entry function.

If called successfully, `ChainTask()` does not return to the call level and the status cannot be evaluated.

If called unsuccessfully from a heavyweight task in the Extended build an error is returned. This can then be evaluated in the application.

In the Extended build a lightweight task will always be terminated, even if there is an error. The error hook, if configured, will be called.

If the current task has a resource that has not been released, then the call is unsuccessful.

Tasks declared as lightweight can only call `ChainTask()` from the function declared with the corresponding `TASK()` macro. The call must be a top-level statement, not embedded in any expression.

For correct timing analysis, lower priority tasks must not chain higher priority tasks.

See also:

`ActivateTask()`
`DeclareTask()`
`GetTaskState()`
`TerminateTask()`

3.11 ChainTaskset()

Terminate the calling task and activate a set of tasks.

Function declaration:

```
StatusType ChainTaskset(TasksetConstType TasksetID)
```

Static version: ChainTaskset_TasksetID()

Parameters:

Parameter	Input/Output	Description
TasksetID	Input	Name of the taskset containing the tasks to be activated after terminating the current task.

Description:


This API call causes the termination of the calling task. After termination of the calling task, the succeeding tasks in `TasksetID` are activated. Using this call ensures that the succeeding tasks start to run at the earliest possible point after the calling task has been terminated. Internal resources are released automatically.

A task can chain itself without affecting the activation count.

Error codes:

Build	Code	Description
Extended	E_OS_ID	TasksetID not a valid taskset type.
Extended	E_OS_LIMIT	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	E_OS_CALLEVEL	Call at interrupt level.
Extended	E_OS_SYS_IDLE	Called from idle task.
Extended	E_OS_RESOURCE	Calling task still occupies resources.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	x
Task	
Category 1 interrupts	x
Category 2 interrupts	x

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

`ChainTaskset()` must be called as the last statement in a task entry function.

If called successfully, `ChainTaskset()` does not return to the call level and the status cannot be evaluated.

If called unsuccessfully from a heavyweight task in the Extended build, an error is returned. This can then be evaluated in the application.

In the Extended build a lightweight task will always be terminated, even if there is an error. The error hook, if configured, will be called.

If the current task has a resource that has not been released, then the call is unsuccessful.

Tasks declared as lightweight can only call `ChainTaskset()` from the function declared with the corresponding `TASK()` macro. The call must be a top-level statement, not embedded in any expression.

For correct timing analysis, lower priority tasks must not chain tasksets containing higher priority tasks.

See also:

`ActivateTaskset()`

`AssignTaskset()`

`GetTasksetRef()`

`MergeTaskset()`

`RemoveTaskset()`

`TestEquivalentTaskset()`

`TestSubTaskset()`

3.12 ClearEvent()

Clear the calling task's events.

Function declaration:

```
StatusType ClearEvent(EventMaskType Mask)
```

Parameters:

Parameter	Input/Output	Description
Mask	Input	Mask of the events to be cleared.


Description:

Clears the calling task's events as specified by `Mask`.

Error codes:

Build	Code	Description
Standard	E_OK	No error.
Extended	E_OS_ACCESS	Call not from extended task.
Extended	E_OS_CALLEVEL	Call at interrupt level.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✗

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

Can only be called from an extended task.
Any events that are not set in the event mask remain unchanged.

See also:`DeclareEvent()``GetEvent()``SetEvent()``WaitEvent()`

3.13 CloseCOM()

Release low-level hardware resources needed for COM communication.

Function declaration:

```
StatusType CloseCOM(void)
```

Parameters:

None.



Description:

`CloseCOM()` is used to release low-level resources that are used for COM communication.

Error codes:

Build	Code	Description
All	E_OK	No error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✗
Shutdown	✓
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

To finish COM processing, use `StopCOM()` and then use `CloseCOM()`. For portability, interrupts must be masked when the call is made from task level.

The RTA-OSEK libraries contain a default implementation of `CloseCOM()` that simply returns `E_OK`. Application programmers are free to supply their own implementation of `CloseCOM()`, which will be linked into the application in preference to this default.

See also:

```
CloseCOM()  
DeclareMessage()  
InitCOM()  
MessageInit()  
ReadFlag()  
ReceiveMessage()  
ResetFlag()  
SendMessage()  
StartCOM()  
StopCOM()
```


3.14 DisableAllInterrupts()

Disables all interrupts.

Function declaration:

```
void DisableAllInterrupts(void)
```

Parameters:

None.

Description:

This API call allows disables all interrupts that can be disabled in the hardware.


The state before the call is made is saved for the `EnableAllInterrupts()` call.







This call is intended to start a critical section of the code. This critical section must be finished by calling `EnableAllInterrupts()`. No API calls are allowed within this critical section.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	
Posttask	
Error	
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

This API call does not support nesting. If nesting is required for critical sections `SuspendAllInterrupts()` and `ResumeAllInterrupts()` should be used.

See also:

```
DeclareISR()  
EnableAllInterrupts()  
ResumeAllInterrupts()  
ResumeOSInterrupts()  
SuspendAllInterrupts()  
SuspendOSInterrupts()
```

3.15 EnableAllInterrupts()

Ends a critical section started by `DisableAllInterrupts()`.

Function declaration:

```
void EnableAllInterrupts(void)
```

Parameters:

None.

Description:


This call restores the state saved by `DisableAllInterrupts()`.







The critical section of code, started by a `DisableAllInterrupts()` call, is finished by calling `EnableAllInterrupts()`. No API calls are allowed within this critical section.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	
Posttask	
Error	
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:

```
DeclareISR()  
DisableAllInterrupts()  
ResumeAllInterrupts()  
ResumeOSInterrupts()  
SuspendAllInterrupts()
```

SuspendOSInterrupts()

3.16 GetActiveApplicationMode()

Get the current application mode.

Function declaration:

```
AppModeType GetActiveApplicationMode(void)
```

Parameters:

None.

Description:

This call returns the current application mode.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	✓
Task	✓
Category interrupts 1	✗
Category interrupts 2	✓

Environment (Hooks)	Valid
Startup	✓
Shutdown	✓
Pretask	✓
Posttask	✓
Error	✓
Overrun	✓

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:

None.

3.17 GetAlarm()

Returns the number of ticks before the alarm next expires.

Function declaration:

```
StatusType GetAlarm(AlarmType AlarmID,
                    TickRefType Tick)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Name of an alarm.
Tick	Output	Relative value, in ticks, before alarm expires.


Description:


Returns the relative value in ticks before the alarm `AlarmID` expires.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_NOFUNC	Alarm <code>AlarmID</code> is not used.
Extended	E_OS_ID	Alarm <code>AlarmID</code> is not valid.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:

CancelAlarm()
DeclareAlarm()
GetAlarmBase()
SetAbsAlarm()
SetRelAlarm()

3.18 GetAlarmBase()

Get the alarm base characteristics.

Function declaration:

```
StatusType GetAlarmBase(AlarmType AlarmID,
                        AlarmBaseRefType Info)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Alarm name.
Info	Output	Reference to structure with constants of alarm base.


Description:


`GetAlarmBase()` reads the alarm base characteristics. The return value `Info` is a structure in which the information of data type `AlarmBaseType` is stored.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	Alarm <code>AlarmID</code> is not valid.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

For more information on `AlarmBaseType` see Chapter 2.1.1.

See also:

CancelAlarm()
DeclareAlarm()
GetAlarm()
SetAbsAlarm()
SetRelAlarm()

3.19 GetArrivalpointDelay()

Get the delay between one arrivalpoint and the successor.

Function declaration:

```
StatusType GetArrivalpointDelay(
    ArrivalpointType ArrivalpointID,
    TickRefType      Delay)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint name.
Delay	Output	Reference to tick value.

Description:

Updates `Delay` from the 'delay' property of the arrivalpoint.

If the 'delay' property is zero, the delay is equal to the modulus of the associated schedule.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.20 GetArrivalpointNext()

Get the successor of an arrivalpoint.

Function declaration:

```
StatusType GetArrivalpointNext(
    ArrivalpointType ArrivalpointID,
    ArrivalpointRefType ArrivalpointNextID)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint name.
ArrivalpointNextID	Output	Reference to arrivalpoint.

Description:

The return value `ArrivalpointNextID` returns a handle to the 'next' (arrivalpoint) property of the arrivalpoint supplied.

The arrivalpoint `ArrivalpointNextID` can be null, which is represented by zero for this parameter.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.21 GetArrivalpointTasksetRef()

Get a reference to an arrivalpoint's taskset.

Function declaration:

```
StatusType GetArrivalpointTasksetRef(
    ArrivalpointType ArrivalpointID,
    TasksetRefType   TasksetID)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint name.
TasksetID	Output	Name of a taskset.

Description:

TasksetID is a pointer to a TasksetType. This call returns a pointer to the taskset embedded in the arrivalpoint. It can then be used in the taskset API calls and, if the arrivalpoint is writable, the arrivalpoint's taskset can be updated directly.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

```
AdvanceSchedule()  
GetArrivalpointDelay()  
GetArrivalpointNext()  
GetScheduleNext()  
GetScheduleStatus()  
GetScheduleValue()  
SetArrivalpointDelay()  
SetScheduleNext()  
StartSchedule()  
StopSchedule()  
TestArrivalpointWritable()  
TickSchedule()
```


3.22 GetCounterValue()

Get the counter value.

Function declaration:

```
StatusType GetCounterValue(CounterType CounterID,
                            TickRefType CountRef)
```

Parameters:

Parameter	Input/Output	Description
CounterID	Input	Counter name.
CountRef	Output	Reference to tick value.

Description:

Returns the current value of the specified counter `CounterID` in `CountRef`.

For advanced counters, the user callback `Get_<CounterID>` will be called.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_COUNTER_INVALID	Invalid counter.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

The application should serialize the use of `GetCounterValue()` and `Tick_<CounterID>()/osAdvanceCounter_<CounterID>` to ensure that a meaningful value is obtained.

See also:

```
DeclareCounter()  
IncrementCounter()  
InitCounter()  
osAdvanceCounter_<CounterID>()  
Tick_<CounterID>()
```

3.23 GetEvent()

Get the events for the specified task.

Function declaration:

```
StatusType GetEvent (TaskType TaskID,
                    EventMaskRefType Event)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Name of the task.
Event	Output	Reference to the memory of the return data.


Description:


This call returns the current state of all event bits of the task `TaskID` (not the events that the task is waiting for).

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ACCESS	Referenced task is not an extended task.
Extended	E_OS_ID	Task <code>TaskID</code> is invalid.
Extended	E_OS_STATE	Events cannot be set because the referenced task is in the <i>suspended</i> state.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category interrupts 1	✗
Category interrupts 2	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

TaskID must identify an extended task.

See also:

ClearEvent()
DeclareEvent()
SetEvent()
WaitEvent()

3.24 GetISRID()

Returns the identity of the current ISR.

Function declaration:

```
ISRType GetISRID(void)
```

Parameters:

None.

Description:

This function returns the identity of the currently executing ISR.

Return Parameter:

Build	Code	Description
All	INVALID_ISR	Caller is not an active ISR.

Calling environment:

Environment	Valid
Idle task	✘
Task	✘
Category interrupts 1	✘
Category interrupts 2	✓

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	
Posttask	
Error	
Overrun	

Portability:

AUTOSAR

Notes:

None.

See also:

GetTaskID()

3.25 GetMessageResource()

Get the message resource.

Function declaration:

```
StatusType GetMessageResource(SymbolicName Message)
```

Parameters:

Parameter	Input/Output	Description
Message	Input	Symbolic name of the message.

Description:

For a system configured to have message resources, this call occupies the resource for message object `Message` and then sets the status of the message object to busy. It can be used, in conjunction with `ReleaseMessageResource()`, to protect the message object against concurrent access.



Tasks and/or Category 2 ISRs can share the message resource.

A critical section must always be exited using `ReleaseMessageResource()`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_COM_ID	Message parameter is invalid.
Extended	E_OS_ACCESS	Either: An attempt to get a resource that is already occupied by any task, Category 2 ISR or statically assigned priority of the calling task. Or: The interrupt routine is higher than the calculated ceiling priority.
Extended	E_OS_SYS_R_PERMISSION	Called by a task that has not declared that it uses the message resource.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✘
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

OSEK

Notes:

Obtaining resources must be strictly nested. Nested occupation of the same resource is forbidden.

It is not permitted to terminate a task or leave an ISR while it holds resources.

Message resources should only be used for messages whose mode is `WithoutCopy`

See also:

`GetMessageStatus()`

`ReleaseMessageResource()`

3.26 GetMessageStatus()

Return the message status.

Function declaration:

```
StatusType GetMessageStatus(SymbolicName Message)
```

Parameters:

Parameter	Input/Output	Description
Message	Input	Symbolic name of the message object.


Description:





This API call returns the current status of the message object `Message`.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_COM_BUSY	Message identified by <code>Message</code> already set to busy.
All	E_COM_LIMIT	Overflow of FIFO of the queued message, identified by <code>Message</code> , has occurred.
All	E_COM_NOMSG	No message available (applies to a queued message where the FIFO is empty).
Extended	E_COM_ID	Message parameter is invalid.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	
Posttask	
Error	
Overrun	

Portability:

OSEK

Notes:

None.

See also:`GetMessageResource()``ReleaseMessageResource()`

3.27 GetResource()

Get a resource.

Function declaration:

```
StatusType GetResource(ResourceType ResID)
```

Static version: `GetResource_ResID()`

Parameters:

Parameter	Input/Output	Description
ResID	Input	The resource to be held.

Description:

Protects a critical section in the code against concurrent access by other tasks or ISRs.


Tasks and/or Category 2 ISRs can share the resource.

A critical section must always be left using `ReleaseResource()`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ACCESS	Either: An attempt to get a resource that is already occupied by any task, Category 2 ISR or statically assigned priority of the calling task. Or: The interrupt routine is higher than the calculated ceiling priority.
Extended	E_OS_ID	ResID is not a valid resource type.
Extended	E_OS_SYS_R_PERMISSION	Called by a task that has not declared that it uses the message resource.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

Nested resource occupation is only allowed if the inner critical sections are completely executed within the surrounding critical section (strictly nested). Nested occupation of the same resource is also forbidden.

If nested occupation of the same resource is required, then the appropriate linked resources should be configured in the RTA-OSEK GUI.

Calls that put the *running* task into any other state, such as `ChainTask()`, `Schedule()`, `TerminateTask()` or `WaitEvent()`, must not be used in critical sections.

See also:

`DeclareResource()`
`ReleaseResource()`

3.28 GetScheduleNext()

Get the next arrivalpoint to be processed by a schedule.

Function declaration:

```
StatusType GetScheduleNext (
    ScheduleType      ScheduleID,
    ArrivalpointRefType ArrivalpointID)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule name.
ArrivalpointID	Output	Reference to next arrivalpoint.




Description:





Updates `ArrivalpointID` from the 'next' property of the schedule.
Defined to be zero when 'next' is null.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_S_INVALID	Invalid schedule handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.29 GetScheduleStatus()

Get the current status of a schedule.

Function declaration:

```
StatusType GetScheduleStatus(
    ScheduleType      ScheduleID,
    ScheduleStatusRefType ScheduleStatus)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule name.
ScheduleStatus	Output	Reference to schedule status.

Description:

Updates `ScheduleStatus` with the current state of the schedule and the number of ticks of the schedule's counter between now and when the schedule is due to process the 'next' arrivalpoint.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_S_INVALID	Invalid schedule handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

For more information on `ScheduleStatusType`, see Section 2.1.1.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.30 GetScheduleTableStatus()

Gets the current status of a schedule table.

Function declaration:

```
StatusType GetScheduleTableStatus(
    ScheduleTableType ScheduleID,
    ScheduleTableStatusRefType ScheduleStatus)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Name of schedule for which status is requested.
ScheduleStatus	Output	Reference to ScheduleStatus.

Description:

Updates ScheduleStatus with the current state of the schedule table.

Error codes:

Build	Code	Description
All	E_OK	No error
All	E_OS_ID	Invalid schedule table handle.

Calling environment:

Environment	Valid
Idle task	✓
Task	✓
Category interrupts 1	✗
Category interrupts 2	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	🔄

Portability:

AUTOSAR

Notes:

None.

See also:

```
NextScheduleTable()  
StartScheduleTable()  
StopScheduleTable()
```

3.31 GetScheduleValue()

Get the 'now' property of a schedule.

Function declaration:

```
StatusType GetScheduleValue(ScheduleType ScheduleID,
                             TickRefType Tick)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule name.
Tick	Output	Reference to tick value.

Description:




Sets `Tick` to the 'now' property of the counter contained in the schedule. This has a value between 0 and the schedule modulus-1.





When the schedule is an advanced schedule, the user supplied device driver, `Now_<ScheduleID>()` function is called.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_S_INVALID	Invalid schedule.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

The application should serialize the use of `GetScheduleValue()` and either `TickSchedule()/AdvanceSchedule()` to ensure that a meaningful value is obtained.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.32 GetStackOffset()

Gets the current stack pointer.

Function declaration:

```
GetStackOffset (StackOffsetRefType StackVal)
```

Parameters:

Parameter	Input/Output	Description
StackVal	Output	Reference to the memory of the return data.

Description:

Returns an indication of the number of bytes on the stack. Depending on the actual target, `StackOffsetType` is either a scalar or a structure of scalars. Further information for a specific target can be found in the appropriate *RTA-OSEK Binding Manual*.

This API is used to assist in the measurement and verification of user code stack usage.

Error codes:

Build	Code	Description
All	E_OK	No error.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

None.

3.33 GetTaskID()

Get the status of the specified task.

Function declaration:

```
StatusType GetTaskID(TaskRefType TaskID)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Output	Name of the task that is currently <i>running</i> .

Description:


Returns the `TaskID` of the task that is currently *running*.


If no task is *running* or if the idle task currently *running*, this call returns `INVALID_TASK`.

Error codes:

Build	Code	Description
All	E_OK	No error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

This API call is intended to be used by library functions and hook routines.

See also:

```
ActivateTask()  
ChainTask()  
DeclareTask()  
GetISRID()  
GetTaskState()  
TerminateTask()
```

3.34 GetTasksetRef()

Create a singleton taskset from the specified task.

Function declaration:

```
StatusType GetTasksetRef(TaskType TaskID,
                          TasksetRefType TasksetRef)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task name.
TasksetRef	Output	Reference to a writable TasksetType handle.

