RTA-OS3.0

Reference Guide

Copyright

Document: 10384-RG-1.0.0

Contents

1 Introduction

RTA-OS3.0 is a statically configurable, pre-emptive, real-time operating system (RTOS) for use in high-performance, resource-constrained applications. RTA-OS3.0 is a full implementation of the open-standard AUTOSAR OS Release 3.0 (Scalability Class 1) and includes functionality that is fully compliant and independently certified to Version 2.2.3 of the OSEK/VDX OS Standard.

This guide contains the complete technical reference for RTA-OS3.0. The content is arranged into two parts:

- Part 1 deals with the OS kernel, describing the API, types, macros, etc. that are supported by RTA-OS3.0 and common to all target hardware.
- Part 2 deals with the PC-based tooling provided with RTA-OS3.0. The command line interfaces, input and output file formats etc. that are common to all target hardware are described.

For each supported target there is an *RTA-OS3.0 Target/Compiler Port Guide* which provides auxiliary details for port-specific OS features.

1.1 About You

You are a trained embedded systems developer who wants to build real-time applications using a pre-emptive operating system. You should have knowledge of the C programming language, including the compilation, assembling and linking of C code for embedded applications with your chosen tool chain. Elementary knowledge about your target microcontroller, such as the start address, memory layout, location of peripherals as so on, is essential.

You should also be familiar with common use of the Microsoft Windows®2000, Windows®XP or Windows®Vista operating systems, including installing software, selecting menu items, clicking buttons, navigating files and directories.

1.2 Document Conventions

The following conventions are used in this guide:

Choose **File > Open**.	Menu options are printed in **bold, blue** characters.
Click **OK**.	Button labels are printed in **bold** characters
Press <Enter>.	Key commands are enclosed in angle brackets.
The “Open file” dialog box appears	The names of program windows, dialog boxes, fields, etc. are enclosed in double quotes.

`Activate(Task1)`	Program code, header file names, C type names, C functions and RTA-OS3.0. Component API call names all appear in the `courier` typeface.
See Section 1.2.	Hyperlinks through the document are shown in red letters.
	Functionality that is provided in RTA-OS3.0 but it may not be portable to another AUTOSAR OS implementation is marked with the ETAS logo.
	Caution! Notes like this contain important instructions that you must follow carefully in order for things to work correctly.

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, please 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-OS3.0 API calls

2.1 Guide to Descriptions

All API calls have the following structure:

Syntax

```
/* C function prototype for the API call */
ReturnValue NameOfAPICall(Parameter Type, ...)
```

Parameters

A list of parameters for each API call and their mode:

in The parameter is passed in to the call

out The parameter is passed out of the API call by passing a reference (pointer) to the parameter into the call.

inout The parameter is passed into the call and then (updated) and passed out.

Return Values

Where API calls return a `StatusType` the values of the type returned and an indication of the reason for the error/warning are listed. The build column indicates whether the value is returned for both standard and extended status builds or for extended status build only.

Description

A detailed description of the behavior of the API call.

Portability

The RTA-OS3.0 API includes four classes of API calls:

OSEK OS calls are those specified by the OSEK OS standard. OSEK OS calls are portable to other implementations of OSEK OS and are portable to other implementations of AUTOSAR OS R3.0.

AUTOSAR OS calls are those specified by the AUTOSAR OS R3.0 standard. AUTOSAR OS calls are portable to other implementations of AUTOSAR OS R3.0. The calls are portable to OSEK OS only if the call is also an OSEK OS call.

RTA-TRACE calls are provided by RTA-OS3.0 for controlling the RTA-TRACE run-time profiling tool. These calls are only available when RTA-TRACE support has been configured.

RTA-OS3.0 calls include all those form the other three classes plus calls that provide extensions AUTOSAR OS functionality. These calls are unique to RTA-OS3.0 and are not portable to other implementations.

Example Code

```
A C code listing showing how to use the API calls
```

Calling Environment

The valid calling environment for the API call. A ✓ indicates that a call can be made in the indicated context. A ✗ indicates that the call cannot be made in the indicated context.

See Also

A list of related API calls.

2.2 ActivateTask

Activate a task.

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	The task to activate.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_LIMIT	all	The requested activation would exceed the maximum number of queued activations specified by configuration. The requested activation is ignored.
E_OS_ID	extended	TaskID is not a valid TaskType.

Description

If TaskID is in the suspended state then it is transferred into the ready state.

If TaskID is in either the ready or the running state and the total number of queued activations is less than the task activation limit then the requested activation is queued.

Rescheduling behavior depends on the caller:

- if the caller is a non-preemptive task the rescheduling does not occur until the caller terminates or makes a Schedule() call.

- if the caller is a preemptive task and TaskID is higher priority then rescheduling will take place immediately.

- if the caller is a Category 2 ISR then rescheduling will not occur until the Category 2 ISR terminates.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  ActivateTask(YourTask);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareTask
ChainTask
TerminateTask
GetTaskState
GetTaskID

2.3 CancelAlarm

Cancel an alarm.

Syntax

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

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Name of the alarm to cancel.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_NOFUNC	all	AlarmID is not running.
E_OS_ID	extended	AlarmID is not a valid alarm.

Description

This call cancels (stops) the specified alarm.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyExtendedTask){
  ...
  CancelAlarm(TimeOutAlarm);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

CancelAlarm
DeclareAlarm
GetAlarm
GetAlarmBase
SetRelAlarm

2.4 ChainTask

Terminate the calling task and activate another task

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	The task to be activated

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OS_LIMIT	all	The requested activation would exceed the maximum number of queued activations specified by configuration. The requested activation is ignored.
E_OS_ID	extended	TaskID is not a valid TaskType.
E_OS_RESOURCE	extended	Calling task still holds resources.
E_OS_CALLEVEL	extended	Called at interrupt level.

Description

This service causes the termination of the calling task followed by the activation of TaskID. A successful call of ChainTask() does not return to the calling context.

Internal resources held by the calling task are released automatically.

Standard or linked resources held by the calling task are also released automatically and this is reported as an error condition in extended status.

A task can chain itself without affecting the queued activation count.

The ChainTask() call always causes re-scheduling. However, note that TaskID may not run immediately - there may be higher priority tasks in the ready queue that will run in preference, for example tasks with a higher priority that share an internal resource with TaskID.

If the 'Fast Terminate' is enabled in Optimizations for RTA-OS then ChainTask() must only be called from the task entry function and the return status should not be checked (ErrorHook, when configured, will be called if there is an error). This optimization saves memory and execution time.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  ChainTask(YourTask);
  /* Any code here will not execute if the call is successful
     */
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareTask
ActivateTask
TerminateTask
GetTaskState
GetTaskID

2.5 ClearEvent

Clear one (or more) events from the task's event mask.

Syntax

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

Parameters

Name	Type	Mode	Description
Mask	EventMaskType	in	The event(s) to be cleared.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ACCESS	extended	Not called from an extended task.
E_OS_CALLEVEL	extended	Called from interrupt level.

Description

The events of the extended task calling ClearEvent are cleared according to the event mask Mask.

Any events that are not set in the event mask remain unchanged.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyExtendedTask){
  EventMaskType WhatHappened;
  while (WaitEvent(Event1 | Event2 | Event3) == E_OK ) {
    GetEvent(MyExtendedTask, &WhatHappened);
    if (WhatHappened & Event1) {
      ClearEvent(Event1);
      /* Take action on Event1 */
      ...
    } else if (WhatHappened & (Event2 | Event3) {
      ClearEvent(Event2 | Event3);
      /* Take action on Event2 or Event3 */
      ...
    }
```

```
  }
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareEvent
GetEvent
SetEvent
WaitEvent

2.6 DisableAllInterrupts

Disables (masks) all interrupts for which the hardware supports disabling.

Syntax

```
void DisableAllInterrupts(void)
```

Return Values

None.