Description:

The TasksetType pointed to by TasksetRef is updated so that it points to a taskset containing the single task represented by TaskID.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	TaskID is not a valid task type.
Extended	E_OS_SYS_IDLE	TaskID is the idle task.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
AssignTaskset()
ChainTaskset()
MergeTaskset()
RemoveTaskset()
TestEquivalentTaskset()
TestSubTaskset()

3.35 GetTaskState()

Get the task state.

Function declaration:

```
StatusType GetTaskState(TaskType TaskID,
                        TaskStateRefType State)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task name.
State	Output	Reference to the state of the task TaskID.

Description:


Returns the state of a task at the time of calling `GetTaskState()`. The state is exactly one of:


- RUNNING
- READY
- SUSPENDED
- WAITING

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	TaskID is not a valid task type.
Extended	E_OS_SYS_IDLE	TaskID is the idle task.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

Within a fully preemptive system, this call only provides a meaningful result if the task runs in an interrupt disabling state at the time of calling.

See also:

`ActivateTask()`
`ChainTask()`
`DeclareTask()`

3.36 IncrementCounter()

Increments a counter by one tick.

Function declaration:

```
StatusType IncrementCounter(CounterType CounterID)
```

Static version: See `Tick_<CounterID>()`

Parameters:

Parameter	Input/Output	Description
CounterID	Input	Counter name

Description:

This call indicates to counter `CounterID` that one tick has elapsed.

For each attached alarm, if sufficient ticks have elapsed, the action(s) for that alarm are executed.

For an expiry point on an attached and running Schedule Table, if sufficient ticks have elapsed, the action(s) for that expiry point are executed.

The match value for the next expiry of the alarm is set if the alarm was started with a cycle time greater than zero.




The match value for the next expiry point on the attached and running schedule table (if any) is set.

Error codes:

`IncrementCounter()` can raise `E_OS_ID` as a direct error. Additional, secondary, errors can result from the activation of tasks/setting of events when alarms expire or expiry points are processed. In all cases the `ErrorHook()` is invoked.

Build	Code	Description
All	<code>E_OK</code>	No error.
Extended	<code>E_OS_ID</code>	CounterID invalid
Extended	<code>E_OS_LIMIT</code>	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	<code>E_OS_STATE</code>	Events cannot be set as the referenced task is in the <i>suspended</i> state.
Extended	<code>E_OS_ACCESS</code>	Referenced task is not an extended task.
Extended	<code>E_OS_SYS_CONFIG_ERROR</code>	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

AUTOSAR

Notes:

Rescheduling after a call to `IncreaseCounter()` depends on the calling environment.

- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Hooks: Rescheduling will not take place until the hook terminates.
- For correct timing analysis, lower priority tasks must not tick counters that result in the expiry of alarms, leading to the activation of higher priority tasks

See also:

```

DeclareCounter()
GetCounterValue()
InitCounter()
osAdvanceCounter_<CounterID>()
Tick_<CounterID>()

```

3.37 InitCOM()

Initializes the low-level resources necessary for COM.

Function declaration:

```
StatusType InitCOM(void)
```

Parameters:

None.



Description:

`InitCOM()` is used to initialize all the low-level resources used by COM. It may be called before starting RTA-OSEK or in any task from within any kernel startup hook routines.

Error codes:

Build	Code	Description
All	E_OK	No error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✓
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

OSEK

Notes:

The recommended procedure is to use `InitCOM()` and then to call `StartCOM()`.

For portability, interrupts must be masked when the call is made from task level.

The RTA-OSEK libraries contain a default implementation of `InitCOM()` that simply returns `E_OK`. Application programmers are free to supply their own implementation of `InitCOM()`, which will be linked into the application in preference to this default.

In the current implementation of RTA-OSEK, `InitCOM()` is unnecessary because RTA-OSEK supports only intra-processor messaging. However, it is advisable to make use of the call to ensure portability.

See also:

`DeclareMessage()`
`ClearEvent()`
`CloseCOM()`
`MessageInit()`
`ReadFlag()`
`ReceiveMessage()`
`ResetFlag()`
`SendMessage()`
`StartCOM()`
`StopCOM()`

3.38 InitCounter()

Initialize the counter to a given tick value.

Function declaration:

```
StatusType InitCounter(CounterType Counter,
                       TickType Start)
```

Parameters:

Parameter	Input/Output	Description
CounterID	Input	Counter name.
Start	Input	Tick value.

Description:




Sets the specified counter to the absolute value specified by *Start*.


For advanced counters, the user callback function `Set_<CounterID>` will be called.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_STATE	Counter is already running.
Extended	E_OS_SYS_COUNTER_INVALID	Invalid counter.
Extended	E_OS_VALUE	Start exceeds maxallowedvalue.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

RTA-OSEK initializes ticked counters to zero automatically at startup.

The application should serialize the use of `InitCounter()` and `Tick_<CounterID>()/osAdvanceCounter_<CounterID>()`

See also:

```
DeclareCounter()  
GetCounterValue()  
IncrementCounter()  
osAdvanceCounter_<CounterID>()  
Tick_<CounterID>()
```


3.39 MergeTaskset()

Merge two tasksets.

Function declaration:

```
StatusType MergeTaskset(TasksetType DestTaskset,
                        TasksetType SourceTaskset)
```

Parameters:

Parameter	Input/Output	Description
DestTaskset	Output	Name of a writable taskset.
SourceTaskset	Input	Name of taskset containing tasks to merge.

Description:

The `DestTaskset` is merged with the `SourceTaskset` and the result is returned in the `DestTaskset`. The operation is equivalent to set union.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_TS_INVALID	Either taskset invalid.
Extended	E_OS_SYS_TS_READONLY	<code>DestTaskset</code> not writable.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
AssignTaskset()
ChainTaskset()
GetTasksetRef()
RemoveTaskset()
TestEquivalentTaskset()
TestSubTaskset()

3.40 NextScheduleTable()

Stops one schedule table and starts another.

Function declaration:

```
StatusType NextScheduleTable(
    ScheduleTableType ScheduleTableID_current,
    ScheduleTableType ScheduleTableID_next)
```

Parameters:

Parameter	Input/Output	Description
ScheduleTableID_current	Input	Current schedule table handle.
ScheduleTableID_next	Input	Next schedule table handle

Description:

Schedule table `ScheduleTableID_next` will be started after the current cycle of expiry points in `ScheduleTableID_current` has finished processing.

If `ScheduleTableID_current` is a single-shot schedule table then `ScheduleTableID_next` will be set to start during the processing of the final expiry point of `ScheduleTableID_current`.

If `ScheduleTableID_current` is a periodic schedule table then `ScheduleTableID_next` will be set to start at the end of the current period.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_ID	Invalid schedule table handle.
All	E_OS_NOFUNC	<code>ScheduleTableID_current</code> has not been started
All	E_OS_STATE	<code>ScheduleTableID_next</code> is already in started or next state

Calling environment:

Environment		Valid
Idle task		✓
Task		✓
Category interrupts	1	✗
Category interrupts	2	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR

Notes:

If `ScheduleTableID_current` is stopped before `ScheduleTableID_next` is started, then `ScheduleTableID_next` will not be started.

See also:

`GetScheduleTableStatus()`
`StartScheduleTable()`
`StopScheduleTable()`

3.41 osAdvanceCounter_<CounterID>()

Process the next alarm and/or schedule table expiry point associated with the advanced counter.

Function declaration:

```
StatusType osAdvanceCounter_<CounterID>(
    ScheduleStatusRefType osStat)
```

Parameters:

Parameter	Input/Output	Description
osStat	Input	Reference to schedule status.

Description:




This call indicates that an object using advanced counter `CounterID` that the next alarm and/or expiry point must be processed. The alarm action(s) and/or expiry point action(s) are then executed.

If there are no set alarms and/or expiry points to process then the advanced counter is stopped and a call is made to the user device driver callback `Cancel_<CounterID>()`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_STATE	<code>CounterID</code> is not running.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

Rescheduling after a call to `osAdvanceCounter_<CounterID>()` depends on the calling environment:

- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`.
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Hooks: Rescheduling will not take place until the hook terminates.

If the returned status is *running*, the call must become due again 'expire' ticks after this instance of the call became due. It is the responsibility of the application programmer to ensure that this happens.

Further information can be found in Section 2.1.1.

For correct timing analysis, lower priority tasks must not advance counters that result in the activation of higher priority tasks.

See also:

```
DeclareCounter()  
GetCounterValue()  
IncrementCounter()  
InitCounter()  
Tick_<CounterID>()
```

3.42 osResetOS()

Resets the operating system data structures.

Function declaration:

```
void osResetOS(void)
```

Parameters:

Parameter	Input/Output	Description
Parameter Name	Input/Output	Description.


Description:

Resets key OS data structures so that the OS can be restarted safely.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✘
Category interrupts 1	✘
Category interrupts 2	✘

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

RTA-OSEK

Notes:

The application must be in the idle task with all other OS objects, such as alarms, schedule tables and schedules, inactive when this API is called.

See also:

StartOS()
ShutdownOS().

3.43 ReadFlag()

Read the status of the message flag associated with an OSEK COM message.

Function declaration:

```
FlagValue ReadFlag(FlagType FlagName)
```

Parameters:

Parameter	Input/Output	Description
FlagValue	Result	State of the flag <code>FlagName</code> .
FlagName	Input	Message flag name.

Description:

`ReadFlag()` returns the value of the specified notification flag `FlagName`. The call returns `TRUE` if the message has arrived and `FALSE` otherwise.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	
Posttask	
Error	
Overrun	

Portability:

OSEK

Notes:

None.

See also:

```
DeclareMessage()  
CloseCOM()  
InitCOM()  
MessageInit()  
ReceiveMessage()  
ResetFlag()  
SendMessage()  
StartCOM()  
StopCOM()
```

3.44 ReceiveMessage()

Receive the specified OSEK COM message.

Function declaration:

```
StatusType ReceiveMessage(SymbolicName Message,
                          AccessNameRef Data)
```

Parameters:

Parameter	Input/Output	Description
Message	Input	Symbolic name of the message.
Data	Output	Reference to the message data field to store the received data.


Description:

`ReceiveMessage()` results in the specified message being delivered according to the message copy configuration. For `WithCopy` the message object identified by `Message` is copied to the message data field referenced by `Data`. For `WithoutCopy` the application accesses the message object directly (this means that only the returned status is significant).

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_COM_LIMIT	The FIFO for queued message <code>Message</code> has become full and at least one message has been lost as a result.
All	E_COM_NOMSG	The FIFO for queued message <code>Message</code> is empty.
Extended	E_COM_ID	Parameter <code>Message</code> is invalid.
Extended	E_COM_SYS_STOPPED	COM not started.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✘
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

OSEK.

Notes:

None.

See also:

DeclareMessage()
 CloseCOM()
 InitCOM()
 MessageInit()
 ReceiveMessage()
 ResetFlag()
 SendMessage()
 StartCOM()
 StopCOM()

3.45 ReleaseMessageResource()

Release a previously held message resource.

Function declaration:

```
StatusType ReleaseMessageResource(SymbolicName Message)
```

Parameters:

Parameter	Input/Output	Description
Message	Input	Symbolic name of the message.



Description:

This call sets the status of the `Message` object to not busy. It then releases the message resource. This indicates the end of the critical section protected by the specified message resource.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_COM_ID	Invalid <code>Message</code> parameter.
Extended	E_OS_ACCESS	Attempt to release a resource whose ceiling priority is lower than the calling task.
Extended	E_OS_NOFUNC	Either attempting to release a resource that this task or Category 2 ISR does not currently hold or failing to observe strict nesting of resource locks.
Extended	E_OS_SYS_R_PERMISSION	Called by a task that has not declared that it uses the message resource.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

OSEK

Notes:

API calls that change the status of the task while it holds resources, such as `Schedule()`, `TerminateTask()` and `WaitEvent()` (and possibly `ActivateTask()` in the case of upward activation) cannot be made.

See also:

`GetMessageResource()`
`GetMessageStatus()`

3.46 ReleaseResource()

Release a previously held resource.

Function declaration:

```
StatusType ReleaseResource(ResourceType ResID)
```

Static version: `ReleaseResource_ResID()`

Parameters:

Parameter	Input/Output	Description
ResID	Input	Resource name.


Description:

Releases the specified resource, indicating the end of the protected critical section.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ID	ResID is not a valid resource type.
Extended	E_OS_ACCESS	Attempt to release a resource whose ceiling priority is lower than the calling task.
Extended	E_OS_NOFUNC	Either attempting to release a resource that this task or Category 2 ISR does not currently hold or failing to observe strict nesting of resource locks.
Extended	E_OS_SYS_R_PERMISSION	Called by a task that has not declared that it uses the message resource.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

Obtaining resources must be strictly nested. Nested occupation of the same resource is forbidden.

API calls that change the status of the task while it holds resources, such as `Schedule()`, `TerminateTask()` and `WaitEvent()` (and possibly `ActivateTask()` in the case of upward activation) cannot be made.

See also:

`DeclareResource()`
`GetMessageResource()`

3.47 RemoveTaskset()

Remove tasks from a taskset.

Function declaration:

```
StatusType RemoveTaskset (TasksetType DestTaskset,
                          TasksetType SourceTaskset)
```

Parameters:

Parameter	Input/Output	Description
DestTaskset	Output	Name of a writable taskset.
SourceTaskset	Input	Name of taskset containing tasks to remove.

Description:

The taskset that is pointed to by `DestTaskset` has the tasks in `SourceTaskset` removed from it.

If the source and destination are the same taskset, then the membership of this taskset becomes empty as a result of this call.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_TS_INVALID	Either taskset invalid.
Extended	E_OS_SYS_TS_READONLY	DestTaskset not writable.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
AssignTaskset()
ChainTaskset()
GetTasksetRef()
MergeTaskset()
TestEquivalentTaskset()
TestSubTaskset()

3.48 ResetFlag()

Reset the flag associated with an OSEK COM message.

Function declaration:

```
StatusType ResetFlag(FlagType FlagName)
```

Parameters:

Parameter	Input/Output	Description
FlagName	Input	Message flag name.


Description:





ResetFlag() sets the specified notification flag FlagName to FALSE.

Error codes:

Build	Code	Description
All	E_OK	No error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	
Posttask	
Error	
Overrun	

Portability:

OSEK

Notes:

None.

See also:

```
DeclareMessage()  
CloseCOM()  
InitCOM()  
MessageInit()  
ReadFlag()  
ReceiveMessage()  
SendMessage()  
StartCOM()  
StopCOM()
```

3.49 ResumeAllInterrupts()

Resumes processing of interrupts.

Function declaration:

```
void ResumeAllInterrupts(void)
```

Parameters:

None.

Description:


This API call marks the end of a section of code that is protected from interrupts. The section must have been commenced using the `SuspendAllInterrupts()` call.




Interrupt processing is restored to that in effect before the `SuspendAllInterrupts()` call.

Error codes:

Build	Code	Description
Extended	E_OS_STATE	Interrupts were not previously suspended

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

This API call and its counterpart, `SuspendAllInterrupts()`, work by adjusting the interrupt processing level of the processor.

API calls are not permitted in critical regions between `SuspendAllInterrupts()` and `ResumeAllInterrupts()`. `SuspendAllInterrupts()` and `ResumeAllInterrupts()` calls can, however, be nested.

In the Extended build of RTA-OSEK, the error hook can be called with `E_OS_STATE` if there is no matching suspend call.

When API calls are made in the critical regions, combined resources should be used to achieve mutual exclusion with ISRs.

See also:

```
DeclareISR()  
DisableAllInterrupts()  
EnableAllInterrupts()  
ResumeOSInterrupts()  
SuspendAllInterrupts()  
SuspendOSInterrupts()
```

3.50 ResumeOSInterrupts()

Restore interrupt processing of Category 2 interrupts.

Function declaration:

```
void ResumeOSInterrupts(void)
```

Parameters:

None.


Description:

This call follows a call to `SuspendOSInterrupts()` and re-enables any Category 2 interrupts that were enabled prior to the `SuspendOSInterrupts()` call.

Error codes:

Build	Code	Description
Extended	E_OS_STATE	Interrupts were not previously suspended

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

This API call and its counterpart, `SuspendOSInterrupts()`, work by adjusting the interrupt processing level of the processor.

API calls are not permitted in critical regions between `SuspendOSInterrupts()` and `ResumeOSInterrupts()`. `SuspendOSInterrupts()` and `ResumeOSInterrupts()` calls can, however, be nested.

In the Extended build of RTA-OSEK, the error hook can be called with `E_OS_STATE` if there is no matching `SuspendOSInterrupts()` call.

When API calls are made in the critical regions, combined resources should be used to achieve mutual exclusion with ISRs.

See also:

```
DeclareISR()  
DisableAllInterrupts()  
EnableAllInterrupts()  
ResumeAllInterrupts()  
SuspendAllInterrupts()  
SuspendOSInterrupts()
```


3.51 Schedule()

Provides a rescheduling point for non-preemptive tasks.

Function declaration:

```
StatusType Schedule(void)
```

Parameters:

None.

Description:

This call is made by a task that is non-preemptive or uses an internal resource whenever it is acceptable to allow a higher priority task to preempt it. If any higher priority tasks are *ready* when this call is made, the highest priority task will start executing. If no higher priority tasks are *ready*, the calling task continues execution.

The internal resource is released if the `Schedule()` call results in a higher priority task executing. In this case, the internal resource is held again when the `Schedule()` call returns and the calling task resumes execution.

Error codes:

Build	Code	Description
Standard	E_OK	No error.
Extended	E_OS_CALLEVEL	Call made when task has disabled interrupts.
Extended	E_OS_RESOURCE	Call made while a resource is held.

Calling environment:

Environment	Valid
Idle task	✘
Task	✓
Category 1 interrupts	✘
Category 2 interrupts	✘

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

AUTOSAR
OSEK

Notes:

This call has no effect in fully preemptive tasks with no internal resources.

This call should not be used if timing analysis or stack optimizations are required.

See also:

None.

3.52 SendMessage()

Sends the specified data as an OSEK COM message.

Function declaration:

```
StatusType SendMessage(SymbolicName Message,
                        AccessNameRef Data)
```

Static version: `SendMessage_Message_Data()`

Parameters:

Parameter	Input/Output	Description
Message	Input	Symbolic name of the message.
Data	Input	Reference to message data field to be transmitted.

Description:


`SendMessage()` updates the message object identified by `Message`, depending on the message copy configuration. It also requests transmission of the message object, depending on the transmission mode specified. For `WithCopy`, the message copy referenced by `Data` is copied to the message object referenced by `Message`. For `WithoutCopy`, the application accesses the message object directly.

Where a receiving task is specified, the task is transferred from the *suspended* state into the *ready* state. The operating system ensures that the task code is executed from the first statement. If requested, a callback is made.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_COM_ID	Message is invalid.
Extended	E_COM_LIMIT	Overflow of FIFO of the queued message, identified by <code>Message</code> , has occurred.
Extended	E_COM_SYS_STOPPED	COM not started.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

OSEK

Notes:

For correct timing analysis, lower priority tasks must not send messages that result in the activation of or setting of events for higher priority tasks.

See also:

```

DeclareMessage()
CloseCOM()
InitCOM()
MessageInit()
ReadFlag()
ReceiveMessage()
ResetFlag()
StartCOM()
StopCOM()

```

3.53 SetAbsAlarm()

Set the value of the counter at which the alarm expires.

Function declaration:

```
StatusType SetAbsAlarm(AlarmType AlarmID,
    TickType start, TickType cycle)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Alarm to be set.
start	Input	Value of counter that triggers alarm.
cycle	Input	Number of counter ticks before alarm is triggered again.

Description:


This call starts an alarm running and sets the match value with the associated counter that triggers the alarm. The alarm may be triggered once only (if `cycle` is equal to zero) or repeatedly (`cycle` gives the number of counter ticks before the alarm is triggered again).


When the alarm is triggered, the task associated with the alarm is activated. The alarm can activate a task, set an event or call an alarm callback (depending on configuration).

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_STATE	Alarm AlarmID is currently running.
Extended	E_OS_ID	Alarm AlarmID is not valid.
Extended	E_OS_VALUE	Value of <code>start</code> or <code>cycle</code> is outside the range permitted by the counter (<code>start</code> must be between 0 and <code>maxallowedvalue</code> , <code>cycle</code> must be 0 or between <code>mincycle</code> and <code>maxallowedvalue</code> , inclusive of boundary values).

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

To change values of the alarm, `CancelAlarm()` must be used to stop the alarm running before using `SetAbsAlarm()`.

Care must be taken when `start` is close to the current value of the counter. `SetAbsAlarm()` will produce different results depending on whether the counter has ticked passed the `start` value before the call completes. It will either result in the alarm activating the task almost immediately or when the value `start` is reached again (after the next overrun of the counter).

See also:

`CancelAlarm()`
`DeclareAlarm()`
`GetAlarm()`
`GetAlarmBase()`
`SetRelAlarm()`

3.54 SetArrivalpointDelay()

Set the delay between one arrivalpoint and the successor.

Function declaration:

```
StatusType SetArrivalpointDelay(
    ArrivalpointType ArrivalpointID,
    TickType          Delay)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint to be modified.
Delay	Input	New delay to the next arrivalpoint.




Description:


Sets the 'delay' property of the arrivalpoint to the specified value.
If the 'delay' property is zero, the delay is equal to the modulus of the associated schedule.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.
Extended	E_OS_SYS_AP_READONLY	Arrivalpoint is read-only.
Extended	E_OS_SYS_S_MODULO	Delay is greater than or equal to the modulo for the associated schedule.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

This call can only be used on writable arrivalpoints.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.55 SetArrivalpointNext()

Reconfigure a schedule by changing the successor of an arrivalpoint.

Function declaration:

```
StatusType SetArrivalpointNext(
    ArrivalpointType ArrivalpointID,
    ArrivalpointType ArrivalpointNextID)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint to be modified.
ArrivalpointNextID	Input	New next arrivalpoint.

Description:

Sets the 'next' (arrivalpoint) property of the arrivalpoint `ArrivalpointID`.

The arrivalpoint `ArrivalpointNextID` can be null (represented by zero). This allows a schedule to be terminated at the arrivalpoint `ArrivalpointID`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint <code>ArrivalpointID</code> or <code>ArrivalpointNextID</code> .
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint <code>ArrivalpointID</code> .
Extended	E_OS_SYS_AP_READONLY	Arrivalpoint <code>ArrivalpointID</code> read-only.
Extended	E_OS_SYS_S_MISMATCH	Arrivalpoints have mismatched schedules ('null' arrivalpoint matches all schedules).

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

This call can only be used on writable arrivalpoints.

See also:

```

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

```

3.56 SetEvent()

Set events for a specified task.

Function declaration:

```
StatusType SetEvent (TaskType      TaskID,
                    EventMaskType Mask)
```

Static version: `SetEvent_TaskID_Mask()`

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Name of the task for which events are to be set.
Mask	Input	Mask of the events to be set.


Description:

This API call sets events for task `TaskID` according to `Mask`. If the task is *waiting* for that event, it is immediately set to either *ready* or *running* if the waiting task is now the highest priority task in the system. Multiple events can be set by logically bitwise 'or'ing events.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_ACCESS	Referenced task is not an extended task.
Extended	E_OS_ID	Task <code>TaskID</code> is invalid.
Extended	E_OS_STATE	Events cannot be set as the referenced task is in the <i>suspended</i> state.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

Any unset events in the event mask remain unchanged.

Each static call sets a single event. The call name is constructed from the task name and the event name.

For correct timing analysis, lower priority tasks must not set events for higher priority tasks.

Setting an event for a *suspended* task in the Standard or Timing build has no effect and the status returned is `E_OK`. This is implementation specific behavior for RTA-OSEK.

See also:

`ClearEvent()`
`DeclareEvent()`
`GetEvent()`
`WaitEvent()`

3.57 SetRelAlarm()

Set how many counter ticks occur before the alarm expires.

Function declaration:

```
StatusType SetRelAlarm (AlarmType AlarmID,
                        TickType increment
                        TickType cycle)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Alarm to be set.
increment	Input	Number of counter ticks before alarm is triggered.
cycle	Input	Number of counter ticks before alarm is triggered again.

Description:

This call starts an alarm running and sets the number of counter ticks that will occur before the alarm is triggered. The alarm may be triggered once only (if `cycle` is equal to zero) or repeatedly (`cycle` gives the number of counter ticks before the alarm is triggered again).

When the alarm is triggered, the task associated with the alarm is activated. The alarm can activate a task, set an event or call an alarm callback (depending on configuration).


The behavior when `increment` is zero is dependant on configuration as follows:


- OSEK (default) the alarm occurs in `maxallowedvalue+1` ticks of the counter
- OSEK (SetRelAlarm(,0) disallowed) the alarm is not set and the API call returns `E_OS_VALUE`
- AUTOSAR the alarm is not set and the API call returns `E_OS_VALUE`

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_STATE	Alarm AlarmID is currently running.
Extended	E_OS_ID	Alarm AlarmID is not valid.
Extended	E_OS_VALUE	Value of <code>increment</code> or <code>cycle</code> is outside the range permitted by the counter: <code>increment: >= 0 or > 0</code> <code><= maxallowedvalue</code> <code>cycle: = 0</code> <code>>= mincycle</code> <code><= maxallowedvalue</code>

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
 OSEK

Notes:

To change values of the alarm, the alarm must not be running before using `SetRelAlarm()`.

Care must be taken when the value of `increment` is small, the outcome of `SetRelAlarm()` will produce different results, depending on whether the counter has ticked passed the `increment` value before the call completes. It will either result in the alarm expiring almost immediately or when the value is reached again (after the next overrun of the counter).

See also:

```
CancelAlarm()  
DeclareAlarm()  
GetAlarm()  
GetAlarmBase()  
SetAbsAlarm()
```

3.58 SetScheduleNext()

Indicate the next arrivalpoint to the schedule.

Function declaration:

```
StatusType SetScheduleNext (
    ScheduleType ScheduleID,
    ArrivalpointType ArrivalpointID)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule name.
ArrivalpointID	Input	Next arrivalpoint to be handled by schedule.

Description:




Indicates to the schedule that the next arrivalpoint will be the one given by `ArrivalpointID`.


This function can be used regardless of the schedule state.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.
Extended	E_OS_SYS_S_INVALID	Invalid schedule handle.
Extended	E_OS_SYS_S_MISMATCH	Arrivalpoint <code>ArrivalpointID</code> does not belong to schedule <code>ScheduleID</code> .

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

3.59 ShutdownOS()

Perform operating system shutdown.

Function declaration:

```
void ShutdownOS(StatusType Error)
```

Parameters:

Parameter	Input/Output	Description
Error	Input	Error message to be passed to the shutdown hook.


Description:


Results in the operating system ceasing all activities (processing tasks, interrupts and alarms) and calling the shutdown hook, if defined. If the shutdown hook returns, `ShutdownOS()` enters an endless loop.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✓
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:

`StartOS()`

3.60 StartCOM()

Start the COM service.

Function declaration:

```
StatusType StartCOM(void)
```

Parameters:

None.

Description:



This API call starts the COM by initializing internal data structures. `StartCOM()` calls the `MessageInit()` callback function provided by the application programmer (if it is used) to initialize the application specific message objects.


`StartCOM()` must be called from within a task.

Error codes:

Build	Code	Description
Standard	E_OK	No error.
		Implementation or application specific error code as returned by <code>MessageInit()</code> .

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

OSEK

Notes:

The `StatusType` returned by `MessageInit()` is in turn returned by `StartCOM()`. RTA-OSEK provides a default implementation of `MessageInit()` that returns `E_OK`, unless this is replaced by a user provided version.

See also:

```
CloseCOM()  
DeclareMessage()  
InitCOM()  
MessageInit()  
ReadFlag()  
ReceiveMessage()  
ResetFlag()  
SendMessage()  
StopCOM()
```

3.61 StartOS()

Start the OS.

Function declaration:

```
void StartOS(AppModeType Mode)
```

Parameters:

Parameter	Input/Output	Description
Mode	Input	Application mode.

Description:

`StartOS()` starts RTA-OSEK running in the specified `Mode` application mode. All data structures are initialized and counters are set to zero on startup.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	x
Task	x
Category 1 interrupts	x
Category 2 interrupts	x

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

AUTOSAR
OSEK

Notes:

Code following the call to `StartOS()` becomes the idle task and must not terminate.

See also:

`ShutdownOS()`

3.62 StartSchedule()

Start a stopped schedule.

Function declaration:

```
StatusType StartSchedule(ScheduleType ScheduleID,
                          TickType      When)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule to be started.
When	Input	Counter value at which first arrivalpoint is processed.

Description:

This starts a schedule processing arrivalpoints and sets the schedule state to *running*.

If the schedule state was previously *stopped*, a counter event is set up for the absolute counter value `When`. (Valid values for `When` are in the range 0 to schedule modulus-1, where modulus is the modulus of the associated schedule).

If `When` is equal to 0, the schedule will start the next time the associated counter wraps around. The application programmer must take care of this if it is an advanced schedule.




If the state was previously *running*, this call has no effect.


For an advanced schedule, the device driver `Set_<ScheduleID>()` function is called if the state was previously *stopped*.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.
Extended	E_OS_SYS_S_MODULO	<code>When</code> is greater than or equal to the modulus value of the associated schedule.
Extended	E_OS_SYS_S_INVALID	Invalid schedule handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	✘
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

RTA-OSEK

Notes:

None.

See also:

```

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StopSchedule()
TestArrivalpointWritable()
TickSchedule()

```

3.63 StartScheduleTable()

Starts a schedule table.

Function declaration:

```
StatusType StartScheduleTable(
    ScheduleTableType ScheduleTableID,
    TickType           Offset)
```

Parameters:

Parameter	Input/Output	Description
ScheduleTableID	Input	Schedule table to be started.
Offset	Input	Relative tick value between now and the first alarm expiry

Description:

Starts the schedule table `ScheduleTableID` at its first expiry point after a delay of `Offset` counter ticks.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_ID	Invalid schedule table handle.
All	E_OS_VALUE	Offset is greater than MAXALLOWEDVALUE in case of a ticked counter.
All	E_OS_STATE	Schedule table was already started or the counter to which the schedule table belongs runs another schedule table.

Calling environment:

Environment		Valid
Idle task		✓
Task		✓
Category interrupts	1	✗
Category interrupts	2	✓

Environment (Hooks)	Valid
Startup	✓
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR

Notes:

None.

See also:`GetScheduleTableStatus()``NextScheduleTable()``StopScheduleTable()`

3.64 StopCOM()

Stops the COM service.

Function declaration:

```
StatusType StopCOM(UIntType ShutdownMode)
```

Parameters:

Parameter	Input/Output	Description
ShutdownMode	Input	The shutdown mode.

Description:

StopCOM() causes all COM activity to cease and all resources used by COM to be released. All operations will cease immediately. By implication, data will be lost. StopCOM() will not return until all pending COM operations have completed and their resources can be released.



When StopCOM() has completed successfully, the system is left in a state where StartCOM() can be called to restart COM.


The ShutdownMode that is available is COM_SHUTDOWN_IMMEDIATE. The shutdown will occur immediately without waiting for pending operations to complete.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_COM_BUSY	COM could not shutdown because an application is holding a message resource.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✗
Shutdown	
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

OSEK

Notes:

None.

See also:

InitCOM()
MessageInit()
ReadFlag()
ReceiveMessage()
ResetFlag()
SendMessage()
StartCOM()

3.65 StopSchedule()

Stop a running schedule.

Function declaration:

```
StatusType StopSchedule(ScheduleType ScheduleID)
```

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule to be stopped.

Description:

This call stops a schedule from processing arrivalpoints.




The schedule's state is set to *stopped*.

If `ScheduleID` is an advanced schedule, a call to the user provided advanced schedule driver `Cancel_<ScheduleID>()` is made.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_S_INVALID	Invalid schedule handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
TestArrivalpointWritable()
TickSchedule()

3.66 StopScheduleTable()

Stops a schedule table.

Function declaration:

```
StatusType StopScheduleTable(
    ScheduleTableType ScheduleTableID)
```

Parameters:

Parameter	Input/Output	Description
ScheduleTableID	Input	Schedule table to be stopped.

Description:

Stops the schedule table `ScheduleTableID` from processing any further expiry points.

Error codes:

Build	Code	Description
All	E_OK	No error.
All	E_OS_ID	Invalid schedule table handle.
All	E_OS_NOFUNC	Schedule table was not started.

Calling environment:

Environment	Valid
Idle task	✓
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR

Notes:

None.

See also:

`GetScheduleTableStatus()`
`NextScheduleTable()`
`StartScheduleTable()`.

3.67 SuspendAllInterrupts()

Disables interrupt processing of all interrupts.

Function declaration:

```
void SuspendAllInterrupts(void)
```

Parameters:

None.


Description:




This call disables the interrupt processing of all interrupts. The current interrupt state is saved, so it can be restored later.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	
Shutdown	
Pretask	✓
Posttask	✓
Error	✓
Overrun	

Portability:

AUTOSAR
OSEK

Notes:

This API call and its counterpart, `ResumeAllInterrupts()`, work by adjusting the interrupt processing level of the processor.

API calls are not permitted in critical regions between `SuspendAllInterrupts()` and `ResumeAllInterrupts()`. `SuspendAllInterrupts()` and `ResumeAllInterrupts()` calls can, however, be nested.

When API calls are made in the critical regions, combined resources should be used to achieve mutual exclusion with ISRs.

See also:

```
DeclareISR()  
DisableAllInterrupts()  
EnableAllInterrupts()  
ResumeAllInterrupts()  
ResumeOSInterrupts()  
SuspendOSInterrupts()
```

3.68 SuspendOSInterrupts()

Disables interrupt processing of Category 2 interrupts.

Function declaration:

```
void SuspendOSInterrupts(void)
```

Parameters:

None.


Description:

This call disables interrupt processing of any Category 2 interrupts after saving the current interrupt processing level. The saved interrupt processing level is restored by a call to `ResumeOSInterrupts()`.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✓
Category 2 interrupts	✓

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

This API call and its counterpart, `ResumeOSInterrupts()`, work by adjusting the interrupt processing level of the processor.

No API calls are permitted in the critical region between `SuspendOSInterrupts()` and `ResumeOSInterrupts()`. `SuspendOSInterrupts()` and `ResumeOSInterrupts()` calls can, however, be nested. When API calls are made in the critical regions, combined resources should be used to achieve mutual exclusion with ISRs.

See also:

```
DeclareISR()  
DisableAllInterrupts()  
EnableAllInterrupts()  
ResumeAllInterrupts()  
ResumeOSInterrupts()  
SuspendAllInterrupts()
```

3.69 TerminateTask()

Terminates the calling task.

Function declaration:

```
StatusType TerminateTask(void)
```

Parameters:

None.

Description:

`TerminateTask()` is used to cause the termination of the calling task. This transfers the calling task from the *running* state to the *suspended* state. A BCC2 task with further activations queued moves from the *running* state to the *ready* state.

Internal resources are released automatically.

Error codes:

Build	Code	Description
Extended	E_OS_CALLEVEL	Call at interrupt level.
Extended	E_OS_RESOURCE	Task still occupies resources.
Extended	E_OS_SYS_IDLE	Called from idle task.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	✘
Task	✓
Category 1 interrupts	✘
Category 2 interrupts	✘

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

AUTOSAR
OSEK

Notes:

All resources (other than internal resources) must have been released prior to the call to `TerminateTask()`.

All tasks must end with a call to `TerminateTask()`, `ChainTask()` or `ChainTaskset()`.

Tasks declared as lightweight can only call `TerminateTask()` from the function declared with the `Task()` macro. The call must also be a top-level statement and not embedded in any expression.

If called unsuccessfully from a heavyweight task in the Extended build, an error is returned. This can then be evaluated in the application.

In the Extended build, a lightweight task will always be terminated, even if there is an error. The error hook, if configured, will be called.

See also:

`ActivateTask()`

`ChainTask()`

`DeclareTask()`

`GetTaskState()`

3.70 TestArrivalpointWritable()

Test whether or not an arrivalpoint is writable.

Function declaration:

```
StatusType TestArrivalpointWritable(
    ArrivalpointType ArrivalpointID,
    BooleanRefType WritableFlag)
```

Parameters:

Parameter	Input/Output	Description
ArrivalpointID	Input	Arrivalpoint to be tested.
WritableFlag	Output	Reference to the Boolean flag.




Description:


Sets `WritableFlag` to `TRUE` if the arrivalpoint is writable.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
Extended	E_OS_SYS_AP_NULL	Null arrivalpoint.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

None.

See also:

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TickSchedule()

3.71 TestEquivalentTaskset()

Test whether or not two tasksets contain the same task(s).

Function declaration:

```
StatusType TestEquivalentTaskset(
    TasksetType Taskset1,
    TasksetType Taskset2,
    BooleanRefType EqFlag)
```

Parameters:

Parameter	Input/Output	Description
Taskset1	Input	The first taskset.
Taskset2	Input	The second taskset.
EqFlag	Output	Reference to a Boolean flag.

Description:

The Boolean pointed to by `EqFlag` is set to `TRUE` if the tasks in the taskset `Taskset1` are identical to those in the taskset `Taskset2`. Otherwise it is set to `FALSE`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_TS_INVALID	Either taskset invalid.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
AssignTaskset()
ChainTaskset()
GetTasksetRef()
MergeTaskset()
RemoveTaskset()
TestSubTaskset()

3.72 TestSubTaskset()

Test whether or not tasks are present in a taskset.

Function declaration:

```
StatusType TestSubTaskset (
    TasksetType    SubTaskset,
    TasksetType    RefTaskset,
    BooleanRefType SubsetFlag)
```

Parameters:

Parameter	Input/Output	Description
SubTaskset	Input	The taskset being tested.
RefTaskset	Input	The reference taskset.
SubsetFlag	Output	Reference to a Boolean flag.

Description:

This API call can be used to test whether one taskset is a subset of another.

The Boolean pointed to by `SubsetFlag` is set to `TRUE` if the tasks in the taskset `SubTaskset` all exist in the taskset `RefTaskset`. Otherwise it is set to `FALSE`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_TS_INVALID	Either taskset invalid.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	
Shutdown	x
Pretask	
Posttask	
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

None.

See also:

ActivateTaskset()
AssignTaskset()
ChainTaskset()
GetTasksetRef()
MergeTaskset()
RemoveTaskset()
TestEquivalentTaskset()

3.73 Tick_<CounterID>()

Increments a counter by one tick.

Function declaration:

```
void Tick_<CounterID>()
```

Parameters:

None.

Description:

This call indicates to counter `CounterID` that one tick has elapsed.

For each attached alarm, if sufficient ticks have elapsed, the action(s) for that alarm are executed.

For an expiry point on an attached and running Schedule Table, if sufficient ticks have elapsed, the action(s) for that expiry point are executed.

The match value for the next expiry of the alarm is set if the alarm was started with a cycle time greater than zero.




The match value for the next expiry point on the attached and running schedule table (if any) is set.

Error codes:

`Tick_<CounterID>()` cannot directly raise any errors. However, the following errors can result from the activation of tasks when alarms expire. In this case, `ErrorHook()` is invoked.

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_LIMIT	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

Rescheduling after a call to `Tick_<CounterID>()` depends on the calling environment.

- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Hooks: Rescheduling will not take place until the hook terminates.
- For correct timing analysis, lower priority tasks must not tick counters that result in the expiry of alarms, leading to the activation of higher priority tasks

See also:

```

DeclareCounter()
GetCounterValue()
InitCounter()
IncrementCounter()
osAdvanceCounter_<CounterID>()

```

3.74 TickSchedule()

Advance the 'now' value of a ticked schedule, possibly activating tasks.

Function declaration:

```
StatusType TickSchedule(ScheduleType ScheduleID)
```

Static version: TickSchedule_ScheduleID()

Parameters:

Parameter	Input/Output	Description
ScheduleID	Input	Schedule name.

Description:




This call indicates to ticked schedule `ScheduleID` that one tick has elapsed. If sufficient ticks have elapsed, the next arrivalpoint is processed. This activates the tasks for that arrivalpoint.

If the final arrivalpoint is processed by this call, the ticked schedule is stopped. Typically the application continues to call `TickSchedule()` periodically.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_LIMIT	Too many activations of a task. As a result, all task activations resulting from this call are ignored.
Extended	E_OS_ID	ScheduleType ScheduleID not valid.
Extended	E_OS_SYS_S_MISMATCH	ScheduleID is an advanced schedule.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	✘
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	✘
Shutdown	✘
Pretask	✘
Posttask	✘
Error	✘
Overrun	✘

Portability:

RTA-OSEK

Notes:

Rescheduling after a call to `TickSchedule()` depends on the calling environment:

- Category 2 ISR: Rescheduling will not take place until the Category 2 ISR terminates.
- Non-Preemptive Task: Rescheduling will not take place until the non-preemptive task terminates or calls `Schedule()`.
- Preemptive Task: Rescheduling will take place immediately if the activated task is higher priority.
- Hooks: Rescheduling will not take place until the hook terminates.
- For correct timing analysis, lower priority tasks must not tick schedules that result in the activation of higher priority tasks.

See also:

```

AdvanceSchedule()
GetArrivalpointDelay()
GetArrivalpointNext()
GetArrivalpointTasksetRef()
GetScheduleNext()
GetScheduleStatus()
GetScheduleValue()
SetArrivalpointDelay()
SetScheduleNext()
StartSchedule()
StopSchedule()
TestArrivalpointWritable()

```

3.75 WaitEvent()

Wait for one or more events.

Function declaration:

```
StatusType WaitEvent(EventMaskType Mask)
```

Parameters:

Parameter	Input/Output	Description
Mask	Input	Mask of the events waited for.


Description:

This API call puts the calling task into the *waiting* state until one of the specified events is set. If one or more is already set, the task remains in the *running* state.

Error codes:

Build	Code	Description
Standard	E_OK	No error.
Extended	E_OS_ACCESS	Calling task is not an extended task.
Extended	E_OS_CALLEVEL	Call at interrupt level.
Extended	E_OS_RESOURCE	Calling task occupies resources.
Extended	E_OS_SYS_CONFIG_ERROR	Configuration error.

Calling environment:

Environment	Valid
Idle task	
Task	✓
Category 1 interrupts	✗
Category 2 interrupts	✗

Environment (Hooks)	Valid
Startup	✗
Shutdown	✗
Pretask	✗
Posttask	✗
Error	✗
Overrun	✗

Portability:

AUTOSAR
OSEK

Notes:

None.

See also:

`ClearEvent()`
`DeclareEvent()`
`GetEvent()`
`SetEvent()`

4 Constructional Elements

4.1 DeclareAccessor()

Declare an accessor.

Function declaration:

```
DeclareAccessor (AccessorID)
```

Parameters:

Parameter	Input/Output	Description
AccessorID	Input	Accessor to be declared.

Description:

`DeclareAccessor()` provides an external constant declaration of an accessor.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

It is not necessary to manually insert `DeclareAccessor()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

None.

4.2 DeclareAlarm()

Declare an alarm.

Function declaration:

```
DeclareAlarm(AlarmID)
```

Parameters:

Parameter	Input/Output	Description
AlarmID	Input	Alarm to be declared.

Description:

`DeclareAlarm()` provides an external constant declaration of an alarm.

Error codes:

None.

Calling environment:

None.

Portability:

AUTOSAR
OSEK

Notes:

It is not necessary to manually insert `DeclareAlarm()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
CancelAlarm()  
GetAlarm()  
GetAlarmBase()  
SetAbsAlarm()  
SetRelAlarm()
```

4.3 DeclareCounter()

Declare a counter.

Function declaration:

```
DeclareCounter (CounterID)
```

Parameters:

Parameter	Input/Output	Description
CounterID	Input	Counter to be declared.

Description:

`DeclareCounter ()` provides an external constant declaration of a counter.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

It is not necessary to manually insert `DeclareCounter ()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
GetCounterValue ()
IncrementCounter ()
InitCounter ()
osAdvanceCounter_<ConterID> ()
Tick_<CounterID> ()
```

4.4 DeclareEvent()

Declare an event.

Function declaration:

```
DeclareEvent (EventID)
```

Parameters:

Parameter	Input/Output	Description
EventID	Input	Event to be declared.

Description:

`DeclareEvent ()` provides an external constant declaration of an event.

Error codes:

None.

Calling environment:

None.

Portability:

AUTOSAR
OSEK

Notes:

It is not necessary to manually insert `DeclareEvent ()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
ClearEvent ()
GetEvent ()
SetEvent ()
WaitEvent ()
```

4.5 DeclareFlag()

Declare a flag.

Function declaration:

```
DeclareFlag(FlagID)
```

Parameters:

Parameter	Input/Output	Description
FlagID	Input	Flag to be declared.

Description:

`DeclareFlag()` provides an external constant declaration of a flag.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

It is not necessary to manually insert `DeclareFlag()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

`ReadFlag()`

`ResetFlag()`

4.6 DeclareISR()

Declare an ISR.

Function declaration:

```
DeclareISR(ISRID)
```

Parameters:

Parameter	Input/Output	Description
ISRID	Input	ISR to be declared.

Description:

`DeclareISR()` provides an external constant declaration of an ISR.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

It is not necessary to manually insert `DeclareISR()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

`DisableAllInterrupts()`

`EnableAllInterrupts()`

`ResumeAllInterrupts()`

`ResumeOSInterrupts()`

`SuspendAllInterrupts()`

`SuspendOSInterrupts()`

4.7 DeclareMessage()

Declare a message.

Function declaration:

```
DeclareMessage (MessageID)
```

Parameters:

Parameter	Input/Output	Description
MessageID	Input	Message to be declared.

Description:

`DeclareMessage()` provides an external constant declaration of a message.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

It is not necessary to manually insert `DeclareMessage()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
InitCOM()
MessageInit()
ReadFlag()
ReceiveMessage()
ResetFlag()
SendMessage()
StartCOM(),
StopCOM()
```


4.8 DeclareResource()

Declare a resource.

Function declaration:

```
DeclareResource (ResourceID)
```

Parameters:

Parameter	Input/Output	Description
ResourceID	Input	Resource to be declared.

Description:

`DeclareResource()` provides an external constant declaration of a resource.

Error codes:

None.

Calling environment:

None.

Portability:

AUTOSAR
OSEK

Notes:

It is not necessary to manually insert `DeclareResource()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
GetResource()  
ReleaseResource()
```

4.9 DeclareTask()

Declare a task.

Function declaration:

```
DeclareTask(TaskID)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task to be declared.

Description:

`DeclareTask()` provides an external constant declaration of a task.

Error codes:

None.

Calling environment:

None.

Portability:

AUTOSAR
OSEK

Notes:

It is not necessary to manually insert `DeclareTask()` into any files. RTA-OSEK will place it in the appropriate files.

Constructional elements cannot be called at run-time.

See also:

```
ActivateTask()  
ChainTask()  
GetTaskID()  
GetTaskState()  
TerminateTask()
```


5 Advanced Counter & Schedule Driver Interface

The application programmer must supply device driver functions to interface RTA-OSEK to the tick sources that drive each advanced schedule and each advanced counter.

Important: You must use exactly one set of hardware tick source device driver functions for each advanced schedule and for each advanced counter defined in your OIL configuration file. It is not possible to share tick source drivers between advanced schedules and advanced counters.

The driver functions for schedules take the following form:

- Set_<ScheduleID> ()
- State_<ScheduleID> ()
- Now_<ScheduleID> ()
- Cancel_<ScheduleID> ()

where <ScheduleID> is the name of the schedule.

Similarly, the driver functions for advanced alarms take the following form:

- Set_<CounterID> ()
- State_<CounterID> ()
- Now_<CounterID> ()
- Cancel_<CounterID> ()

where <CounterID> is the name of the advanced counter.

Portability Note: The OSEK OS and the AUTOSAR OS specifications do not provide a standardized API to access tick sources. The interface described here is therefore RTA-OSEK specific.

RTA-OSEK does not place any reentrancy requirements on the driver functions. They are always called in mutual exclusion (from OS level). However, if the functions are also called directly by the application, then care should be taken to ensure that they are still only called from OS level.

RTA-OSEK API calls must not be made from these driver functions.

5.1 Tick Source Semantics

Each advanced schedule and each advanced counter is assumed to have an underlying hardware tick source with the following semantics:

- There is a **now** value which increments each **tick**, wraps at a given modulus value and is free running.
- There is a **match** value.
- When **now** becomes equal to **match**, a tick source expiry is said to have occurred. The tick source can be set up to interrupt on an expiry.

- The tick source can also be set up not to interrupt. (Referred to as canceling the expiry.)

Typically, a hardware implementation (hardware counter and timer output compare channel) can be used to implement the underlying tick source for each advanced schedule or advanced counter as required.

5.2 Initialization

It is assumed that each advanced schedule/counter has been initialized by the application before any OS calls are made:

- The modulus of the schedule/counter driver tick source must correspond to the Max Value configured for the schedule/counter (as specified in the OIL configuration file).
- The tick source must be set up such that each tick of the tick source corresponds to the duration of 1 tick for the associated schedule's/counter's tick rate.
- The tick source is free running (incrementing).
- The tick source is configured to not interrupt.

5.3 Cancel_<ScheduleID>() [User Provided]

Driver callback routine to cancel any outstanding tick source expiry.

Function declaration:

```
OS_CALLBACK(void) Cancel_<ScheduleID>(void)
```

Parameters:

None.

Description:

The function `Cancel_<ScheduleID>()` must cancel any outstanding tick source expiry. It must set up the hardware to not interrupt and then clear any pending interrupts.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

This function is not required to stop the associated hardware from incrementing.

See also:

```
Now_<ScheduleID>()  
Set_<ScheduleID>()  
State_<ScheduleID>()
```

5.4 `Now_<ScheduleID>()` [User Provided]

Driver callback routine that returns the current tick source value.

Function declaration:

```
OS_CALLBACK(TickType) Now_<ScheduleID>(void)
```

Parameters:

This call returns a `TickType` value.

Description:

The function `Now_<ScheduleID>()` must return the **now** value of the underlying tick source.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

None.

See also:

```
Cancel_<ScheduleID>()  
Set_<ScheduleID>()  
State_<ScheduleID>()
```

5.5 Set_<ScheduleID>() [User Provided]

Driver callback routine to set the next match value for an advanced schedule.

Function declaration:

```
OS_CALLBACK(void) Set_<ScheduleID>(TickType Match)
```

Parameters:

Parameter	Input/Output	Description
Match	Input	Next absolute match value.

Description:

The function `Set_<ScheduleID>()` takes the `Match` value as a parameter. It must clear any pending interrupt and set up the hardware to interrupt when **now** reaches the new `Match` value. `Match` is, therefore, an absolute value at which the schedule will process the next arrivalpoint.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK.

Notes:

`Set_<ScheduleID>()` is not required to initialize hardware. This should be done by initialization code before RTA-OSEK is started.

See also:

`Cancel_<ScheduleID>()`

`Now_<ScheduleID>()`

`State_<ScheduleID>()`

5.6 State_<ScheduleID>() [User Provided]

Driver callback routine to get the state of the underlying tick source hardware.

Function declaration:

```
OS_CALLBACK(void)
State_<ScheduleID>(ScheduleStatusRefType State)
```

Parameters:

Parameter	Input/Output	Description
State	Output	Status of the underlying tick source.

Description:

This function must update a schedule status structure. It will only be called when the schedule is *running*.

For more information on `ScheduleStatusType` see Section 2.1.1.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

If **match** is equal to **now**, then the value of the `expiry` field is set to zero. Zero indicates that it is the full schedule modulus number of ticks before the next counter event.

See also:

```
Cancel_<ScheduleID>()
Now_<ScheduleID>()
Set_<ScheduleID>()
```

5.7 Cancel_<CounterID> [User Provided]

Driver callback routine to cancel any outstanding tick source expiry.

Function declaration:

```
OS_CALLBACK(void) Cancel_<CounterID>(void)
```

Parameters:

None.

Description:

The function `Cancel_<CounterID>()` must cancel any outstanding tick source expiry. It must set up the hardware to not interrupt and then clear any pending interrupts.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

This function is not required to stop the associated hardware from incrementing.

See also:

```
Now_<CounterID>()  
Set_<CounterID>()  
State_<CounterID>()
```

5.8 `Now_<CounterID>()` [User Provided]

Driver callback routine that returns the current tick source value.

Function declaration:

```
OS_CALLBACK(TickType) Now_<CounterID>(void)
```

Parameters:

This call returns a `TickType` value.

Description:

The schedule function `Now_<CounterID>()` must return the **now** value of the tick source.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

None.

See also:

```
Cancel_<CounterID>()  
Set_<CounterID>()  
State_<CounterID>()
```

5.9 Set_<CounterID>() [User Provided]

Driver callback routine to set the next match value on an advanced counter.

Function declaration:

```
OS_CALLBACK(void) Set_<CounterID>(TickType Match)
```

Parameters:

Parameter	Input/Output	Description
Match	Input	Next absolute match value.

Description:

The function `Set_<CounterID>()` takes the `Match` value as a parameter. It must set up the hardware to interrupt when **now** reaches the new `Match` value. `Match` is, therefore, an absolute value at which the next alarm and/or schedule table expiry point needs to be processed by RTA-OSEK.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK.

Notes:

`Set_<CounterID>()` is not required to initialize hardware. This should be done by initialization code before RTA-OSEK is started.

See also:

`Cancel_<CounterID>()`

`Now_<CounterID>()`

`State_<CounterID>()`

5.10 State_<CounterID>() [User Provided]

Driver callback routine to get the state of the underlying tick source hardware.

Function declaration:

```
OS_CALLBACK(void)
State_<CounterID>(ScheduleStatusRefType State)
```

Parameters:

Parameter	Input/Output	Description
State	Output	Status of the underlying counter.

Description:

This function must update a schedule status structure. It will only be called when the schedule is *running*.

For more information on `ScheduleStatusType` see Section 2.1.1.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK.

Notes:

If 'match' is equal to 'now', then the value of the `expiry` field will be set to zero, indicating that it is the full schedule modulus number of ticks before the next counter event.

See also:

```
Cancel_<CounterID>()
Now_<CounterID>()
Set_<CounterID>()
```


6 Execution Time Monitoring

Portability Note: All of the execution time monitoring API calls are RTA-OSEK specific.

6.1 GetExecutionTime()

Get the consumed execution time for the current invocation of the calling task or Category 2 ISR.

Function declaration:

```
StopwatchTickType GetExecutionTime(void)
```

Parameters:

None.

Description:

Returns the execution time consumed, since the start of execution, by the current invocation of the calling basic task or Category 2 ISR.




In the case of an extended task, it returns execution time consumed by the current invocation of the calling task or since the start of execution or the previous `WaitEvent()` call.

If the value overflows, then the returned value will be the wrapped value.

Error codes:

None.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	x
Overrun	x

Portability:

RTA-OSEK

Notes:

This API call requires that the user provides access to a free running counter through the `GetStopwatch()` callback routine.

This API call always returns zero when the OS Status is set to Standard in the OIL configuration file.

See also:

`GetLargestExecutionTime()`

`GetStopwatch()`

`GetStopwatchUncertainty()`

`ResetLargestExecutionTime()`

6.2 GetLargestExecutionTime()

Get the largest observed execution time for the specified task.

Function declaration:

```
StatusType GetLargestExecutionTime(
    TaskType          TaskID,
    StopwatchTickRefType WCET)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task name.
WCET	Output	Worst case execution time.