Description

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

The call does not support nesting. If nesting is needed for critical sections, e.g. for libraries, then SuspendAllInterrupts()/ResumeAllInterrupts() should be used.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  DisableAllInterrupts();
    /* Critical section */
    /* No RTA-OS3.0 API calls allowed */
  EnableAllInterrupts();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

EnableAllInterrupts
ResumeAllInterrupts
ResumeOSInterrupts
SuspendAllInterrupts
SuspendOSInterrupts

2.7 EnableAllInterrupts

Enables (unmasks) all interrupts.

Syntax

```
void EnableAllInterrupts(void)
```

Return Values

None.

Description

This API call marks the end of a critical section that is protected from any maskable interrupt occuring. The critical section must have been entered using the DisableAllInterrupts() call.

This call restores the state of the interrupt mask saved by DisableAllInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  DisableAllInterrupts();
    /* Critical section */
    /* No RTA-OS3.0 API calls allowed */
  EnableAllInterrupts();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DisableAllInterrupts
ResumeAllInterrupts
ResumeOSInterrupts
SuspendAllInterrupts
SuspendOSInterrupts

2.8 GetActiveApplicationMode

Get the currently active application mode.

Syntax

```
AppModeType GetActiveApplicationMode(void)
```

Return Values

The call returns values of type AppModeType.

Description

The call returns the currently active application mode (i.e. the value of parameter that was passed to StartOS()). The call can be used to write application-mode dependant code.

It will return OS_NOAPPMODE if the OS is not running.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  if (GetActiveApplicationMode() == DiagnosticsMode) {
    /* Send diagnostic message */
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

StartOS

2.9 GetAlarm

Get the number of ticks before an alarm expires.

Syntax

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

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Name of the alarm of interest.
Tick	TickRefType	out	Reference to a TickType variable.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_NOFUNC	all	AlarmID is not currently set.
E_OS_ID	extended	AlarmID is not a valid alarm.

Description

Returns the relative number of ticks from the point at which the call was made before the alarm AlarmID is due to expire.

Note that between making this call and evaluating the out parameter Tick the task may have been pre-empted and the alarm may have already expired. Exercise caution when making program decisions based on the value of Tick.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  TickType  TicksToExpiry;
  ...
  GetAlarm(MyAlarm, &TicksToExpiry);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

CancelAlarm
DeclareAlarm
GetAlarmBase
SetAbsAlarm
SetRelAlarm

2.10 GetAlarmBase

Get properties of the counter associated with an alarm.

Syntax

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

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Name of the alarm of interest.
Info	AlarmBaseRefType	out	Reference to a AlarmBaseType structure.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	AlarmID is not a valid alarm.

Description

GetAlarmBase() reads the alarm base characteristics. These are the static properties of the counter with which AlarmID is associated.

The out parameter Info refers to a structure in which the information of data type AlarmBaseType gets stored.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  AlarmBaseType Info;
  TickType maxallowedvalue;
  TickType ticksperbase;
  TickType mincycle;

  GetAlarmBase(MyAlarm, &Info);
  maxallowedvalue = Info.maxallowedvalue;
  ticksperbase = Info.ticksperbase;
  mincycle = Info.mincycle;
```

```
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

CancelAlarm
DeclareAlarm
GetAlarm
SetAbsAlarm
SetRelAlarm

2.11 GetCounterValue

Initialize a counter to an absolute value.

Syntax

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

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	The counter to read.
Value	TickRefType	out	The current value of the counter.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	CounterID is not a valid counter.

Description

Returns the current value of the specified counter CounterID in Value.

The Operating System ensures that the lowest value is zero and consecutive reads return an increasing count value until the counter wraps.

If CounterID is a hardware counter, then the user callback Os_Cbk_Now_<CounterID> will be called.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
Task(MyTask){
  TickType Value;
  ...
  GetCounterValue(MyCounter,&Value);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

IncrementCounter
GetElapsedCounterValue
Os_AdvanceCounter
Os_AdvanceCounter_<CounterID>
Os_IncrementCounter_<CounterID>

2.12 GetElapsedCounterValue

Returns the number of elapsed ticks since the given <Value> value via <ElapsedValue>.

Syntax

```
StatusType GetElapsedCounterValue(
    CounterType CounterID,
    TickRefType Value,
    TickRefType ElapsedValue
)
```

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	Name of the counter.
Value	TickRefType	in	A previous counter value.
Value	TickRefType	out	The current value of the counter.
ElapsedValue	TickRefType	out	The difference from the in Value.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	CounterID is not a valid counter.

Description

Returns the number of ticks that have elapsed on the counter current since Value.

Value is updated with the the current value of the counter when the call returns.

Note that the call can only return a value up to maxallowedvalue ticks in length.

If the counter has ticked more than maxallowedvalue ticks since Value then ElapsedValue will be Value modulo maxallowedvalue.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
Task(MyTask){
```

```
  TickType Value;
  TickType ElapsedValue;
  ...
  GetCounterValue(MyCounterID,&Value);
  /* Value => current count */
  ...
  GetElapsedCounterValue(MyCounter,&Value,&ElapsedValue);
  /* ElapsedValue => ticks since original Value, Value =>
     current count */
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

GetCounterValue

2.13 GetEvent

Get the state of all event bits for a task.

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Name of the Task of interest.
Mask	EventMaskRefType	out	Reference to an event mask.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	TaskID is not a valid task.
E_OS_ACCESS	extended	TaskID is not an extended task.
E_OS_STATE	extended	TaskID is in the suspended state.

Description

This call returns all event that are set for the extended task TaskID.

Note that all set events are returned, regardless of which events the task may have been waiting for.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyExtendedTask){
  EventMaskType WhatHappened;
  while (WaitEvent(Event1 | Event2 | Event3) == E_OK ) {
    GetEvent(MyExtendedTask, &WhatHappened);
    if(WhatHappened & Event1) {
      ClearEvent(Event1);
      /* Take action on Event1 */
      ...
    } else if (WhatHappened & (Event2 | Event3) {
      ClearEvent(Event2 | Event3);
```

```
      /* Take action on Event2 or Event3 */
      ...
    }
  }
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

ClearEvent
DeclareEvent
SetEvent
WaitEvent

2.14 GetISRID

Get the identifier of the currently running ISR.

Syntax

```
ISRType GetISRID(void)
```

Return Values

The call returns values of type ISRType.

Description

The call returns the ID of the currently running Category2 ISR or INVALID_ISR no ISR is running. The main use of the call is to identify which ISR is running in hook functions.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) ErrorHook(StatusType Error){
  ISRType ISRInError;
  TaskType TaskInError;

  ISRInError = GetISRID();
  if (ISRInError != INVALID_ISR) {
    /* Must be an ISR in error */
  } else {
    /* Maybe its a task in error */
    GetTaskID(&TaskInError);
    ...
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

GetTaskID

2.15 GetResource

Get (lock) a resource to enter a critical section.

Syntax

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

Parameters

Name	Type	Mode	Description
ResID	ResourceType	in	The resource to get.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	ResID is not a valid resource.
E_OS_ACCESS	extended	Attempt to get a resource which is (a) already locked by another task or ISR, or (b) the priority of the calling task or interrupt routine is higher than the actual priority of ResID.

Description

This call enters a named critical section (the resource), protecting the code inside the critical section against concurrent access by any other tasks and ISRs that are configured to be able to access the resource.

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

Nested resource occupation is allowed, but only where the inner critical sections are completely executed within the surrounding critical section as shown in the example.

Nested occupation of the same resource is not allowed, although you can use linked resources to achieve this effect.

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

A system where Category 2 ISRs can lock a resource has slightly higher run-time overheads than one where only Tasks lock resources.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  GetResource(Outer);
    /* Outer Critical Section */
    ...
    GetResource(Inner);
      /* Inner Critical Section */
    ReleaseResource(Inner);
    ...
  ReleaseResource(Outer);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareResource
ReleaseResource

2.16 GetScheduleTableStatus

Get the status of a schedule table.

Syntax

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

Parameters

Name	Type	Mode	Description
ScheduleTableID	ScheduleTableType	in	Schedule table for which the status is required.
ScheduleStatus	ScheduleTableStatusRefType	in	Reference to the sched-ule table status.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	ScheduleTableID is not a valid ScheduleTable.

Description

This call returns the status of the ScheduleTableID.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
TASK(MyTask){
  ScheduleTableStatusType Status;

  GetScheduleTableStatus(MyScheduleTable, &Status);
  if (Status != SCHEUDLETABLE_RUNNING){
      StartScheduleTableAbs(MyScheduleTable,42);
```

```
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareScheduleTable
NextScheduleTable
StartScheduleTableAbs
StartScheduleTableRel
StopScheduleTable

2.17 GetTaskID

Identify the currently running task.

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskRefType	in	A reference to the running task.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.

Description

The call returns a reference to the currently running Task. If the call is made from a task, then it will return the identifer of that task. The main use of the call is to identify which task is running in hook functions.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) ErrorHook(StatusType Error){
  TaskType TaskInError;
  GetTaskID(&TaskInError);
  if (TaskInError == INVALID_TASK) {
    /* Must be an ISR in error */
  } else if (TaskInError == MyTask) {
    /* Do something */
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DeclareTask
TerminateTask
GetTaskState
GetTaskID
GetISRID

2.18 GetTaskState

Get the current state (suspended, ready, running, waiting) of a specified task.

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	The task of interest.
State	TaskStateRefType	out	Reference to the task state.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	TaskID is not a valid TaskType.

Description

The call returns the state of the task at the point GetTaskState() was called.

The main use of this API is to check that an extended task is not in the suspended state before setting an event.

A task that is pre-empted by an ISR remains in the running state.

Note that when called from a pre-emptive task or from an ISR the state may already be incorrect at the time it is evaluated because preemption may have occurred between the call returning and the result being evaluated.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  TaskStateType CurrentState;
  ...
  GetTaskState(YourTask, &CurrentState);
  switch (CurrrentState) {
```

```
    case SUSPENDED:
      /* YourTask is suspended */
    case READY:
      /* YourTask is ready to run */
    case WAITING:
      /* YourTask is waiting (for an event) */
    case RUNNING:
      /* YourTask is running. Not possible as MyTask must be
         running to make the call */
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

DeclareTask
TerminateTask
GetTaskState
GetTaskID

2.19 IncrementCounter

Increment a software counter.

Syntax

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

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	Name of the counter to increment.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	CounterID is not a software counter.

Description

This call increments (adds one to) CounterID. CounterID must be a software counter.

If any alarms on the counter are triggered by the increment then the alarm actions will be executed before the call returns.

Note that if an error occurs during the expiry of an alarm (for example, a task activation raises E_OS_LIMIT), the error hook(s) are called for each error that occurs.

However, the IncrementCounter() service itself will still return E_OK.

The API call may cause re-scheduling to take place.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
ISR(MillisecondTimerInterrupt){
  ...
  IncrementCounter(MillisecondCounter);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_AdvanceCounter
Os_AdvanceCounter_<CounterID>
Os_IncrementCounter_<CounterID>

2.20 NextScheduleTable

Change the execution pattern from one ScheduleTable to another.

Syntax

```
StatusType NextScheduleTable(
    ScheduleTableType ScheduleTableID_From,
    ScheduleTableType ScheduleTableID_To
)
```

Parameters

Name	Type	Mode	Description
ScheduleTableID_From	ScheduleTableType	in	Schedule table to switch from.
ScheduleTableID_To	ScheduleTableType	in	Schedule table to switch into.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_NOFUNC	all	ScheduleTableID_From is not started.
E_OS_ID	extended	ScheduleTableID_From or ScheduleTableID_To is not a valid ScheduleTable.
E_OS_STATUS	extended	ScheduleTableID_To is already started or nexted.

Description

This call starts the processing of schedule table of ScheduleTableID_To ScheduleTableID_From.FinalDelay ticks after the Final Expiry Point on ScheduleTableID_From has been processed.

The Initial Expiry Point on ScheduleTableID_To is processed ScheduleTable_To.InitialOffset ticks after the start of ScheduleTableID_To.

If ScheduleTableID_From already has a 'nexted' schedule table then ScheduleTableID_To replaces the previous 'nexted' schedule table and that previous table is set to state SCHEDULETABLE_STOPPED.

If either schedule table is not valid or they are driven by different counters then the states of both tables remain unchanged.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
TASK(MyTask){
    /* Stop MyScheduleTable at the end and start
       YourScheduleTable */
    NextScheduleTableAbs(MyScheduleTable, YourScheduleTable);
    ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareScheduleTable
GetScheduleTableStatus
StartScheduleTableAbs
StartScheduleTableRel
StopScheduleTable

2.21 Os_AdvanceCounter

Inform the OS that a hardware counter has reached the previously programmed value.

Syntax

```
StatusType Os_AdvanceCounter(
    CounterType CounterID
)
```

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	Name of the counter that has reached a programmed value.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	CounterID is not a hardware counter.
E_OS_STATE	extended	CounterID is not running.

Description

This call tells the OS that the counter value has matched the value previously set via the Os_Cbk_Set_<CounterID> callback.

The OS will then process any alarm or expiry point actions that are due. It will then either set a new match value (via Os_Cbk_Set_<CounterID>) or cancel counter matching (via Os_Cbk_Cancel_<CounterID>).

Note that it is possible for the new counter match value to be reached before leaving any interrupt that is being used to drive the counter. It is important that this occurance is not missed because otherwise the counter will not be awoken again until a complete wrap of the underlying hardware counter value has occured.

On some hardware platforms no special action is needed because the interrupt will simply get reasserted when the existing instance exits.

On other platforms, the interrupt has to be reasserted in software or, where this is not possible, the code must loop as shown in the example. In either case great care has to be taken to avoid missing matches that occur while the driver is executing.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
/* For systems where the interrupt will be re-entered
   automatically if the match occurs before leaving the ISR: */
ISR(SimpleCounterDriver){
  Os_AdvanceCounter(MyHWCounter);
}
/* For systems where the software can force the interrupt to
   get re-entered if the match occurs before leaving the ISR:
   */
ISR(RetriggeringCounterDriver){
  Os_ScheduleTableStatusType CurrentState;
  Os_AdvanceCounter(MyHWCounter);
  Os_Cbk_State_MyHWCounter(&CurrentState);
  if (CurrentState.Running && CurrentState.Pending) {
    /* Retrigger this interrupt */
  }
}
/* For systems where the software has to loop if the match
   occurs before leaving the ISR: */
ISR(LoopingCounterDriver){
  Os_ScheduleTableStatusType CurrentState;
  do {
    Os_AdvanceCounter(MyHWCounter);
    Os_Cbk_State_MyHWCounter(&CurrentState);
  } while (CurrentState.Running && CurrentState.Pending);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

IncrementCounter
Os_AdvanceCounter_<CounterID>
Os_IncrementCounter_<CounterID>
Os_Cbk_Set_<CounterID>
Os_Cbk_Cancel_<CounterID>
Os_Cbk_State_<CounterID>

2.22 Os_AdvanceCounter_<CounterID>

Inform the OS that a hardware counter has reached a programmed value.