Description:



Returns the largest execution time at completion for the task or Category 2 ISR identified by `TaskID`. This maximum value is over all complete invocations of the task or Category 2 ISR that have completed since the previous call to `ResetLargestExecutionTime()` for that task or Category 2 ISR or to `StartOS()`.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_T_INVALID	Invalid task or Category 2 ISR handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	x
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	x
Shutdown	x
Pretask	x
Posttask	x
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

If no invocations of the specified task have been completed the API call returns zero.

The call always returns zero when the OS Status is set to Standard in the OIL configuration file.

See also:

`GetExecutionTime()`

`GetStopwatch()`

`GetStopwatchUncertainty()`

`ResetLargestExecutionTime()`

6.3 GetStopwatch() [User Provided]

RTA-OSEK callback routine to return the current value of a free-running counter.

Function declaration:

```
OS_NONREENTRANT (StopwatchTickType) GetStopwatch(void)
```

Parameters:

The return value is a `StopwatchTickType`.

Description

`GetStopwatch()` must return the current value of a free-running timer (which increments and overflows at the end of its range). This timer provides the timebase for execution time measurements. The execution time API calls return values in units of this timebase.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

The user must provide a definition for `GetStopwatch()` if the OS Status is set to Timing or Extended in the OIL Configuration file.

See also:

```
GetExecutionTime()  
GetLargestExecutionTime()  
GetStopwatchUncertainty()  
ResetLargestExecutionTime()
```

6.4 GetStopwatchUncertainty() [User Provided]

RTA-OSEK callback routine to provide `StopwatchUncertainty()` for execution time monitoring.

Function declaration:

```
OS_NONREENTRANT (StopwatchTickType)
GetStopwatchUncertainty(void)
```

Parameters:

Returns a `StopwatchTickType` indication of uncertainty.

Description:

On some targets, it is not possible to have a stopwatch that runs at the same rate as CPU cycles. This introduces an uncertainty into timing measurements.

All timing measurements are based on the calculation of the elapsed time between two points in the code. Where the `GetStopwatch()` clock is slower than the CPU clock, simply subtracting two times can underestimate the elapsed time by a given amount. This is the uncertainty in the elapsed time measurement. `GetStopwatchUncertainty()` returns the measurement uncertainty that applies to calls to `GetStopwatch()`.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

The user must provide a definition for `GetStopwatch()` if the OS Status is set to Timing or Extended in the OIL Configuration file.

See also:

```
GetExecutionTime()
GetLargestExecutionTime()
GetStopwatch()
ResetLargestExecutionTime()
```

6.5 ResetLargestExecutionTime()

Resets the current largest execution time.

Function declaration:

```
StatusType ResetLargestExecutionTime(TaskType TaskID)
```

Parameters:

Parameter	Input/Output	Description
TaskID	Input	Task name.




Description:



It sets the recorded maximum execution time of the task or Category 2 ISR supplied to zero.

Error codes:

Build	Code	Description
All	E_OK	No error.
Extended	E_OS_SYS_T_INVALID	Invalid task or Category 2 ISR handle.

Calling environment:

Environment	Valid
Idle task	
Task	
Category 1 interrupts	×
Category 2 interrupts	

Environment (Hooks)	Valid
Startup	×
Shutdown	×
Pretask	×
Posttask	×
Error	
Overrun	

Portability:

RTA-OSEK

Notes:

This API call has no effect when the OS Status is set to Standard in the OIL configuration file.

See also:

```
GetExecutionTime()
GetLargestExecutionTime()
GetStopwatch()
GetStopwatchUncertainty()
```


7 Hook Routines

When the system is configured to use the hook routines you must provide the callback routines that RTA-OSEK will call.

7.1 ErrorHook() [User Provided]

Hook routine used for trapping errors resulting from incorrect use of the RTA-OSEK API.

Function declaration:

```
OS_HOOK(void) ErrorHook(StatusType Error)
```

Parameters:

Parameter	Input/Output	Description
Error	Input	The error that occurred.

Description:

This is called by RTA-OSEK when an API call returns a `StatusType` not equal to `E_OK`. It is called before returning to the user level. RTA-OSEK passes the `StatusType` into the `ErrorHook()`.

This hook is not called if an API call, called from `ErrorHook()`, does not return `E_OK` as status value.

Macros are provided for obtaining information within `ErrorHook()`. The macros can be used to find out information about the API call that called `ErrorHook()` and the parameters that were passed to it.

The macro `OSErrorGetServiceID` returns an `OSServiceIDType` with values `OSServiceId_XXX` where `XXX` is the name of an API call. This is used to determine which API call resulted in the error.

The `OSError_APICallName_Parameter()` macros are available for each API call. These are used to determine which parameters were passed to the API call.

These macros are only available in the `ErrorHook()` and RTA-OSEK must be configured correctly in order to use them.

Error codes:

None.

Calling environment:

None.

Portability:

OSEK

Notes:

None.

See also:

`OverrunHook()`

7.2 OverrunHook() [User Provided]

Hook routine for trapping execution budget overruns.

Function declaration:

```
OS_HOOK(void) OverrunHook(void)
```

Parameters:

None.

Description:

This hook routine is called in the Timing and Extended builds when specified task execution budgets have been exceeded.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

The user must provide a definition for `OverrunHook()` if the OS Status is set to Timing or Extended in the OIL configuration file, irrespective of whether execution time monitoring is being performed.

See also:

```
GetExecutionTime()  
GetLargestExecutionTime()  
GetStopwatch()  
GetStopwatchUncertainty()
```

7.3 MessageInit() [User Provided]

Hook routine to initialize user message data structures.

Function declaration:

```
StatusType MessageInit(void)
```

Parameters:

None.

Description:

This routine initializes all application specific message objects.

Error codes:

Build	Code	Description
All	E_OK	No error.
		Return an implementation or application specific error code if the initialization did not complete successfully.

Calling environment:

None.

Portability:

OSEK

Notes:

`MessageInit()` function has to be provided by the application programmer and can only be called by the `StartCOM()` routine.

The `StatusType` returned by `MessageInit()` is in turn returned by `StartCOM()`. RTA-OSEK provides a default implementation of `MessageInit()` that returns `E_OK`, unless this is replaced by a user provided version.

See also:

`DeclareMessage()`

`ClearEvent()`

7.4 PostTaskHook() [User Provided]

Hook routine called when a task enters the *ready* or *suspended* state.

Function declaration:

```
OS_HOOK(void) PostTaskHook(void)
```

Parameters:

None.

Description:

This hook routine is called by the operating system after executing the current task, but before leaving the task's *running* state (to allow evaluation of the TaskID).

Error codes:

None.

Calling environment:

None.

Portability:

OSEK.

Notes:

This routine is called for each preemption of a task.

See also:

PreTaskHook ()

7.5 PreTaskHook() [User Provided]

Hook routine called when a task enters the *running* state.

Function declaration:

```
OS_HOOK(void) PreTaskHook(void)
```

Parameters:

None.

Description:

This hook routine is called by the operating system before executing a new task, but after the transition of the task to the *running* state (to allow evaluation of the TaskID).

Error codes:

None.

Calling environment:

None.

Portability:

OSEK

Notes:

This routine is called for each preemption of a task.

See also:

PostTaskHook()

7.6 ShutdownHook() [User Provided]

Hook routine called during operating system shutdown.

Function declaration:

```
OS_HOOK(void) ShutdownHook(StatusType Error)
```

Parameters:

Parameter	Input/Output	Description
Error	Input	Error occurred.

Description:

If a `ShutdownHook()` is configured, this hook routine is called by the operating system when the OS API call `ShutdownOS()` has been called. This routine is called during the operating system shutdown.

Error codes:

None.

Calling environment:

None.

Portability:

OSEK

Notes:

The possible error parameter values arising from internal calls of `ShutdownOS()` are:

- `E_OS_SYS_STACK_FAULT`: The application returned from `StackFaultHook`.
- `E_OS_SYS_CALLEVEL`: The application used illegal OS services above OS level.

See also:

`ShutdownOS()`

7.7 StackFaultHook() [User Provided]

Called if the stack exceeds configured stack size for one or more tasks.

Function declaration:

```
OS_HOOK(void) StackFaultHook(SmallType StackID,
                              SmallType StackError, UIntType Overflow)
```

Parameters:

Parameter	Input/Output	Description
StackID	Input	Shows the stack to which the data applies.
StackError	Input	(See Description below.)
Overflow	Input	Number of bytes on the stack beyond the limits for which the system has been configured.

Description:

`StackFaultHook()` must be provided when extended tasks are present. It is called if the stack exceeds stack size that has been configured for one or more tasks.

`StackFaultHook()` is usually called because of a configuration error, where the stack usage of one of more tasks has been incorrectly described. It may, however, be triggered as a result of an application defect, such as run-away stack.

`StackError` is one of:

- `OS_EXTENDED_TASK_STARTING`: Extended task is being dispatched.
- `OS_EXTENDED_TASK_RESUMING`: Extended task is being resumed after calling `WaitEvent()`.
- `OS_EXTENDED_TASK_WAITING`: Extended task is calling `WaitEvent()`.

Error codes:

None.

Calling environment:

None.

Portability:

RTA-OSEK

Notes:

`StackID` is always zero, unless otherwise defined in the target specific *RTA-OSEK Binding Manual*.

`GetTaskID()` and `GetActiveApplicationMode()` are the only API calls that should be made from this hook routine.

See also:

None.

7.8 StartupHook() [User Provided]

Hook routine called after OS initialization and before the scheduler is running.

Function declaration:

```
OS_HOOK(void) StartupHook(void)
```

Parameters:

None.

Description:

If a `StartupHook()` is configured, this hook routine is called by the operating system at the end of the operating system initialization and before the scheduler is running. At this time the application can start tasks, initialize device drivers and so on.

Error codes:

None.

Calling environment:

None.

Portability:

OSEK

Notes:

None.

See also:

`StartOS()`

8 Callbacks

8.1 ALARMCALLBACK() [User Provided]

Called when an alarm expires and when it has a callback configured.

Function declaration:

```
void ALARMCALLBACK(AlarmcallbackRoutineName) (void)
```

Parameters:

None.

Description:

This is called by RTA-OSEK when an alarm expires and when that alarm has a callback configured.

Error codes:

None.

Calling environment:

None.

Portability:

OSEK.

Notes:

The only RTA-OSEK API calls that can be made are `ResumeAllInterrupts()` and `SuspendAllInterrupts()`.

See also:

```
osAdvanceCounter_<CounterID>  
Tick_<CounterID>()
```

8.2 COMCALLBACK() [User Provided]

Called when a message is received and when it has a callback configured.

Function declaration:

```
COMCALLBACK (COMcallbackRoutineName) (void)
```

Parameters:

None.

Description:

This is called by RTA-OSEK when a message is received and when that message has a callback configured.

Error codes:

None.

Calling environment:

None.

Portability:

OSEK.

Notes:

The only API calls that can be made are `SendMessage()`, `ReceiveMessage()`, `GetMessageStatus()`.

See also:

None.

9 Predefined Objects

As well as objects explicitly declared in the configuration file, RTA-OSEK also generates the symbols, objects and handles defined in the following sections for use in your code.

9.1 OSEK Counter Attributes

Object	Description	OSEK
OS_CYCLES_PER_x	Number of CPU cycles per tick of counter x.	
OS_CYCLES_PER_SWTICK	Number of CPU cycles per tick of the system counter.	
OSMAXALLOWEDVALUE	Maximum possible allowed value of the system counter in ticks.	✓
OSMAXALLOWEDVALUE_x	Maximum possible allowed value of counter x in ticks.	✓
OSMINCYCLE	Minimum allowed number of ticks for a cyclic alarm of the system counter.	✓
OSMINCYCLE_x	Minimum allowed number of ticks for a cyclic alarm of counter x.	✓
OS_NS_PER_CYCLE	Duration of a CPU cycle in nanoseconds.	
OSTICKDURATION	Duration of a tick of the system counter in nanoseconds.	✓
OSTICKDURATION_x	Duration of a tick of the counter x in nanoseconds.	
OSTICKSPERBASE	Number of ticks required to reach a specific unit of the system counter.	✓
OSTICKSPERBASE_x	Number of ticks required to reach a specific unit of counter x.	✓

9.2 OSEK Task States

Object	Description
READY	Constant of data type <code>TaskStateType</code> for task state <i>ready</i> .
RUNNING	Constant of data type <code>TaskStateType</code> for task state <i>running</i> .
SUSPENDED	Constant of data type <code>TaskStateType</code> for task state <i>suspended</i> .
WAITING	Constant of data type <code>TaskStateType</code> for task state <i>waiting</i> . Not used for BCC tasks.

9.3 OSEK Resources

Object	Description
RES_SCHEDULER	Constant of data type <code>ResourceType</code> . Shared by all tasks.

9.4 OSEK Application Modes

Object	Description
OSDEFAULTAPPMODE	Default application mode. Always a valid parameter to <code>StartOS()</code> .

9.5 RTA-OSEK Build Levels

Object	Description
OS_EXTENDED_BUILD	Macro emitted by the RTA-OSEK configuration tool in the Extended build.
OS_STANDARD_BUILD	Macro emitted by the RTA-OSEK configuration tool in Standard build.
OS_TIMING_BUILD	Macro emitted by the RTA-OSEK configuration tool in the Timing build.
OS_ET_MEASURE	Macro emitted by the RTA-OSEK configuration tool in both the Timing and Extended builds.

9.6 RTA-OSEK Tasksets

Object	Description
<code>os_all_tasks</code>	<code>os_all_tasks</code> is a read-only taskset containing all defined tasks, including the idle task.
<code>os_no_tasks</code>	<code>os_no_tasks</code> is a read-only taskset containing no tasks (including the idle task).
<code>os_ready_tasks</code>	<code>os_ready_tasks</code> is a read-only taskset, but its contents are changed by RTA-OSEK so that it identifies all the tasks in the <i>ready</i> state and the <i>running</i> task.
<code>osek_cc2_tasks</code>	<code>osek_cc2_tasks</code> is a taskset containing all the BCC2 and ECC2 tasks. It is defined only if there are BCC2 or ECC2 tasks or in the Extended build.
<code>osek_ecc_tasks</code>	<code>osek_ecc_tasks</code> identifies all the ECC1 and ECC2 tasks. It is defined only if there are ECC1 or ECC2 tasks or in the Extended build.

9.7 RTA-OSEK Application Characteristics

Amongst other characteristics, `osekcomn.h` defines the following symbols which characterize aspects of the application:

Symbol	Characteristic
OSEK_BCC1	Contains only BCC1 tasks.
OSEK_BCC2	Contains BCC2 tasks.
OSEK_BCC2C	Contains BCC2 tasks with unique priorities.
OSEK_BCC2F	Contains BCC2 tasks with shared priorities.
OSEK_ECC1	Contains ECC1 tasks.
OSEK_ECC2	Contains ECC2 tasks.
OSEK_ECC2C	Contains BCC2 tasks with unique priorities and ECC1 tasks.
OSEK_ECC2F	Contains ECC2 tasks with shared and/or both BCC2 tasks with shared priorities and ECC tasks.
OS_STC_COMPATIBLE	STC compliant (the system is constructed in such a way that RTA-OSEK Planner can analyze the timing behavior. See the <i>RTA-OSEK User Guide</i> for further details).
OS_NO_SCHEDULECALL	The application does not call <code>Schedule()</code> .

The "F" and "C" Characteristics

If your application contains tasks with queued activations that share priorities with other tasks typically stores activations in a FIFO queue. In this case `OSEK_BCC2F` or `OSEK_ECC2F` is defined.

When your application does not use shared priorities, RTA-OSEK uses an optimized implementation called 'counted activation'. In this case, the `OSEK_BCC2C` or `OSEK_ECC2C` is defined.

10 Macro Definitions

Macro	Description
ALARMCALLBACK()	Format of the callback routine called when the alarm expires.
ISR()	Format that the Application Category 2 ISRs must be written in. <pre>ISR(IsrID) { ... }</pre>
OSError_APICallName_Parameter()	Available for each API call. Used to determine which parameters were passed to the call.
OSErrorGetServiceID	Determines which API call resulted in the error. Returns an <code>OSServiceIDType</code> with values <code>OSServiceId_xxx</code> where <code>xxx</code> is the name of an API call.
OS_ATOMIC(expr)	Evaluate C expression <code>expr</code> with all interrupts temporarily disabled.
OS_CALLBACK()	Marks user functions that are called directly from RTA-OSEK. Function declaration: <pre>OS_CALLBACK(void) Cancel_<ScheduleID>(void);</pre>
OS_HOOK()	Marks hook routines that are supported by the OS. <pre>OS_HOOK(void) ErrorHook(StatusType Error)</pre>
OS_MAIN()	Marks main functions in a portable manner. Function declaration: <code>OS_MAIN()</code>
OS_NONREENTRANT()	Marks user functions that will not be reentered. Function declaration: <pre>OS_NONREENTRANT(void) GetStopwatch(void);</pre>
OS_STATUS_RUNNING	Indicates that the schedule is <i>running</i> . <pre>#define OS_STATUS_RUNNING (SmallType (2))</pre>
OS_STATUS_PENDING	Indicates that a schedule is pending (it is due to process an arrivalpoint). <pre>#define OS_STATUS_PENDING (SmallType (1))</pre>

Macro	Description
TASK()	Format that application tasks must be written in. TASK(TaskID) { ... }

11 Quick Reference Guide

11.1 Dynamic Interface RTA-OSEK API Calls

Name	Description
ActivateTask()	Activates a task.
ActivateTaskset()	Activate a set of tasks.
AdvanceSchedule()	Process the next arrivalpoint on the advanced schedule.
AssignTaskset()	Assign the members of a taskset to another taskset.
CancelAlarm()	Cancel an alarm.
ChainTask()	Terminate the calling task and activate a task.
ChainTaskset()	Terminate the calling task and activate a set of tasks.
ClearEvent()	Clear the calling task's events.
CloseCOM()	Release low-level hardware resources needed for COM communication.
DisableAllInterrupts()	Disables all interrupts.
EnableAllInterrupts()	Ends a critical section started by DisableAllInterrupts().
GetActiveApplicationMode()	Get the current application mode.
GetAlarm()	Returns the number of ticks before the alarm next expires.
GetAlarmBase()	Get the alarm base characteristics.
GetArrivalpointDelay()	Get the delay between one arrivalpoint and the successor.
GetArrivalpointNext()	Get the successor of an arrivalpoint.
GetArrivalpointTasksetRef()	Get a reference to an arrivalpoint's taskset.
GetCounterValue()	Get the counter value.

Name	Description
GetEvent ()	Get the events for the specified task.
GetISRID ()	Get the handle of the current ISR.
GetMessageResource ()	Get the message resource.
GetMessageStatus ()	Return the message status.
GetResource ()	Get a resource.
GetScheduleNext ()	Get the next arrivalpoint to be processed by a schedule.
GetScheduleStatus ()	Get the current status of a schedule.
GetScheduleTableStatus ()	Get the status of a schedule table.
GetScheduleValue ()	Get the 'now' property of a schedule.
GetStackOffset ()	Gets the current stack pointer.
GetTaskID ()	Get the status of the specified task.
GetTasksetRef ()	Create a singleton taskset from the specified task.
GetTaskState ()	Get the task state.
IncrementCounter ()	Initializes the low-level resources necessary for COM.
InitCOM ()	Increments a counter by one tick
InitCounter ()	Initialize the counter to a given tick value.
MergeTaskset ()	Merge two tasksets.
NextScheduleTable ()	Stop a schedule table and start another.
osAdvanceCounter_<CounterID> ()	Process the next alarm/schedule table expiry point due on the counter.
osResetOS ()	Reset the OS variables before restarting.
ReadFlag ()	Read the status of the message flag associated with an OSEK COM message.

Name	Description
ReceiveMessage()	Receive the specified OSEK COM message.
ReleaseMessageResource()	Release a previously held message resource.
ReleaseResource()	Release a previously held resource.
RemoveTaskset()	Remove tasks from a taskset.
ResetFlag()	Reset the flag associated with an OSEK COM message.
ResumeAllInterrupts()	Resumes processing of interrupts.
ResumeOSInterrupts()	Restore interrupt processing of Category 2 interrupts.
Schedule()	Provides a rescheduling point for non-preemptive tasks.
SendMessage()	Sends the specified data as an OSEK COM message.
SetAbsAlarm()	Set the value of the counter at which the alarm expires.
SetArrivalpointDelay()	Set the delay between one arrivalpoint and the successor.
SetArrivalpointNext()	Reconfigure a schedule by changing the successor of an arrivalpoint.
SetEvent()	Set events for a specified task.
SetRelAlarm()	Set how many counter ticks occur before the alarm expires.
SetScheduleNext()	Indicate the next arrivalpoint to the schedule.
ShutdownOS()	Perform operating system shutdown.
StartCOM()	Start the COM service.
StartOS()	Start the OS.
StartSchedule()	Start a stopped schedule.
StartScheduleTable()	Start a schedule table.

Name	Description
StopCOM()	Stops the COM service.
StopSchedule()	Stop a running schedule.
StopScheduleTable()	Stop a schedule table.
SuspendAllInterrupts()	Disables interrupt processing of all interrupts.
SuspendOSInterrupts()	Disables interrupt processing of Category 2 interrupts.
TerminateTask()	Terminates the calling task.
TestArrivalpointWritable()	Test whether or not an arrivalpoint is writable.
TestEquivalentTaskset()	Test whether or not tasksets contain the same tasks.
TestSubTaskset()	Test whether or not tasks are present in a taskset.
Tick_<CounterID>()	Increments a counter by one tick.
TickSchedule()	Advance the 'now' value of a ticked schedule, possibly activating tasks.
WaitEvent()	Wait for one or more events.

11.2 Static Interface RTA-OSEK API Calls

Name	Description
ActivateTask_<TaskID>()	Activates a task.
ActivateTaskset_<TasksetID>()	Activate a set of tasks.
AdvanceSchedule_<ScheduleID>()	Process the next arrivalpoint on the advanced schedule.
ChainTask_<TaskID>()	Terminate the calling task and activate a task.
ChainTaskset_<TasksetID>()	Terminate the calling task and activate a set of tasks.
GetResource_<ResID>()	Get a resource.

Name	Description
ReleaseResource_<ResID>()	Release a previously held resource.
SendMessage_<Message>_<Data>()	Sends the specified data as an OSEK COM message.
SetEvent_<TaskID>_<Mask>()	Set events for a specified task.
TickSchedule_<ScheduleID>()	Advance the 'now' value of a ticked schedule, possibly activating tasks.

11.3 Constructional Elements

Name	Description
DeclareAccessor()	Declare an accessor.
DeclareAlarm()	Declare an alarm.
DeclareCounter()	Declare a counter.
DeclareEvent()	Declare an event.
DeclareFlag()	Declare a flag.
DeclareISR()	Declare an ISR.
DeclareMessage()	Declare a message.
DeclareProcess()	Declare a process.
DeclareResource()	Declare a resource.
DeclareTask()	Declare a task.

11.4 Advanced Schedule Driver Interface

Name	Description
Cancel_<ScheduleID>()	Driver callback routine to cancel any outstanding counter expiry.
Now_<ScheduleID>()	Driver callback routine that returns the current counter value.

Name	Description
Set_<ScheduleID> ()	Driver callback routine to set the next match value for the advanced schedule driver.
State_<ScheduleID> ()	Driver callback routine to get the state of the underlying counter/compare hardware.

11.5 Advanced Counter Driver Interface

Name	Description
Cancel_<CounterID> ()	Driver callback routine to cancel any outstanding counter expiry.
Now_<CounterID> ()	Driver callback routine that returns the current counter value.
Set_<CounterID> ()	Driver callback routine to set the next match value for the advanced counter driver.
State_<CounterID> ()	Driver callback routine to get the state of the underlying counter/compare hardware.

11.6 Execution Time Monitoring Interface

Name	Description
GetExecutionTime()	Get the consumed execution time for the calling task or Category 2 ISR.
GetLargestExecutionTime()	Get the largest observed execution time for the specified task.
GetStopwatch()	RTA-OSEK callback routine to return the current value of a free-running counter.
GetStopwatchUncertainty()	Callback routine to provide <code>StopwatchUncertainty()</code> for execution time monitoring.
ResetLargestExecutionTime()	Resets the current largest execution time.

11.7 Hooks

Name	Description
ErrorHook()	Hook routine used for trapping errors resulting from incorrect use of the RTA-OSEK API.
OverrunHook()	Hook routine for trapping execution budget overruns.
MessageInit()	Hook routine to initialize user message data structures.
PostTaskHook()	Hook routine used when a task enters the <i>ready</i> or <i>suspended</i> state.
PreTaskHook()	Hook routine used when a task enters the <i>running</i> state.
ShutdownHook()	Hook routine called during operating system shutdown.
StackFaultHook()	Called if the stack exceeds configured stack size for one or more tasks.
StartupHook()	Hook routine called after OS initialization and before the scheduler is running.

11.8 Other Callbacks

Name	Description
------	-------------

Name	Description
ALARMCALLBACK(AlarmcallbackRoutineName)	Called when an alarm expires and when it has a callback configured.
COMCALLBACK(COMcallbackRoutineNam	Called when a message is received and when it has a callback configured.

11.9 Predefined Objects

As well as objects explicitly declared in the configuration file, RTA-OSEK also generates the following symbols, objects and handles.

Object	Description
os_all_tasks	os_all_tasks is a read-only taskset containing all defined tasks, including the idle task.
OS_EXTENDED_BUILD	Macro emitted by the RTA-OSEK configuration tool in the Extended build.
os_no_tasks	os_no_tasks is a read-only taskset containing no tasks (including the idle task).
os_ready_tasks	os_ready_tasks is a read-only taskset, but its contents are changed by RTA-OSEK so that it identifies all the tasks in the <i>ready</i> state and the <i>running</i> task
OS_STANDARD_BUILD	Macro emitted by the RTA-OSEK configuration tool in Standard build.
OS_TIMING_BUILD	Macro emitted by the RTA-OSEK configuration tool in the Timing build.
OSDEFAULTAPPMODE	Default application mode. Always a valid parameter to StartOS().
osek_cc2_tasks	osek_cc2_tasks is a taskset containing all the BCC2 and ECC2 tasks. It is defined only if there are BCC2 or ECC2 tasks or in the Extended build.
osek_ecc_tasks	osek_ecc_tasks identifies all the ECC1 and ECC2 tasks. It is defined only if there are ECC1 or ECC2 tasks or in the Extended build.

Object	Description
OSMAXALLOWEDVALUE	Maximum possible allowed value of the system counter in ticks.
OSMAXALLOWEDVALUE_x	Maximum possible allowed value of counter x in ticks.
OSMINCYCLE	Minimum allowed number of ticks for a cyclic alarm of the system counter.
OSMINCYCLE_x	Minimum allowed number of ticks for a cyclic alarm of counter x .
OSTICKDURATION	Duration of a tick of the system counter in nanoseconds.
OSTICKSPERBASE	Number of ticks required to reach a specific unit of the system counter.
OSTICKSPERBASE_x	Number of ticks required to reach a specific unit of counter x .
READY	Constant of data type <code>TaskStateType</code> for task state <i>ready</i> .
RES_SCHEDULER	Constant of data type <code>ResourceType</code> . Shared by all tasks.
RUNNING	Constant of data type <code>TaskStateType</code> for task state <i>running</i> .
SUSPENDED	Constant of data type <code>TaskStateType</code> for task state <i>suspended</i> .
WAITING	Constant of data type <code>TaskStateType</code> for task state <i>waiting</i> . Not used for BCC tasks.

11.10 Macro Definitions

Macro	Description
ALARMCALLBACK()	Format of the callback routine called when the alarm expires.

Macro	Description
ISR()	Format that the Application Category 2 ISRs must be written in. <pre>ISR(IsrID) { ... }</pre>
OSError_APICallName_Parameter()	Available for each API call. Used to determine which parameters were passed to the call.
OSErrorGetServiceID	Determines which API call resulted in the error. Returns an <code>OSServiceIDType</code> with values <code>OSServiceId_xxx</code> where <code>xxx</code> is the name of an API call.
OS_ATOMIC(expr)	Evaluate C expression <code>expr</code> with all interrupts temporarily disabled.
OS_CALLBACK()	Marks user functions that are called directly from RTA-OSEK. <code>Funcnt006Fn</code> declaration: <pre>OS_CALLBACK(void) Cancel_<ScheduleID>(void);</pre>
OS_HOOK()	Marks hook routines that are supported by the OS. <pre>OS_HOOK(void) ErrorHook(StatusType Error)</pre>
OS_MAIN()	Marks main functions in a portable manner. Function declaration: <code>OS_MAIN()</code>
OS_NONREENTRANT()	Marks user functions that will not be reentered. Function declaration: <pre>OS_NONREENTRANT(void) GetStopwatch(void);</pre>
OS_STATUS_RUNNING	Indicates that the schedule is <i>running</i> . <pre>#define OS_STATUS_RUNNING (SmallType (2))</pre>

Macro	Description
OS_STATUS_PENDING	Indicates that a schedule is pending (it is due to process an arrivalpoint). #define OS_STATUS_PENDING (SmallType (1))
TASK()	Format that application tasks must be written in. TASK(TaskID) { ... }

11.11 Error Codes

Error Code	Description
E_COM_BUSY	Message in use by application task/function.
E_COM_ID	Invalid message name passed as parameter.
E_COM_LIMIT	Overflow of FIFO associated with queued messages.
E_COM_LOCKED	Rejected service call, message object is held.
E_COM_NOMSG	No message available.
E_COM_SYS_STOPPED	COM not started.
E_OK	No error.
E_OS_ACCESS	Resource is already held, or assigned priority of calling task or Category 2 ISR is higher than the calculated ceiling priority.
E_OS_CALLEVEL	This API cannot be called by a Category 2 ISR.
E_OS_ID	The specified object identifier is invalid.
E_OS_LIMIT	The specified task activation limit has been exceeded.
E_OS_NOFUNC	Functionality of API could not be completed.
E_OS_RESOURCE	A resource is still held and must be released before calling this service.

Error Code	Description
E_OS_STATE	Attempt to carry out action on an object that is in the wrong state.
E_OS_SYS_AP_INVALID	Invalid arrivalpoint.
E_OS_SYS_AP_NULL	Null arrivalpoint.
E_OS_SYS_AP_READONLY	Attempt to alter a read-only arrivalpoint.
E_OS_SYS_CALLEVEL	The application used illegal OS services above OS level.
E_OS_SYS_CONFIG_ERROR	API call in inappropriate configuration. This may occur for the following reasons: <ul style="list-style-type: none"> The API call has activated a higher priority task, but the application is optimized for no activation of higher priority tasks. A task or Category 2 ISR has attempted to occupy a resource that it is not configured to use. The application C file has included an RTA-OSEK header file that is out-of-date for the current configuration, resulting in an illegal context for the API call. The application C file has included the wrong RTA-OSEK header file for the application code, resulting in an illegal context for the API call.
E_OS_SYS_COUNTER_INVALID	Invalid counter.
E_OS_SYS_IDLE	Call from idle task not allowed.
E_OS_SYS_R_PERMISSION	Called by a task that has not declared that it uses the message resource.
E_OS_SYS_S_INVALID	Invalid schedule handle.
E_OS_SYS_S_MISMATCH	Schedule contains a ticked/advanced counter (call only permitted for advanced/ticked).
E_OS_SYS_S_MODULO	Delay exceeds modulus of schedule.

Error Code	Description
E_OS_SYS_STACK_FAULT	The application returned from <code>StackFaultHook()</code> .
E_OS_SYS_T_INVALID	Invalid task or Category 2 ISR handle.
E_OS_SYS_TS_INVALID	Invalid taskset handle.
E_OS_SYS_TS_READONLY	Taskset is read-only.
E_OS_VALUE	The specified alarm values are outside the specified counter limits.

12 Application Build Reference

12.1 Command Line Options

The following table shows the options that can be passed to `rtabuild` on the command-line.

Note that if none of the options `-a`, `-p` or `-s` are selected, `rtabuild` automatically creates the kernel and application support files.

12.1.1 General Options

Options	Description
<code>-e</code>	Treat warnings as errors. If warnings have been output, <code>rtabuild</code> exits preceded by error E0402.
<code>-i<path></code>	Add to the search path for include files.
<code>-k</code>	Keep intermediate files. During execution, <code>rtabuild</code> may create files that contain information that is passed to other command-line tools. These files are normally deleted after use.
<code>-o[name]</code>	Generate listing file. If <code>name</code> is not specified it is constructed from the root part of the first input file with a <code>.lst</code> extension. The listing file contains all output from the launched programs, including commentary, such as the command lines invoked.
<code>-v</code>	Turn on verbose information detailing progress.
<code>-w[vvvv]</code>	Ignore all warnings (<code>-w</code>) or ignore a specific numbered warning (<code>-wvvvv</code>). If all warnings are ignored, no warnings are shown on screen or in the listing file. If a specific warning <code>vvvv</code> is to be ignored, the <code>vvvv</code> characters are used to match the warning message code. If a matching message occurs, the warning is removed from screen and listing file output. Warnings are ignored before the <code>-e</code> option can have an effect.

12.1.2 Build Options

Options	Description
-d<status>	Override the build status specified in the input OIL file. <status> = s for STANDARD <status> = t for TIMING <status> = e for EXTENDED <status> = ts for STANDARD + simple RTA-TRACE <status> = tt for TIMING + simple RTA-TRACE <status> = te for EXTENDED + simple RTA-TRACE <status> = att for TIMING + advanced RTA-TRACE <status> = ate for EXTENDED + advanced RTA-TRACE
-g	Suppress generation of vector table. This is target specific. Only relevant when generating output files.
-l	Allow very long schedules. The limit of 64K bytes for the size of target schedules is turned off. Only relevant when generating output files.
-xname	Save the input data in a new output OIL file called <code>name</code> . This has the same effect as saving a file in the RTA-OSEK GUI. If the <code>-d</code> option is used, the file generated will reflect the value specified.
-yname	Extract the RTA-OSEK custom build script into a file called <code>name</code> . Usually this is a batch file that can be run from the command-line to achieve the same effect as custom build in the RTA-OSEK GUI. Also causes <code>rtkbuild.bat</code> to be generated.

12.1.3 Analysis Options

Options	Description
-a[n]	Selects schedulability analysis and sets the schedulability algorithm used. <i>n</i> determines the depth of the analysis. It may take values 1 to 9 (1 is tractable analysis and 9 is exact analysis and the other values are reserved for future use). Note that -a is equivalent to -a1. Specifying this option with -s or -p, sets the schedulability algorithm used for sensitivity analysis and priority allocation respectively.
-c[n]	Perform clock optimization to determine the lowest clock rate at which a schedulable system can be achieved. Task priorities may be adjusted. The priority <i>n</i> determines the 'packing aggressiveness' for the priority allocation. It defaults to 1 if not specified. This option cannot be selected if either -p or -s is also selected
-nname	Ignore named task/interrupt/profile during timing analysis.
-p[n]	Select automatic task priority allocation. <i>n</i> determines the 'packing aggressiveness'. <i>n</i> defaults to 1 if not specified. This option cannot be selected if either -c or -s is also selected.
-s[name]	Perform sensitivity analysis. If <i>name</i> is specified, perform sensitivity analysis for the task or interrupt with that name. Otherwise perform analysis for each executable object in turn. This option cannot be selected if either -c or -p is also selected.
-u	Do not treat non-schedulability as an error. Only relevant for analysis.

12.2 Generated Files

RTA-OSEK generates three types of files from an OIL configuration file:

1. Multiple C header files that provide access to the standard and static interfaces of RTA-OSEK
2. A C source code file, `osekdefs.c`, which contains C data structures for RTA-OSEK and generated OSEK API calls.
3. An assembly code file, `osgen.<asm>`, which contains assembly data structures for RTA-OSEK and (when configured) the interrupt vector table.