Syntax

```
StatusType Os_AdvanceCounter_CounterID(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_STATE	extended	CounterID is not running.

Description

This call has the same behavior as Os_AdvanceCounter(CounterID) but is customized for a specific counter. This makes the call faster and more suitable for use in interrupt handlers.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
/* For systems where the interrupt will be re-entered
   automatically if the match occurs before leaving the ISR: */
ISR(SimpleCounterDriver){
  Os_AdvanceCounter_MyHWCounter();
}
/* For systems where the software can force the interrupt to
   get re-entered if the match occurs before leaving the ISR:
   */
ISR(RetriggeringCounterDriver){
  Os_ScheduleTableStatusType CurrentState;
  Os_AdvanceCounter_MyHWCounter();
  Os_Cbk_State_MyHWCounter(&CurrentState);
  if (CurrentState.Running && CurrentState.Pending) {
    /* Retrigger this interrupt */
  }
}
/* For systems where the software has to loop if the match
   occurs before leaving the ISR: */
ISR(LoopingCounterDriver){
  Os_ScheduleTableStatusType CurrentState;
  do {
```

```
    Os_AdvanceCounter_MyHWCounter();
    Os_Cbk_State_MyHWCounter(&CurrentState);
  } while (CurrentState.Running && CurrentState.Pending);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

IncrementCounter
Os_AdvanceCounter
Os_IncrementCounter_<CounterID>
Os_Cbk_Set_<CounterID>
Os_Cbk_Cancel_<CounterID>
Os_Cbk_State_<CounterID>

2.23 Os_GetExecutionTime

Get the execution time consumed by the calling Task/ISR.

Syntax

```
Os_StopwatchTickType Os_GetExecutionTime(void)
```

Return Values

The call returns values of type Os_StopwatchTickType.

Description

Returns the net execution time consumed (i.e. excluding all preemptions) since the start of the Task or ISR.

In the case of an extended task, execution time restarts on return from a WaitEvent() call.

The value is not valid in PreTaskHook().

Any value read in PostTaskHook() is valid, but it will be greater than the value that is used to determine a task's maximum execution time.

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

Time monitoring must be enabled for this API to give meaningful results. It returns zero if time monitoring is not enabled.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(MyTask){
  Os_StopwatchTickType Start, Finish, Used, APICallCorrection;
  Start = GetExecutionTime();
  Finish = GetExecutionTime();
  APICallCorrection = Finish - Start;  /* Get time for
     GetExecutionTime() call itself. */
  Start = GetExecutionTime();
  Call3rdPartyLibraryFunction();       /* Measure 3rd Party
     Library Code Execution Time */
  Finish = GetExecutionTime();
  Used = Finish - Start - APICallCorrection; /* Calculate the
     amount of time used. */
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_GetISRMaxExecutionTime
Os_GetTaskMaxExecutionTime
Os_ResetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime

2.24 Os_GetISRMaxExecutionTime

Get the longest observed execution time consumed by an ISR.

Syntax

```
Os_StopwatchTickType Os_GetISRMaxExecutionTime(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	The ISR of interest.

Return Values

The call returns values of type Os_StopwatchTickType.

Description

Returns the maximum observed execution time for the Category 2 ISR identified by ISRID.

This maximum value is over all complete invocations of the Category 2 ISR that have completed since the previous call to ResetISRMaxExecutionTime() for that Category 2 ISR or to StartOS().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(LoggingTask){
  Os_StopwatchTickType ExecutionTimes[MAXISRS];
  ...
  ExecutionTimes[0] = GetISRMaxExecutionTime(ISR1);
  ExecutionTimes[1] = GetISRMaxExecutionTime(ISR2);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetExecutionTime
Os_GetTaskMaxExecutionTime
Os_ResetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime

2.25 Os_GetISRMaxStackUsage

Get the maximum observed stack usage of an ISR.

Syntax

```
Os_StackSizeType Os_GetISRMaxStackUsage(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	The ISR of interest.

Return Values

The call returns values of type Os_StackSizeType.

Description

Returns the maximum observed stack usage for the Category 2 ISR identified by ISRID.

This maximum value is over all invocations of the Category 2 ISR since the previous call to ResetISRMaxStackUsage() for that Category 2 ISR or to StartOS().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(LoggingTask){
  Os_StackSizeType StackUsages[MAXISRS];
  ...
  StackUsages[0] = GetISRMaxStackUsage(ISR1);
  StackUsages[1] = GetISRMaxStackUsage(ISR2);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetStackUsage
Os_GetTaskMaxStackUsage
Os_ResetISRMaxStackUsage
Os_ResetTaskMaxStackUsage

2.26 Os_GetStackSize

Get the difference between 2 stack values.

Syntax

```
Os_StackSizeType Os_GetStackSize(
    Os_StackValueType Base,
    Os_StackValueType Sample
)
```

Parameters

Name	Type	Mode	Description
Base	Os_StackValueType	in	The position to measure the stack from.
Sample	Os_StackValueType	in	The position to measure the stack to.

Return Values

The call returns values of type Os_StackSizeType.

Description

Returns the difference between 2 Os_StackValueType values. To obtain a correct

value, it is important that 'Base' represents an instant when the stack size

was smaller than (or the same as) the point at which 'Sample' was measured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_StackValueType start_position;
Os_StackValueType end_position;
Os_StackSizeType stack_size;
TASK(MyTask){
  start_position = Os_GetStackValue();
  nested_call();
  stack_size = Os_GetStackSize(start_position, end_position);
}
void nested_call(void) {
  end_position = Os_GetStackValue();
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

Os_GetStackValue
Os_GetStackUsage

2.27 Os_GetStackUsage

Get the amount of stack consumed by the calling Task/ISR.

Syntax

```
Os_StackSizeType Os_GetStackUsage(void)
```

Return Values

The call returns values of type Os_StackSizeType.

Description

Returns the amount of stack used by the calling Task or ISR at the point of the call.

The value is measured from the point at which the OS kernel starts to run the Task or ISR, and it includes overheads within the kernel so that the values returned can be used directly in the configuration of the stack allocation budget for a Task or ISR.

Calling this API has the side-effect of updating the recorded maximum stack usage for the calling Task or ISR (where necessary).

If the Task/ISR has a stack allocation budget, then a stack overrun may be reported before this API returns.

Stack monitoring must be enabled in general OS configuration for this API to give meaningful results. It returns zero if stack monitoring is not enabled.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(MyTask){
  Os_StackSizeType stack_size;
  stack_size = Os_GetStackUsage();
  nested_call();
}
void nested_call(void) {
  Os_GetStackUsage(); /* Identifies a possible max stack usage
      location */
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_Cbk_StackOverrunHook
Os_GetISRMaxStackUsage
Os_GetTaskMaxStackUsage
Os_ResetISRMaxStackUsage
Os_ResetTaskMaxStackUsage

2.28 Os_GetStackValue

Get the current stack value.

Syntax

```
Os_StackValueType Os_GetStackValue(void)
```

Return Values

The call returns values of type Os_StackValueType.

Description

Returns the current position of the stack pointer (or pointers).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_StackValueType start_position;
Os_StackValueType end_position;
Os_StackSizeType stack_size;
TASK(MyTask){
  start_position = Os_GetStackValue();
  nested_call();
  stack_size = Os_GetStackSize(start_position, end_position);
}
void nested_call(void) {
  end_position = Os_GetStackValue();
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

Os_GetStackSize
Os_GetStackUsage

2.29 Os_GetTaskMaxExecutionTime

Get the longest observed execution time consumed by a Task.

Syntax

```
Os_StopwatchTickType Os_GetTaskMaxExecutionTime(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	The Task of interest.

Return Values

The call returns values of type Os_StopwatchTickType.

Description

Returns the maximum observed execution time for TaskID.

This maximum value is over all complete invocations of TaskID that have completed since the previous call to ResetTaskMaxExecutionTime() for TaskID or to StartOS().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(LoggingTask){
  Os_StopwatchTickType ExecutionTimes[MAXTASKS];
  ...
  ExecutionTimes[0] = GetTaskMaxExecutionTime(Task1);
  ExecutionTimes[1] = GetTaskMaxExecutionTime(Task2);
  ExecutionTimes[2] = GetTaskMaxExecutionTime(Task3);
  ExecutionTimes[3] = GetTaskMaxExecutionTime(Task4);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetExecutionTime
Os_GetISRMaxExecutionTime
Os_ResetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime

2.30 Os_GetTaskMaxStackUsage

Get the maximum observed stack usage of a Task.

Syntax

```
Os_StackSizeType Os_GetTaskMaxStackUsage(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	The Task of interest.

Return Values

The call returns values of type Os_StackSizeType.

Description

Returns the maximum observed stack usage for TaskID.

This maximum value is over all invocations of TaskID since the previous call to ResetTaskMaxStackUsage() for TaskID or to StartOS().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(LoggingTask){
  Os_StackSizeType StackUsages[MAXTASKS];
  ...
  StackUsages[0] = GetTaskMaxStackUsage(Task1);
  StackUsages[1] = GetTaskMaxStackUsage(Task2);
  StackUsages[2] = GetTaskMaxStackUsage(Task3);
  StackUsages[3] = GetTaskMaxStackUsage(Task4);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetStackUsage
Os_GetISRMaxStackUsage
Os_ResetISRMaxStackUsage
Os_ResetTaskMaxStackUsage

2.31 Os_GetVersionInfo

Get the version information for the OS

Syntax

```
void Os_GetVersionInfo(
    Std_VersionInfoType *versioninfo
)
```

Parameters

Name	Type	Mode	Description
versioninfo	Std_VersionInfoType	out	Pointer to variable used to get the OS Version information

Return Values

None.

Description

The content of the structure 'Std_VersionInfoType' is defined in Std_Types.h

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
Std_VersionInfoType ver;
Os_GetVersionInfo(&ver);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

None.

2.32 Os_IncrementCounter_<CounterID>

Increment a software counter.

Syntax

```
StatusType IncrementCounter_<CounterID>(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.

Description

This call has the same behavior as IncrementCounter(CounterID) but is customized for a named counter. This makes the call faster and more suitable for use in interrupt handlers.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
ISR(MillisecondTimerInterrupt){
  ...
  Os_IncrementCounter_MillisecondCounter();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

IncrementCounter
Os_AdvanceCounter
Os_AdvanceCounter_<CounterID>

2.33 Os_ResetISRMaxExecutionTime

Reset the maximum observed execution time for an ISR.

Syntax

```
StatusType Os_ResetISRMaxExecutionTime(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Name of the ISR to reset.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	ISRID is not a valid Category 2 ISR.

Description

Reset the maximum observed execution time for the Category 2 ISR identified by ISRID to zero.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(ProfilingTask){
  Os_StopwatchTickType ExecutionTimeLog[SAMPLES];
  ...
  ExecutionTimeLog[index++] = GetISRMaxExecutionTime(ISR1);
  ResetISRMaxExecutionTime(ISR1);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetExecutionTime
Os_GetTaskMaxExecutionTime
Os_GetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime

2.34 Os_ResetISRMaxStackUsage

Reset the maximum observed stack usage for an ISR.

Syntax

```
StatusType Os_ResetISRMaxStackUsage(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Name of the ISR to reset.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	ISRID is not a valid Category 2 ISR.

Description

Reset the maximum observed stack usage for ISRID to zero.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(ProfilingTask){
  Os_StackSizeType StackUsageLog[SAMPLES];
  ...
  StackUsageLog[index++] = GetISRMaxStackUsage(ISR1);
  ResetISRMaxStackUsage(ISR1);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetStackUsage
Os_GetTaskMaxStackUsage
Os_GetISRMaxStackUsage
Os_ResetTaskMaxStackUsage

2.35 Os_ResetTaskMaxExecutionTime

Reset the maximum observed execution time for a task.

Syntax

```
StatusType Os_ResetTaskMaxExecutionTime(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Name of the task to reset.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	TaskID is not a valid task.

Description

Reset the maximum observed execution time for TaskID to zero.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(ProfilingTask){
  Os_StopwatchTickType ExecutionTimeLog[SAMPLES];
  ...
  ExecutionTimeLog[index++] = GetTaskMaxExecutionTime(Task1);
  ResetTaskMaxExecutionTime(Task1);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetExecutionTime
Os_GetISRMaxExecutionTime
Os_GetTaskMaxExecutionTime
Os_ResetISRMaxExecutionTime

2.36 Os_ResetTaskMaxStackUsage

Reset the maximum observed stack usage for a task.

Syntax

```
StatusType Os_ResetTaskMaxStackUsage(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Name of the task to reset.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	TaskID is not a valid task.

Description

Reset the maximum observed stack usage for TaskID to zero.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
TASK(ProfilingTask){
  Os_StackSizeType StackUsageLog[SAMPLES];
  ...
  StackUsageLog[index++] = GetTaskMaxStackUsage(Task1);
  ResetTaskMaxStackUsage(Task1);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_GetStackUsage
Os_GetISRMaxStackUsage
Os_GetTaskMaxStackUsage
Os_ResetISRMaxStackUsage

2.37 Os_Restart

Restart the OS by jumping to a previously specified marker.

Syntax

```
StatusType Os_Restart(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OS_SYS_RESTART	all	The call was not made from the ShutdownHook.
E_OS_SYS_NO_RESTART	all	No restart point has been set.

Description

The call re-initializes any necessary context and branches to retstart point set by Os_SetRestartPoint. The call does not return to the calling context.

The restart point must occur before a call to StartOS(), so that all OS re-initialization re-occurs with the subsequent call to StartOS().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) ShutdownHook(StatusType Error){
  ...
  Os_Restart();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_SetRestartPoint
ShutdownOS
StartOS

2.38 Os_SetRestartPoint

Mark a location in code before StartOS() from where a restart of the OS can be made.

Syntax

```
StatusType Os_SetRestartPoint(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OS_SYS_NO_RESTART	all	The call was not made before StartOS.

Description

The call marks the location from which the code should resume following a call to Os_Restart(). The location must be outside of OS control, i.e. at a point before StartOS() was called. Making the call when a restart point is already sets the restart point to the new location.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
OS_MAIN() {
  ...
  Os_SetRestartPoint();
  ...
  StartOS(OSDEFAULTAPPMODE);

}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_Restart
ShutdownOS
StartOS

2.39 ReleaseResource

Release (unlock) a previously held resource to leave a critical section.

Syntax

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

Parameters

Name	Type	Mode	Description
ResID	ResourceType	in	The resource to release.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	ResID is not a valid resource.
E_OS_ACCESS	extended	Attempt to release a resource which has a lower ceiling priority than the configured priority of the calling task/ISR.

Description

ReleaseResource is the counterpart of GetResource and serves to quit a critical section in the code.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  GetResource(Outer);
    /* Outer Critical Section */
    ...
    GetResource(Inner);
      /* Inner Critical Section */
    ReleaseResource(Inner);
    ...
  ReleaseResource(Outer);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareResource
GetResource

2.40 ResumeAllInterrupts

Resume recognition of Category 1 and Category 2 interrupts.

Syntax

```
void ResumeAllInterrupts(void)
```

Return Values

None.

Description

This API call marks the end of a critical section that is protected from any maskable interrupt occuring. The critical section must have been entered using the SuspendAllInterrupts() call.

No API calls beside SuspendAllInterrupts()/ResumeAllInterrupts() pairs and SuspendOSInterrupts()/ResumeOSInterrupts() pairs are allowed within this critical section.

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

When calls to SuspendAllInterrupts() and ResumeAllInterrupts() are nested then the interrupt recognition status saved by the first call of SuspendAllInterrupts() is restored by the last call of the ResumeAllInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  SuspendAllInterrupts():
    /* Critical Section 1 */
    FunctionWithNestedCriticalSection();
  ResumeAllInterrupts():
  ...
}
void FunctionWithNestedCriticalSection(void) {
  ...
  SuspendAllInterrupts():
    /* Critical Section 2 */
  ResumeAllInterrupts():
  ...
```

```
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DisableAllInterrupts
EnableAllInterrupts
ResumeOSInterrupts
SuspendAllInterrupts
SuspendOSInterrupts

2.41 ResumeOSInterrupts

Resume recognition of Category 2 interrupts

Syntax

```
void ResumeOSInterrupts(void)
```

Return Values

None.