12.2.1 Header Files

Each application program module that uses RTA-OSEK will include exactly one generated RTA-OSEK header file. The generated headers files that may be included by user code are shown in the following table:

File Name	Description
<code>Taskname.h</code> <code>ISRname.h</code>	<p>This is a task or ISR specific header file. The name is taken from the name of the task/ISR defined in your OIL configuration file.</p> <p>These header files provide access to both the static and the dynamic OS API calls for your tasks and ISRs.</p> <p>The files are protected against multiple inclusion.</p> <p>You should include each file only in application code that is executed solely by the associated task or ISR.</p>
<code>osekmain.h</code>	<p>This is the header file for <code>osek_idle_task</code> and should be included only by the main program (usually <code>main.c</code>).</p>
<code>osek.h</code>	<p>A general purpose header file. This should be included by modules containing code that is shared between tasks, such as user provided hook routines and callbacks.</p>
<code>oseklib.h</code>	<p>A header file for inclusion in code which uses RTA-OSEK API calls that you want to supply to a 3rd party as an object code library.</p> <p>The file must not be included by tasks or ISRs.</p>

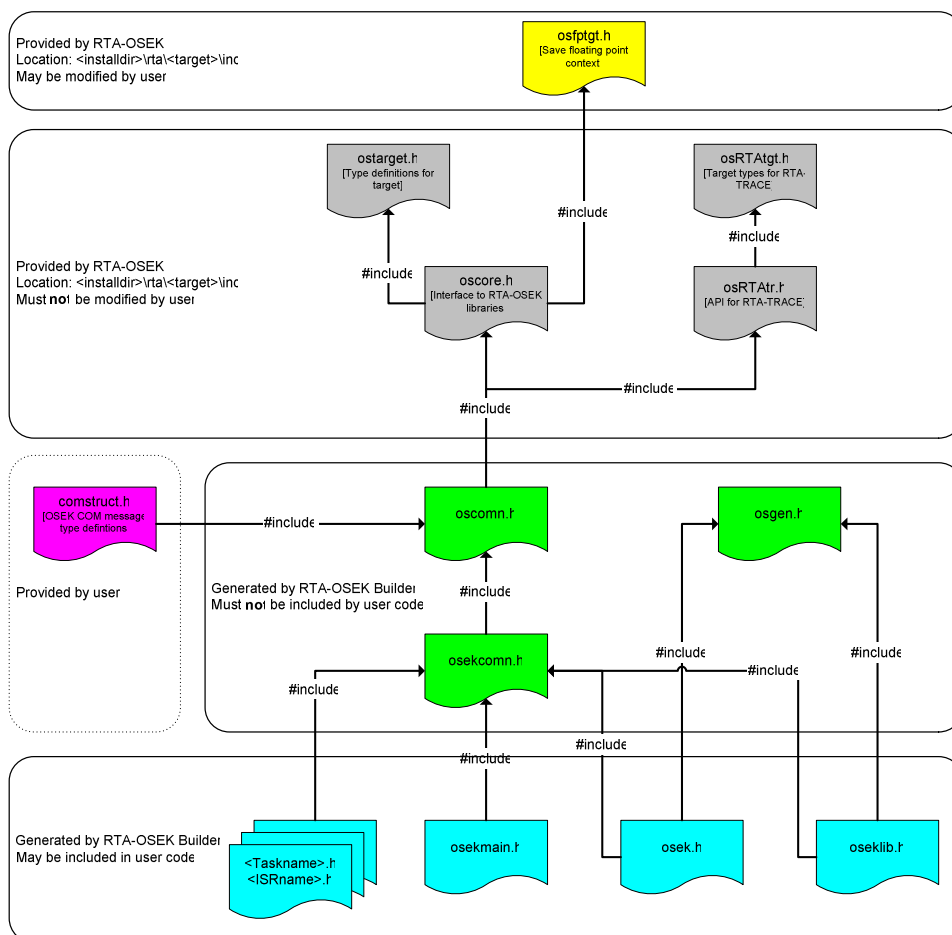
The RTA-OSEK Component library is supplied along with a primary header file `oscore.h`. This file is included in all the configuration tool generated header files. You do not, therefore, have to include it explicitly.

The RTA-OSEK builder generates 3 header files that contain definitions of various symbols, indicating the build of RTA-OSEK Component being used:

File Name	Description
osekcomm.h	Definitions that select enabled optimizations from the RTA-OSEK library, definitions of OSEK objects.
oscomm.h	Defines the build level of the system, the mapping between internal RTA-OSEK object names and internal data structures.
osgen.h	Defines the mapping between internal RTA-OSEK object names and internal data structures for resources.

12.2.2 Header File Include Structure

The header file include hierarchy is shown in the following diagram:



12.3 RTA-OSEK Libraries

RTA-OSEK is provided as 3 pre-compiled library files, one for each of the three supported build levels:

1. `rtk_s.<lib>`
The **standard** build library
2. `rtk_t.<lib>`
The **timing** build library
3. `rtk_e.<lib>`
The **extended** build library

For versions of RTA-OSEK which support RTA-TRACE, there are an additional 5 pre-compiled libraries which contain RTA-TRACE instrumentation:

4. `rtk_ts.<lib>`
The **standard** build library with **simple** tracing
5. `rtk_tt.<lib>`
The **timing** build library with **simple** tracing
6. `rtk_te.<lib>`
The **extended** build library with **simple** tracing
7. `rtk_at.<lib>`
The **timing** build library with **advanced** tracing
8. `rtk_ate.<lib>`
The **extended** build library with **advanced** tracing

Your application must link against the correct library. This is indicated in the Application -> Implementation notes in the RTA-OSEK GUI.

12.4 RTA-OSEK Builder

You can build an RTA-OSEK application with the RTA-OSEK Builder. The Builder provides a set of environment variables and macro definitions that can be used with your RTA-OSEK Build script.

12.4.1 Environment Variables

When using RTA-OSEK's builder the following environment variables are available.

You should check that the environment variables defined in `<installdir>\rta\<target>\Toolinit.bat` are correct for your toolchain.

The environment variables defined in `rtkbuild.bat` are set by RTA-OSEK when generating the `rtkbuild.bat` build script helper file. The values of these variables are defined for each target in the appropriate *RTA-OSEK Binding Manual*.

The variables can be used in a **Custom Build > Build Script** using the DOS convention of %Name%.

Name	Content	Definition
%CBASE%	The base location of your compiler and tools. E.g. C:\compiler	Toolinit.bat
%CC%	The fully qualified path to your compiler. E.g. %CBASE%\bin\cc.exe	Toolinit.bat
%AS%	The fully qualified path to your assembler. E.g. %CBASE%\bin\as.exe	Toolinit.bat
%LNK%	The fully qualified path to your linker. E.g. %CBASE%\bin\lnk.exe	Toolinit.bat
%AR%	The fully qualified path to your librarian. E.g. %CBASE%\bin\ar.exe	Toolinit.bat
%CBASE_INC%	The fully qualified path to your compiler's include files. E.g. %CBASE%\inc	Toolinit.bat
%COPTS%	Default options for compiling C source code file in rtkbuild.bat. You can add options to this by declaring an extra environment variable APP_COPT in the environment section of the custom build setup. e.g. APP_COPT=-debug	rtkbuild.bat
%AOPTS%	Default options for assembling files in rtkbuild.bat. You can add options to this by declaring an extra environment variable APP_AOPT in the environment section of the custom build setup.	rtkbuild.bat

12.4.2 Macros

Name	Content
\$ (ASMEXT)	The extension used for assembler files with this compiler toolchain, for example <code>asm</code> .
\$ (CONFORMANCE)	This macro indicates certain aspects of the build – essential for building libraries. '1' – no shared priorities in the system; '2C' – at least one task has multiple activations, but no shared priorities; '2F' – at least two tasks have the same priority; 'e' – used as a qualifier to the above three indicators, this shows whether events are used (i.e. an ECC system). In summary, there are six permutations: '1', '1e', '2C', '2Ce', '2F', and '2Fe'
\$ (DIR)	The fully qualified path to the directory in which the OIL file is stored. This is the default location into which the build files are generated.
\$ (EDITOR)	The name of the default editor used by the RTA-OSEK GUI. By default, this is the Windows Notepad application, but you can specify your own editor via the RTA-OSEK GUI System Options (accessed by selecting the Options from the File menu).
\$ (LIBEXT)	The extension used for library files with this compiler toolchain, for example <code>lib</code> .
\$ (LIBTYPE)	'S' 'T' or 'E' (corresponding to 'Standard', 'Timing' or 'Extended').
\$ (NAME)	The application name. This is the name of the OIL file omitting the <code>.oil</code> extension. For example, 'UserApp'.
\$ (OBJEXT)	The extension used for object files with this compiler toolchain, for example <code>'obj'</code> .
\$ (OPENFILE)	Causes an 'Open File' dialog to be shown and returns the name of the file selected.

Name	Content
	The 'Quick Edit' button in the RTA-OSEK GUI has been created using the line <code>\$(EDITOR)</code> <code>\$(OPENFILE)</code> .
\$(OS_STATUS)	'Standard', 'Timing' or 'Extended'.
\$(RTABASE)	The fully qualified path to the directory of <code>RTAINIT.BAT</code> . This is installation directory for RTA tools and targets. E.g. <code>C:\rta</code>
\$(RTKOBJECTS)	A space delimited list of all the task and ISR object files that are created within <code>rtkbuild.bat</code> .
\$(RTKOBJECTS_C)	A comma delimited list of all the task and ISR object files that are created within <code>rtkbuild.bat</code> .
\$(RTKLIB)	The name of the RTA-OSEK component library to which the application must link. This differs, for example, between Standard and Extended builds.
\$(TARGET)	The target name, for example, 'HC12/COSMIC 16 task'.
\$(TGTBASE)	Directory of <code>TOOLINIT.BAT</code> . The installation directory for the current target.
\$(VARIANT)	The target variant. For example, 'Star12'. This is commonly passed as a command-line 'define' when compiling target-specific code, so that appropriate settings can be selected for different chip variants.

13 Target .ini files

13.1 What is a target .ini file?

A target .ini file is part of the mechanism which tailors the behavior of RTA-OSEK to your target hardware and debugger. To understand the role played by the target .ini file, it is necessary first to consider the target DLL.

In the folder `\instdir\bin` of your RTA-OSEK installation there will be a DLL which has a name of the form `tgt<target>.dll`. This is the target DLL and is used by RTA-OSEK to do the following:

- Supply information about the target interrupt model. This includes the available vectors and priorities and the rules that apply to their use.
- Supply information about target types and limits.
- Supply information about how to build applications.
- Supply information about ORTI debuggers.

The target DLL provides detailed information about the particular target hardware which is being used with RTA-OSEK.

A target .ini file is associated with a target DLL, and will be stored in the same folder with the name `<target>.ini`, or optionally `<target>_<variant>.ini`. The target .ini file is used to override information in the target DLL. This allows you to create a variant, in which the behavior of the target DLL, and hence of RTA-OSEK, is adjusted for your individual target hardware.

Bear in mind that the RTA-OSEK component (i.e. the operating system kernel) and low-level code generation are independent of any variations you introduce via the target .ini file or target DLL. A target variant created with a .ini file cannot be used to support new vectors, memory models or the like that are not known by the core software.

13.2 Naming the target .ini file

You saw in the previous section that a target .ini file allows you to create a variant of your RTA-OSEK installation. The target DLL can be regarded as providing a default variant, and a variant created with a target .ini file can override this default or can create a new variant altogether. The way in which you name your target .ini file determines which of these possibilities occurs.

You have seen already that the target .ini file can be called `<target>.ini` or `<target>_<variant>.ini`. If you use the first form, your target .ini file will modify the default variant. If you use the second form, you will create a new variant called `<variant>`.

When you use RTA-OSEK to create a new application, the “Select Target” dialog will show all the variants which are available. You can choose which one you want to use for your new application.

13.3 The format of a target .ini file

A target .ini file takes the same format as any other .ini file:

- The file is split into sections, each beginning with a section name in square brackets.
- Each section contains lines of the form `<attribute>=<value>`. Attribute values can contain spaces.
- Lines beginning with a semi-colon are treated as comments.

Here is an example:

```
; A comment at the start of the file
;
[globals]
def_cpuspeed = 100
osgen_name = osgen.s
```

Various other rules also apply to the contents of .ini files:

- There must be no duplication of section names.
- There must be no duplication of attribute names within a section.
- Case is ignored in attribute and section names.
- Whitespace surrounding the “=” character is ignored.
- The ordering of sections and of attributes within sections is unimportant.

Only predefined names can be used for attributes, and these are described in the rest of this chapter. Examples of attribute values are usually printed in double quotes throughout the rest of this chapter. If you want to use one of the quoted values, remove the double quotes first.

13.4 The [globals] section

The attributes in this section of the target .ini file can be split into several sets, each of which is described in one of the following subsections.

13.4.1 CPU data

This set of attributes is used to specify properties of the CPU on your target.

Attribute	Description
def_cpuspeed	The instruction cycle rate in MHz for the target CPU. This is offered as the default when creating a new application. Refer to the RTA-OSEK User Guide for a definition of the instruction cycle rate.

Attribute	Description
def_swspeed	The stopwatch rate in MHz that is achieved if the CPU runs at the rate above. This is offered as the default when creating a new application.

13.4.2 Custom build data

These attributes affect the way in which RTA-OSEK performs custom builds of your application, and in particular the way in which it generates the batch file `rtkbuild.bat`.

`rtkbuild.bat` contains calls to your compiler, assembler, etc. to build parts of your application. When `rtkbuild.bat` needs to pass a command-line option to one of these tools, it usually uses an environment variable which it assumes the tool will recognise:

- `COPTS` is used to pass command-line options to the C compiler.
- `COPTS2` is used to pass command-line options to the C compiler when it is compiling `osekdefs.c`.
- `AOPTS` is used to pass command-line options to the assembler.

However, if a command-line option contains the character "=", `rtkbuild.bat` will not use an environment variable; instead it will place the command-line option directly on the tool's command line.

Note that the environment variables set by `rtkbuild.bat` will be picked up by any calls to your compiler or assembler which you make after calling `rtkbuild.bat`.

Attribute	Description
osgen_name	The name of the main assembler file generated by RTA-OSEK, e.g. "osgen.s".
c_include	The command-line option for your compiler which is used to extend its include search path, e.g. "-i".
c_include_sep	By default, the command-line option defined in <code>c_include</code> is used for each directory to be added to your compiler's include path, e.g. <code>-ia -ib -ic</code> . If <code>c_include_sep</code> is non-empty, these are lumped into one option, separated with the value of <code>c_include_sep</code> . e.g. a value of "," for <code>c_include_sep</code> would give the command-line option <code>-ia,b,c</code> .

Attribute	Description
<code>c_include_trail</code>	Text which is suffixed to the command-line option which extends your compiler's include path. For example, if you want the command-line option to be <code>-include(folder)</code> , set <code>c_include</code> to <code>"-include("</code> and <code>c_include_trail</code> to <code>")"</code> .
<code>c_define</code>	The command-line option for your compiler which is used to define a preprocessor symbol, e.g. <code>"-d"</code> .
<code>c_defopt</code>	Default command-line options which are supplied to your compiler for all task and ISR C files, e.g. <code>"-nowiden"</code> .
<code>osekdefsc_defopt</code>	If this is non-empty, it overrides <code>c_defopt</code> for compiling <code>osekdefs.c</code> .
<code>osekdefsc_ortiopt</code>	If this is non-empty, it overrides <code>c_defopt</code> for compiling <code>osekdefs.c</code> when ORTI data is being generated, i.e. when you have specified a debugger.
<code>c_insertopt</code>	This is similar to <code>c_defopt</code> , except that the options are never placed in the environment variable <code>COPTS</code> : instead they are added directly to the C compiler's command line, immediately before the name of the file being compiled.
<code>c_prefixopt</code>	This is similar to <code>c_insertopt</code> , except that the options are added to the command line immediately <i>after</i> the name of the file being compiled.
<code>a_include</code> <code>a_include_trail</code> <code>a_define</code> <code>a_defopt</code> <code>a_insertopt</code> <code>a_prefixopt</code>	Each of these has a similar purpose to the corresponding <code>c_...</code> attribute, but each applies to the assembler rather than the C compiler.
<code>osgen_defopt</code>	If this is non-empty, it overrides <code>a_defopt</code> for assembling <code>osgen.<ext></code> , where <code><ext></code> is the extension for assembler source files on your target.
<code>extobj</code>	The extension for object files on your target, e.g. <code>"obj"</code> .
<code>extasm</code>	The extension for assembler source files on your target, e.g. <code>"asm"</code> .
<code>extlib</code>	The extension for library files on your target, e.g. <code>"lib"</code> .

13.5 The [vectors] section

The attributes in this section allow you to override the names used by RTA-OSEK for the vectors in your target's vector table. These names are visible when you are using RTA-OSEK to select the vector to associate with an ISR.

Each attribute in the [Vectors] section takes the following form:

```
0x<vector number> = <vector name>
```

<vector number> is the zero-based index of a vector in the list of vectors which is visible when you are using RTA-OSEK to select the vector to associate with an ISR. <vector number> must be expressed as a lowercase hexadecimal number, and must be prefixed with 0x as shown. <vector name> is the new name which you want to give to the vector.

Here is an example showing how to override some of the vector names in your target's vector table:

```
[vectors]
0x0 = Foo vector
0x1 = Bar vector
0xa = Flibble vector
```

13.6 The ORTI debugger sections

13.6.1 Describing an ORTI debugger

The ORTI Guide describes how RTA-OSEK can generate a file with the extension .ort which tells your debugger about your application. No two ORTI debuggers are the same, and there have been several versions of the ORTI specification, so the way in which RTA-OSEK generates the .ort file differs from one debugger to the next. One of the functions of the target DLL, as described above, is to give RTA-OSEK details of the behavior of the particular debugger which is supported by your installation of RTA-OSEK.

There are a number of attributes which are used to describe your ORTI debugger. Four sets of numbered presets exist for these attributes. The presets are:

- Preset 0. A description of a generic ORTI debugger.
- Preset 1. A description of Noral Flex 4.2.
- Preset 2. A description of CrossView OSEK/ORTI v2.0.
- Preset 3. A description of Cosmic Zap 3.5 and 3.6.

The attributes, and their values under each of these presets, are shown in the following tables.

Attribute	Description
-----------	-------------

header_comments	Indicates whether comments are allowed at the start of a .ort file.
inline_comments	Indicates whether comments, preceded by //, are allowed at the end of a line in a .ort file.
enum_addresses	Indicates whether enums can be declared in the form ENUM <address> [...] in a .ort file.
totrace_used	Indicates whether the debugger understands the meaning of the least significant bit of the SERVICETRACE ORTI attribute.