Description

This API call marks the end of a critical section that is protected from any Category 2 (OS level) interrupt occuring. The critical section must have been entered using the SuspendOSInterrupts() call.

No API calls beside SuspendAllInterrupts()/ResumeAllInterrupts() pairs and SuspendOSInterrupts()/ResumeOSInterrupts() pairs are allowed within this critical section.

Interrupt processing is restored to that in effect before the immediately prior SuspendOSInterrupts() call.

When calls to SuspendOSInterrupts() and ResumeOSInterrupts() are nested then the interrupt recognition status saved by the first call of SuspendOSInterrupts() is restored by the last call of the ResumeOSInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  SuspendOSInterrupts():
    /* Longer Critical Section */
    SuspendAllInterrupts();
      /* Shorter Critical Section  */
    ResumeAllInterrupts();
  ResumeOSInterrupts():
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✗
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DisableAllInterrupts
EnableAllInterrupts
ResumeAllInterrupts
SuspendAllInterrupts
SuspendOSInterrupts

2.42 Schedule

Forces the OS to check if a higher priority task can be run.

Syntax

```
StatusType Schedule(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_RESOURCE	extended	Calling task still holds resources.
E_OS_CALLEVEL	extended	Called at interrupt level.

Description

The call allows a non-preemptive task, or a task/ISR that uses an internal resource, to offer a preemption point.

Rescheduling occurs if:

1. The calling task is non-preemptive and a higher priority task has been activated while the calling task was in the running state.

2. The calling task/ISR shares an internal resource with a higher priority task/ISR and that higher priority task/ISR has been activated.

If no higher-priority task/ISR is in the ready state the calling task/ISR resumes.

This service has no influence on preemptive tasks or ISRs that do not use internal resources.

Note that allowing ISRs to share internal resources is an RTA-OS specific feature.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  CooperativeProcessA();
  Schedule();
  CooperativeProcessB();
  Schedule();
  CooperativeProcessC();
```

```
  Schedule();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareTask
TerminateTask
GetTaskState
GetTaskID

2.43 SetAbsAlarm

Set an alarm for an absolute counter value.

Syntax

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

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Name of the alarm to set.
start	TickType	in	Absolute tick value at which the alarm is first triggered.
cycle	TickType	in	Ticks before the alarm is triggered subsequently..

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_STATE	all	AlarmID already running.
E_OS_ID	extended	AlarmID is not a valid alarm.
E_OS_VALUE	extended	The value of start or cycle is outside the permitted range. 0 <= increment <= maxallowedvalue. cycle = 0 or mincycle <= cycle <= maxallowedvalue.

Description

This call starts an alarm running and sets the match value with the associated counter that triggers the alarm.

If cycle is equal to zero then the alarm will be triggered once only. If cycle is nonzero then the alarm will be triggered every cycle ticks after start.

When the alarm expires, the statically configured action (activate a task / set an event / run an alarm callback / increment a counter) occurs.

You must cancel an alarm if it is running before you can restart it with different values.

Note that if the value of start is less than or equal to the current counter value

then AlarmID will not be triggered until a full wrap of the underlying counter.

In particular, note that if an absolute alarm is set at startup with a start of zero - SetAbsAlarm(MyAlarm,0,x) - then the alarm will not be triggered until maxallowedvalue+1 ticks of the counter have elapsed.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  /* SingleShotAlarm at tick 42 */
  SetAbsAlarm(SingleShotAlarm, 42, 0);
  ...
  /* PeriodicAlarm at 10, 60, 110, 160,... */
  SetAbsAlarm(PeriodicAlarm, 10, 50);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

CancelAlarm
DeclareAlarm
GetAlarm
GetAlarmBase
SetRelAlarm

2.44 SetEvent

Set event(s) for a task.

Syntax

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

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Name of the Task to to set the Event for.
Mask	EventMaskType	in	A mask of events to set.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	extended	TaskID is not a valid task.
E_OS_ACCESS	extended	TaskID is not an extended task.
E_OS_STATE	extended	TaskID is in the suspended state.

Description

This API call sets events for task TaskID according to Mask.

If the task is waiting for any event in Event, it is immediately transferred to the ready state and re-scheduling can occur.

Multiple events can be set simultaneously by logically bitwise or-ing events.

Any unset events in the event mask remain unchanged.

Events cannot be set for extended tasks that are in the suspended state. In extended status this results in the error E_OS_STATE. In standard status, setting an event for a suspended task has no effect.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask) {
```

```
  ...
  /* Set a single event */
  SetEvent(MyExtendedTask, Event1);
  ...
  /* Set multiple events */
  SetEvent(MyOtherExtendedTask, Event1 | Event2 | Event3);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

ClearEvent
DeclareEvent
SetEvent
WaitEvent

2.45 SetRelAlarm

Set the number of counter ticks before an alarm expires.

Syntax

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

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Name of the alarm to set.
increment	TickType	in	Relative number of ticks before the alarm is first triggered.
cycle	TickType	in	Ticks before the alarm is triggered subsequently.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_STATE	all	AlarmID already running.
E_OS_ID	extended	AlarmID is not a valid alarm.
E_OS_VALUE	extended	The value of increment or cycle is outside the permitted range. 0 < increment <= maxallowedvalue. cycle =0 or mincycle <= cycle <= maxallowedvalue.

Description

This call starts an alarm running and sets the match value with the associated counter that triggers the alarm. The match value is equal to the current counter value plus the increment.

If cycle is equal to zero then the alarm will be triggered once only. If cycle is nonzero then the alarm will be triggered every cycle ticks after start.

When the alarm expires, the statically configured action (activate a task / set an event / run an alarm callback / increment a counter) occurs.

You must cancel an alarm if it is running before you can restart it with different values.

Care must be taken when the value of increment is small because the outcome of SetRelAlarm() can produce different results depending on whether the counter has ticked past the match 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 wrap of the underlying counter).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  /* SingleShotAlarm in Now+123 ticks */
  SetRelAlarm(SingleShotAlarm, 123, 0);
  ...
  /* PeriodicAlarm at Now+42, Now+142, Now+184... */
  SetRelAlarm(PeriodicAlarm, 42, 100);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

CancelAlarm
DeclareAlarm
GetAlarm
GetAlarmBase
SetAbsAlarm

2.46 ShutdownOS

Shutdown the operating system.

Syntax

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

Parameters

Name	Type	Mode	Description
Error	StatusType	in	The reason for the shutdown.

Return Values

None.

Description

This API causes the OS to shut down. Task scheduling, all interrupts, alarms and schedule tables are stopped.

PostTaskHook (if configured) is not called when ShutdownOS() occurs.

ShutdownHook is called (if configured) and is passed the Error argument as the OS shuts down.

If ShutdownHook() returns, then the operating system disables all interrupts and enter an endless loop.

ShutdownOS() can be called internally by the operating system in response to an unrecoverable error.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  if (ErrorCondition != E_OK) {
    ShutdownOS(ErrorCondition);
  }
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✓		
		ProtectionHook	✗		

See Also

Os_Restart
Os_SetRestartPoint
StartOS

2.47 StartOS

Start the operating system in a specified mode.

Syntax

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

Parameters

Name	Type	Mode	Description
Mode	AppModeType	in	The application mode to use for startup.

Return Values

None.

Description

StartOS() initializes all internal OS data structures and starts the OS in the specified Mode.

Any tasks that are autostarted in the specified Mode are set to the ready state.

Any alarms or schedule tables that are autostarted in the specified Mode are initialized appropriately.

Software counters are initialized to zero.

The Mode OSDEFAULTAPPMODE must always exist, but other names can be configured as needed.

StartOS() is only allowed outside the context of the OS. It has no effect if called while the OS is already running.

StartOS() does not return to the caller.

Restarting the OS can be achieved using Os_SetRestartPoint() to set a restart point before the call the StartOS() and jumping to the point using Os_Restart()..

If StartOS() is called with invalid preconditions, it may call ShutdownOS(E_OS_STATE). The preconditions are port-specific, so are documented in the port user guide. They may include issues such as the CPU being in the wrong mode, or the stack not being set up correctly.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
OS_MAIN() {
  /* Initialize target hardware before starting OS */
  StartOS(OSDEFAULTAPPMODE);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_Restart
Os_SetRestartPoint
ShutdownOS
Os_Cbk_Idle

2.48 StartScheduleTableAbs

Set the counter tick at which a schedule table starts.

Syntax

```
StatusType StartScheduleTableAbs(
    ScheduleTableType ScheduleTableID,
    TickType Start
)
```

Parameters

Name	Type	Mode	Description
ScheduleTableID	ScheduleTableType	in	Name of the schedule table to start.
Start	TickType	in	Absolute counter tick value at which the schedule table starts.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_STATE	all	ScheduleTableID already running.
E_OS_ID	extended	ScheduleTableID is not a valid ScheduleTable.
E_OS_VALUE	extended	Start > maxallowedvalue of the underlying counter.

Description

This call starts ScheduleTableID running when the counter reaches Start.

The first expiry point is processed at Start+InitialOffset ticks.

Note that if this gives a value less than or equal to the current counter value then the first expiry will not happen until a full wrap of the underlying counter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
TASK(MyTask){
  /* Start MyScheduleTable when the associated counter reaches
     100 */
  StartScheduleTableAbs(MyScheduleTable, 100);
```

```
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareScheduleTable
GetScheduleTableStatus
NextScheduleTable
StartScheduleTableRel
StopScheduleTable

2.49 StartScheduleTableRel

Set the number of counter ticks before a schedule table starts.

Syntax

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

Parameters

Name	Type	Mode	Description
ScheduleTableID	ScheduleTableType	in	Name of the schedule table to start.
Offset	TickType	in	Relative number of ticks before the schedule table starts.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_STATE	all	ScheduleTableID already running.
E_OS_ID	extended	ScheduleTableID is not a valid ScheduleTable.
E_OS_VALUE	extended	Offset == zero or Offset > maxallowedvalue - InitialOffset.

Description

This call starts ScheduleTableID running Offset ticks after the call was made.

The first expiry point is processed after Offset+InitialOffset ticks have elapsed.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
TASK(MyTask){
  ...
  /* Start MyScheduleTable at Now+42 ticks */
  StartScheduleTableRel(MyScheduleTable, 42);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareScheduleTable
GetScheduleTableStatus
NextScheduleTable
StartScheduleTableAbs
StopScheduleTable

2.50 StopScheduleTable

Stop a schedule table.

Syntax

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

Parameters

Name	Type	Mode	Description
ScheduleTableID	ScheduleTableType	in	Name of the schedule table to stop.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_NOFUNC	all	ScheduleTableID is not running.
E_OS_ID	extended	ScheduleTableID is not a valid ScheduleTable.

Description

This call stops ScheduleTableID immediately. A call to StartScheduleTableAbs or StartScheduleTableRel will re-start ScheduleTableID at the start.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
TASK(MyTask){
  ...
  StopScheduleTable(MyScheduleTable);
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareScheduleTable
GetScheduleTableStatus
NextScheduleTable
StartScheduleTableAbs
StartScheduleTableRel

2.51 SuspendAllInterrupts

Suspend recognition of Category 1 and Category 2 interrupts.

Syntax

```
void SuspendAllInterrupts(void)
```

Return Values

None.

Description

This API call marks the start of a critical section that is protected from any maskable Category 1 or Category 2 interrupt occuring. The critical section must be left with using the ResumeAllInterrupts() call.

No API calls beside SuspendAllInterrupts()/ResumeAllInterrupts() pairs and SuspendOSInterrupts()/ResumeOSInterrupts() pairs are allowed within this critical section.

The call saves the the current interrupt mask so that it can be restored later by the ResumeAllInterrupts() call.

When calls to SuspendAllInterrupts() and ResumeAllInterrupts() are nested then the interrupt recognition status saved by the first call of SuspendAllInterrupts() is restored by the last call of the ResumeAllInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  SuspendAllInterrupts();;
  ...
  ResumeAllInterrupts();;
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DisableAllInterrupts
EnableAllInterrupts
ResumeAllInterrupts
ResumeOSInterrupts
SuspendOSInterrupts

2.52 SuspendOSInterrupts

Suspend recognition of Category 2 interrupts.

Syntax

```
void SuspendOSInterrupts(void)
```

Return Values

None.

Description

This API call marks the start of a critical section that is protected from any Category 2 interrupt occuring. Category 1 interrupts may still occur. The critical section must be left using the ResumeOSInterrupts() call.

No API calls beside SuspendAllInterrupts()/ResumeAllInterrupts() pairs and SuspendOSInterrupts()/ResumeOSInterrupts() pairs are allowed within this critical section.

The call saves the the current interrupt mask so that it can be restored later by the ResumeOSInterrupts() call.

When calls to SuspendOSInterrupts() and ResumeOSInterrupts() are nested then the interrupt recognition status saved by the first call of SuspendOSInterrupts() is restored by the last call of the ResumeOSInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  SuspendOSInterrupts():
    /* Longer Critical Section */
    ...
    SuspendAllInterrupts();
    /* Shorter Critical Section  */
    ResumeAllInterrupts();
    ...
  ResumeOSInterrupts():
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✗
Category 1 ISR	✓	PostTaskHook	✓	TimeOverrunHook	✗
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✓		
		ProtectionHook	✓		

See Also

DisableAllInterrupts
EnableAllInterrupts
ResumeAllInterrupts
ResumeOSInterrupts
SuspendOSInterrupts

2.53 TerminateTask

Terminates the calling task

Syntax

```
StatusType TerminateTask(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_RESOURCE	extended	Calling task still holds resources.
E_OS_CALLEVEL	extended	Called at interrupt level.

Description

This call terminates the calling task. This transfers the calling task from the running state to the suspended state. The call does not return to the calling context if successful.

If the calling task has queued activations pending then the next instance of the task is automatically transferred into the ready state.

Internal resources are released automatically.

Standard or linked resources are also released automatically and this is reported as an error condition in extended status.

TerminateTask() always causes re-scheduling.

If the 'Fast Terminate' is enabled in Optimizations for RTA-OS then TerminateTask() must only be called from the task entry function and the return status should not be checked (ErrorHook, when configured, will be called if there is an error). This optimization saves memory and execution time. For further savings, you can actually omit the call to TerminateTask().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask){
  ...
  TerminateTask():
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareTask
TerminateTask
GetTaskState
GetTaskID

2.54 WaitEvent

Wait for one or more events.

Syntax

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

Parameters

Name	Type	Mode	Description
Mask	EventMaskType	in	The event(s) to be waited upon.

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ACCESS	extended	Not called from an extended task.
E_OS_CALLEVEL	extended	Called from interrupt level.
E_OS_RESOURCE	extended	The calling task holds a resource.

Description

Puts the calling task into the waiting state until one of the specified events is set.

If one or more of the events is already set, then the task remains in the running state.