Attribute	Description
<code>orti_task</code>	If true, indicates that the contents of <code>osek_running_task</code> cannot be decoded directly by the debugger. The constant <code>orti_task</code> is inserted into <code>osekdefs.c</code> to allow indirect access to the contents of <code>osek_running_task</code> .
<code>orti_active</code>	If true, indicates that the contents of certain internal variables which indicate the active task or ISR cannot be decoded directly by the debugger. Constants are inserted into <code>osekdefs.c</code> to allow indirect access to the contents of these variables.
<code>orti_lasterror</code>	If true, indicates that the contents of <code>osek_last_error</code> cannot be decoded directly by the debugger. The constant <code>orti_err</code> is inserted into <code>osekdefs.c</code> to allow indirect access to the contents of <code>osek_last_error</code> .
<code>orti_appmode</code>	If true, indicates that the contents of <code>osek_cur_mode</code> cannot be decoded directly by the debugger. The constant <code>orti_mode</code> is inserted into <code>osekdefs.c</code> to allow indirect access to the contents of <code>osek_cur_mode</code> .
<code>orti_trace</code>	If true, indicates that the contents of <code>osek_orti_call</code> cannot be decoded directly by the debugger. The constant <code>orti_trace</code> is inserted into <code>osekdefs.c</code> to allow indirect access to the contents of <code>osek_orti_call</code> .
<code>max_string_size</code>	If non-zero, string values in a <code>.ort</code> file will be limited to the number of characters specified.
<code>use_currentservice_name</code>	Indicates whether <code>CURRENTSERVICE</code> should be used instead of <code>SERVICETRACE</code> in <code>.ort</code> files.
<code>var_prefix</code>	This is used to prefix named variables which are referred to by a <code>.ort</code> file and which are not otherwise covered by other prefixes in this table. For example, the CrossView debugger requires the variable <code>MyVar</code> to be accessed as <code>_MyVar</code> .
<code>ienum_address</code>	When <code>enum_addresses</code> is set, enums in a <code>.ort</code> file will be declared as <code>ENUM <ienum_address> [...]</code> . Debuggers conforming to ORTI v2.0c have <code><ienum_address></code> set to <code>ADDRESS</code> , but later ones use a type such as <code>UINT16</code> .
<code>itask_prefix</code>	Specifies the prefix to be used for task and Category 2 ISR names in the implementation section of a <code>.ort</code> file.

<code>dtask_prefix</code>	Specifies the prefix to be used for task and Category 2 ISR names in the information section of a .ort file.
Attribute	Description
<code>dpri_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_pri</code> in the information section of a .ort file.
<code>dtrace_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_trace / osek_orti_call</code> in the information section of a .ort file ¹ .
<code>derr_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_err / osek_last_error</code> in the information section of a .ort file ¹ .
<code>dmode_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_mode / osek_cur_mode</code> in the information section of a .ort file ¹ .
<code>drdy_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_rdy / OS_L0001</code> in the information section of a .ort file ¹ .
<code>dwait_prefix</code>	Specifies the prefix to be used when referring to the variable <code>orti_wai / OS_L0004</code> in the information section of a .ort file ¹ .
<code>dtime_prefix</code>	Specifies the prefix to be used when referring to variables of type <code>TickType</code> in the information section of a .ort file.
<code>dboolptr_prefix</code>	Specifies the prefix to be used when referring to variables of type pointer-to-boolean in the information section of a .ort file.

Some of the attributes described in the above table have boolean types, i.e. they indicate whether something is true or false. When such an attribute is set to "true", its value is "1"; when it is set to "false", its value is "0". This can be seen in the following table of preset values.

Attribute	Preset 0	Preset 1	Preset 2	Preset 3
<code>header_comments</code>	0	1	0	0
<code>inline_comments</code>	0	1	0	0
<code>enum_addresses</code>	0	1	1	0
<code>totrace_used</code>	0	0	0	0
<code>orti_task</code>	0	1	0	0
<code>orti_active</code>	0	1	0	0
<code>orti_lasterror</code>	0	0	0	0

¹ Recall that one of these variables is used when the other cannot be decoded directly by the debugger.

Attribute	Preset 0	Preset 1	Preset 2	Preset 3
orti_appmode	0	0	0	0
orti_trace	0	0	0	0
max_string_size	0	0	55	0
use_currentservice_name	0	1	0	0
var_prefix	<em pty>	<em pty>	–	<em pty>
ienum_address	ADD RES S	ADD RES S	UINT16	ADD RES S
itask_prefix	<em pty>	<em pty>	*_	<em pty>
dtask_prefix	<em pty>	<em pty>	*_	<em pty>
dpri_prefix	<em pty>	<em pty>	*_	<em pty>
dtrace_prefix	<em pty>	<em pty>	*(unsi gned char *)_	<em pty>
derr_prefix	<em pty>	<em pty>	*(unsi gned char *)_	<em pty>
dmode_prefix	<em pty>	<em pty>	*(unsi gned char *)_	<em pty>
drdy_prefix	<em pty>	<em pty>	*_	<em pty>
dwait_prefix	<em pty>	<em pty>	*_	<em pty>
dtime_prefix	<em pty>	<em pty>	*_	<em pty>
dboolptr_prefix	<em pty>	<em pty>	(unsig ned char *)*_	<em pty>

A debugger is most easily described by choosing one of these presets and, if necessary, further modifying some of the attribute values to create a new debugger description with a new name. Your target DLL does this already in

order to describe the debugger which is supported by your installation of RTA-OSEK.

If your debugger differs from the one supported by your installation of RTA-OSEK you can use the target .ini file to override or add to the information in the target DLL and tell RTA-OSEK about the behavior of your debugger. How you do this depends on whether you want to override an existing debugger description or add a new debugger description.

13.6.2 Overriding an existing debugger description

If you want to use the target .ini file to override an existing debugger description in your installation of RTA-OSEK, you must first find the name of the debugger description which you want to modify. To do this, you need to select the "Debugger" subgroup of the "Target" group on the navigation bar in the RTA-OSEK GUI. In the workspace you will see the name of the currently selected debugger description. The first step towards overriding this debugger description is to copy the description name (i.e. the text enclosed in double quotes) to the clipboard.

Open your target .ini file and paste the debugger description into it as a new section name, enclosed in square brackets. Now you can add attribute values, taken from the above tables, into the new section to override the values provided by the target DLL.

For example, suppose your installation of RTA-OSEK supports the Debug-o-matic debugger. If you select the "Debugger" subgroup of the "Target" group on the navigation bar in the RTA-OSEK GUI, the workspace will contain the text:

```
Debug output type "Debug-o-matic", version 2.0.
```

In order to override certain attributes for the Debug-o-matic debugger, you should create a section in the target .ini file which looks like this:

```
[Debug-o-matic]
orti_task = 1      ; Override orti_task
orti_trace = 1    ; Override orti_trace
```

13.6.3 Adding a new debugger description

If you want to use the target .ini file to add a new debugger description to your installation of RTA-OSEK, you must first count how many descriptions are already supported by your installation. To do this, you need to select the "Debugger" subgroup of the "Target" group on the navigation bar in the RTA-OSEK GUI. Click on the "Debugger" button in the workspace and count how many debugger descriptions are in the dropdown list on the "Select debugger" dialog which is displayed, ignoring the description entitled "<none>". We will refer to the number of debugger descriptions as *n*.

Edit your target.ini file and create a section entitled [ORTI] as follows:

```
[ORTI]
ndebuggers = <n+1>
debugger_name_<n> = <Name of new debugger>
```


Then create another section in the target .ini file and give it the heading [*<Name of new debugger>*]. At this point you must decide which of the four presets (numbered 0-3 and described above) you want to use as a base for your new debugger description. Once you have made your choice, add an attribute named `vendor` to the new section, with a value equal to the preset you have chosen. You can then adjust individual attributes from their preset values. The finished result will look like this:

```
[ORTI]
ndebuggers = <n+1>
debugger_name_<n> = <Name of new debugger>

[<Name of new debugger>]
vendor = <number from 0-3>
<attribute> = <value>
<attribute> = <value>
...
```

An example will help to make this clear. Suppose your RTA-OSEK installation already supports two debuggers and you want to add support for a third debugger called Debug-o-matic. You would create the following lines in your target .ini file:

```
[ORTI]
ndebuggers = 3
debugger_name_2 = Debug-o-matic

[Debug-o-matic]
vendor = 1           ; Base it on Noral Flex 4.2
orti_task = 0       ; Make a few further changes
orti_trace = 1      ;
```

Once you have done this, the description "Debug-o-matic" will appear in the dropdown list of debugger descriptions in the "Select debugger" dialog mentioned above. You can select this description from the "Select debugger" dialog in order to use it when a .ort file is created by RTA-OSEK.

13.6.4 Choosing new attribute values

We have seen how you can override an existing debugger description or add a new debugger description by adding attributes to the target .ini file. Choosing which values to assign to these attributes can often be an iterative process:

- Change a value in the target .ini file.
- Use RTA-OSEK to generate a .ort file.
- Load the .ort file into your debugger and see whether it functions as you expect.
- Repeat as necessary.

Sometimes it is easier to try entering a formula directly into the debugger, make changes until the debugger is able to decode it correctly, and then adjust the target .ini file to generate the correct formula.

14 Index

A

AccessNameRef 2-1
 ActivateTask() 3-4
 ActivateTaskset() 3-6
 Advanced Counter Driver Interface 5-1
 Advanced Schedule Driver Interface 5-1
 AdvanceSchedule() 3-8
 AlarmBaseRefType 2-1
 AlarmBaseType 2-1, 2-3
 ALARMCALLBACK 8-1
 ALARMCALLBACK() 10-1
 AlarmType 2-1
 API call template 3-2
 API calls 3-1
 AppModeType 2-1
 ArrivalpointConstType 2-1
 ArrivalpointRefType 2-1
 ArrivalpointType 2-1
 AssignTaskset() 3-10
 AUTOSAR 3-1

B

BooleanRefType 2-1
 BooleanType 2-1
 ByteType 2-1

C

Cancel_< CounterID >() 5-7
 Cancel_<ScheduleID>() 5-3
 CancelAlarm() 3-12
 ChainTask() 3-14
 ChainTaskset() 3-16
 ClearEvent() 3-18
 CloseCOM() 3-20
 COMCALLBACK 8-2

Command Line Options 12-1
 Conformance 3-1
 Constructional Elements 4-1, 11-5
 Counted activation 9-3
 CounterType 2-1
 CycleType 2-1

D

DeclareAccessor() 4-1
 DeclareAlarm() 4-2
 DeclareCounter() 4-3
 DeclareEvent() 4-4
 DeclareFlag() 4-5
 DeclareISR() 4-6
 DeclareMessage() 4-7
 DeclareResource() 4-8
 DeclareTask() 4-9
 DisableAllInterrupts() 3-22
 Dynamic Interface 3-1
 Dynamic Interface Calls 11-1

E

EnableAllInterrupts() 3-24
 Environment Variables 12-6
 Error codes 11-11
 ErrorHook() 7-1
 EventMaskRefType 2-1
 EventMaskType 2-1
 Execution Time 6-1
 Execution Time Monitoring Interface
 11-7
 Expiry 2-4

F

FlagType 2-2
 FlagValue 2-2

G	
Generated files	12-4
GetActiveApplicationMode()	3-25
GetAlarm()	3-26
GetAlarmBase()	3-28
GetArrivalpointDelay()	3-30
GetArrivalpointNext()	3-32
GetArrivalpointTasksetRef()	3-34
GetCounterValue()	3-36
GetEvent()	3-38
GetExecutionTime()	6-1
GetISRID()	3-40
GetLargestExecutionTime()	6-3
GetMessageResource()	3-41
GetMessageStatus()	3-43
GetResource()	3-45
GetScheduleNext()	3-47
GetScheduleStatus()	3-49
GetScheduleTableStatus()	3-51
GetScheduleValue()	3-53
GetStackOffset()	3-55
GetStopwatch()	6-5
GetStopwatchUncertainty()	6-6
GetTaskID()	3-56
GetTasksetRef()	3-58
GetTaskState()	3-60

H

Header files	12-4
Hook Routines	7-1

I

Include Hierarchy	12-5
IncrementCounter()	3-62
InitCOM()	3-64
InitCounter()	3-66
Initialization	5-2
ISR	10-1
ISRname.h	12-4

ISRType	2-2
---------------	-----

L

Libraries	12-5
-----------------	------

M

Macro Definitions	10-1
Macros	12-8
MergeTaskset()	3-68
MessageInit()	7-4

N

NextScheduleTable()	3-70
Now_<CounterID>()	5-8
Now_<ScheduleID>()	5-4

O

os_all_tasks	9-2
OS_ATOMIC(expr)	10-1
OS_CALLBACK()	10-1
OS_EXTENDED_BUILD	9-2
OS_HOOK	10-1
OS_MAIN	10-1
OS_NO_SCHEDULECALL	9-3
os_no_tasks	9-2
OS_NONREENTRANT()	10-1
os_ready_tasks	9-2
OS_STANDARD_BUILD	9-2
OS_STATUS_PENDING	2-4
OS_STATUS_PENDING()	10-1
OS_STATUS_RUNNING	2-4
OS_STATUS_RUNNING()	10-1
OS_STC_COMPATIBLE	9-3
OS_TIMING_BUILD	9-2
osAdvanceCounter_<CounterID>() ..	3-72
oscomn.h	12-5
oscore.h	12-4
OSDEFAULTAPPMODE	9-2
OSEK conformance	3-1

osek.h 12-4, 12-5
 OSEK_BCC1 9-3
 OSEK_BCC2 9-3
 OSEK_BCC2C 9-3
 OSEK_BCC2F 9-3
 osek_cc2_tasks 9-2
 osek_ecc_tasks 9-2
 OSEK_ECC1 9-3
 OSEK_ECC2 9-3
 OSEK_ECC2C 9-3
 OSEK_ECC2F 9-3
 osekcomn.h 12-5
 oseklib.h 12-4
 osekmain.h 12-4
 OSError_APICallName_Param 10-1
 OSErrorGetServiceID 10-1
 OSMAXALLOWEDVALUE 9-1
 OSMAXALLOWEDVALUE_x .. 9-1, 11-9
 OSMINCYCLE 9-1, 11-9
 OSMINCYCLE_x 9-1, 11-9
 osResetOS() 3-74
 OSServiceIDType 2-2
 OSTICKDURATION 9-1
 OSTICKSPERBASE 9-1
 OSTICKSPERBASE_x 9-1
 OverrunHook() 7-3

P

PostTaskHook() 7-5
 Predefined Objects 9-1
 PreTaskHook() 7-6
 Priority
 Allocation 12-3

Q

Quick Reference Guide 11-1

R

ReadFlag() 3-75
 READY 9-1
 ReceiveMessage() 3-77
 References 1-1
 ReleaseMessageResource() 3-79
 ReleaseResource() 3-81
 RemoveTaskset() 3-83
 RES_SCHEDULER 9-2
 ResetFlag() 3-85
 ResetLargestExecutionTime() 6-7
 ResourceType 2-2
 ResumeAllInterrupts() 3-87
 ResumeOSInterrupts() 3-89
 rtabuild 12-1
 RTA-OSEK API Reference 3-1
 RTA-OSEK features 3-1
 rtk_e 12-5
 rtk_s 12-5
 rtk_t 12-5
 RUNNING 9-1

S

Schedule() 3-91
 ScheduleStatusRefType 2-2
 ScheduleStatusType 2-2, 2-4
 ScheduleTableStatusRefType 2-2
 ScheduleTableStatusType 2-2
 ScheduleTableType 2-2
 ScheduleType 2-2
 SendMessage() 3-93
 Set_<CounterID>() 5-9
 Set_<ScheduleID>() 5-5
 SetAbsAlarm() 3-95
 SetArrivalpointDelay() 3-97
 SetArrivalpointNext() 3-99
 SetEvent() 3-101

SetRelAlarm()	3-103
SetScheduleNext()	3-106
ShutdownHook()	7-7
ShutdownOS()	3-108
SmallType	2-2
StackFaultHook()	7-8
StackOffsetRefType	2-2
StackOffsetType	2-2
StartCOM()	3-109
StartOS()	3-111
StartSchedule()	3-112
StartScheduleTable()	3-114
StartupHook()	7-10
State_<CounterID>()	5-10
State_<ScheduleID>()	5-6
Static and Dynamic Interface	3-1
Static Interface	3-1
Static Interface RTA-OSEK API Calls	11-4
StatusType	2-2
StopCOM()	3-116
StopSchedule()	3-118
StopScheduleTable()	3-120
Stopwatch device driver	6-5
StopwatchTickRefType	2-2
StopwatchTickType	2-3
Summary	
Advanced Counter Driver Interface	11-6
Advanced Schedule Driver Interface	11-5
Constructional Elements	11-5
Dynamic Interface Calls	11-1
Error Codes	11-11
Execution Time Monitoring Interface	11-7
Hooks	11-7
Macro Definitions	11-9
Other Callbacks	11-7
Predefined Objects	11-8

Static Interface Calls	11-4
SuspendAllInterrupts()	3-122
SUSPENDED	9-1
SuspendOSInterrupts()	3-124
SymbolicName	2-3

T

TASK	10-1
Taskname.h	12-4
TaskRefType	2-3
Tasks	12-3
TasksetConstType	2-3
TasksetRefType	2-3
TasksetType	2-3
TaskStateRefType	2-3
TaskStateType	2-3
TaskType	2-3
Template	3-2
TerminateTask()	3-126
TestArrivalpointWritable()	3-128
TestEquivalentTaskset()	3-130
TestSubTaskset()	3-132
Tick Source Semantics	5-1
Tick_<CounterID>()	3-134
TickRefType	2-3
TickSchedule()	3-136
TickType	2-3
Type Definitions	2-1

U

UInt16Type	2-3
UInt32Type	2-3
UIntType	2-3
User Provided	
ALARMCALLBACK	8-1
Cancel_<CounterID >()	5-7
Cancel_<ScheduleID>()	5-3
COMCALLBACK	8-2
ErrorHook()	7-1

GetStopwatch() 6-5
GetStopwatchUncertain
ty() 6-6
MessageInit() 7-4
Now_<CounterID>() 5-8
Now_<ScheduleID>() 5-4
OverrunHook() 7-3
PostTaskHook() 7-5
PreTaskHook() 7-6
Set_<CounterID>() 5-9

Set_<ScheduleID>() 5-5
ShutdownHook() 7-7
StackFaultHook() 7-8
StartupHook() 7-10
State_<CounterID>() 5-10
State_<ScheduleID>() 5-6

W

WaitEvent() 3-138
WAITING 9-1