The API call may cause re-scheduling to take place.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyExtendedTask){
  ...
  WaitEvent(Event1);
  /* Task resumes here when Event1 is set */
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

DeclareEvent
ClearEvent
GetEvent
SetEvent

3 RTA-OS3.0 Callbacks

3.1 Guide to Descriptions

Callbacks are code that is required by the OS but must be provided by the user. This section documents all the callbacks in RTA-OS3.0. The descriptions have the following structure:

Syntax

```
/* C function prototype for the callback */
ReturnValue NameOfCallback(Parameter Type, ...)
```

Parameters

A list of parameters for each callback and their mode:

in The parameter is passed in to the callback by the OS

out The parameter is passed out of the API callback by passing a reference (pointer) to the parameter into the call.

inout The parameter is passed into the callback and then (updated) and passed out.

Return Values

A description of the return value of the callback,

Description

A detailed description of the required functionality of the callback.

Portability

The portability of the call between OSEK OS, AUTOSAR OS, RTA-OS3.0 and RTA-TRACE.

Example Code

```
A C code listing showing how to implement the callback.
```

Required When

The configuration that means user code must implement the callback.

See Also

A list of related callbacks.

3.2 ErrorHook

Callback routine used for trapping errors resulting from incorrect use of the OS API.

Syntax

```
FUNC(void, OS_APPL_CODE)ErrorHook(
    StatusType Error
)
```

Parameters

Name	Type	Mode	Description
Error	StatusType	in	The type of the error that has occurred.

Return Values

None.

Description

This is called when an API call returns a StatusType not equal to E_OK. The StatusType is passed into ErrorHook().

Macros are provided for obtaining information about the source of the error ErrorHook(), but they are only available if the OS has been configured to generate them.

The macros should only be used within ErrorHook().

(1) The macro OSErrorGetServiceID() returns an OSServiceIdType that indicates the API that raised the error. The values take the form OSServiceId_xxx where xxx is the name of an API call. e.g. OSServiceId_ActivateTask.

(2) Macros of the form OSError_<APIName>_<ParameterName>() return the values of the parameters were passed to API. e.g. OSError_ActivateTask_TaskID()

ErrorHook runs at a OS level and will not be preempted by Tasks or Category 2 ISRs.

A sample ErrorHook can be generated automatically by rtaosgen. See the RTA-OS3.0 Reference Guide for further details.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) ErrorHook(StatusType Error){
  switch (Error){
    case E_OS_ID:
      /* Handle illegal identifier error */
      break;
    case E_OS_VALUE:
      /* Handle illegal value error */
      break;
    case E_OS_STATE:
      /* Handle illegal state error */
      break;
    default:
      /* Handle all other types of error */
      break;
  }
}
```

Required when

Required when the ErrorHook is configured.

3.3 Os_Cbk_Cancel_<CounterID>

Callback routine to cancel the interrupt from a hardware counter.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_Cbk_Cancel_<CounterID>(void)
```

Return Values

The call returns values of type TickType.

Description

The callback must prevent interrupts related to the hardware counter occurring.

The interrupt source should be disabled and any interrupt that has become pending while the callback was running should be cleared.

It is not required to stop the associated hardware from incrementing.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_Cancel_MyCounter(void){
  DISABLE_HW_COUNTER_INTERRUPT_SOURCE;
  CLEAR_HW_COUNTER_PENDING_INTERRUPT;
}
```

Required when

Required for each hardware counter configured.

See Also

Os_Cbk_Now_<CounterID>
Os_Cbk_Set_<CounterID>
Os_Cbk_State_<CounterID>

3.4 Os_Cbk_GetStopwatch

Callback routine to get the current value of a free-running counter.

Syntax

```
FUNC(Os_StopwatchTickType, OS_APPL_CODE) Os_Cbk_GetStopwatch(
    void)
```

Return Values

The call returns values of type Os_StopwatchTickType.

Description

Os_Cbk_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 and trace measurements.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(Os_StopwatchTickType, OS_APPL_CODE) Os_Cbk_GetStopwatch(
    void){
  return (Os_StopwatchTickType) HARDWARE_TIMER_CHANNEL;
}
```

Required when

The callback must be provided if time monitoring or tracing is configured in the OS.

See Also

Os_GetExecutionTime
Os_GetISRMaxExecutionTime
Os_GetTaskMaxExecutionTime
Os_ResetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime

3.5 Os_Cbk_Idle

Runs when the OS becomes idle.

Syntax

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_Idle(void)
```

Return Values

The call returns values of type boolean.

Description

Os_Cbk_Idle() is called when the OS first becomes idle after startup. Any autostarted tasks will have run before it gets called.

If Os_Cbk_Idle() exits with a return value TRUE then it will be called again immediately. If Os_Cbk_Idle() exits with a return value FALSE then it will not be called again and the OS will busy wait when there are no tasks or ISRs ready to run.

A default implementation is supplied in the library that returns FALSE.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_Idle(void) {
  sleep();
  return TRUE;
}
```

Required when

See Also

StartOS
ShutdownOS

3.6 Os_Cbk_Now_<CounterID>

Callback routine that returns the current tick value of the counter.

Syntax

```
FUNC(TickType, OS_APPL_CODE) Os_Cbk_Now_<CounterID>(void)
```

Return Values

The call returns values of type TickType.

Description

The callback must return the current value of hardware counter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(TickType, OS_APPL_CODE) Os_Cbk_Now_MyCounter(void){
  return (TickType) HW_COUNTER_NOW_VALUE;
}
```

Required when

Required for each hardware counter configured.

See Also

Os_Cbk_Cancel_<CounterID>
Os_Cbk_Set_<CounterID>
Os_Cbk_State_<CounterID>

3.7 Os_Cbk_RegSetRestore_<RegisterSetID>

Callback routine requiring that the context for register set <RegisterSetID> gets restored.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_RegSetRestore_<RegisterSetID>(
    Os_RegSetDepthType Depth
)
```

Return Values

None.

Description

This callback is provided so that the application can restore the current context for register set <RegisterSetID>.

Depth gives the position in the application-provided save buffer from which the context must be read. It ranges from zero to (OS_REGSET_<RegisterSetID>_SIZE - 1).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_REGSET_FP_SIZE
static fp_context_save_area fpsave[OS_REGSET_FP_SIZE];
FUNC(void, OS_APPL_CODE) Os_Cbk_RegSetRestore_FP(
   Os_RegSetDepthType Depth){
  ... = fpsave[Depth];
}
#endif /* OS_REGSET_FP_SIZE */
```

Required when

The callback must be provided if Register Set <RegisterSetID> exists and preemption may require its context to be restored.

See Also

OS_REGSET_<RegisterSetID>_SIZE
Os_Cbk_RegSetSave_<RegisterSetID>

3.8 Os_Cbk_RegSetSave_<RegisterSetID>

Callback routine requiring that the context for register set <RegisterSetID> gets saved.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_RegSetSave_<RegisterSetID>(
    Os_RegSetDepthType Depth
)
```

Return Values

None.

Description

This callback is provided so that the application can save the current context for register set <RegisterSetID>.

Depth gives the position in the application-provided save buffer into which the context must be stored. It ranges from zero to (OS_REGSET_<RegisterSetID>_SIZE - 1).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_REGSET_FP_SIZE
static fp_context_save_area fpsave[OS_REGSET_FP_SIZE];
FUNC(void, OS_APPL_CODE) Os_Cbk_RegSetSave_FP(
   Os_RegSetDepthType Depth){
  fpsave[Depth] = ...;
}
#endif /* OS_REGSET_FP_SIZE */
```

Required when

The callback must be provided if Register Set <RegisterSetID> exists and preemption may require its context to be saved.

See Also

OS_REGSET_<RegisterSetID>_SIZE
Os_Cbk_RegSetRestore_<RegisterSetID>

3.9 Os_Cbk_Set_<CounterID>

Callback routine to set the next match value for a hardware counter.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_Set_<CounterID>(
    TickType Match
)
```

Parameters

Name	Type	Mode	Description
Match	TickType	in	The next absolute match value.

Return Values

None.

Description

The callback must set up the hardware counter to raise the appropriate interrupt when its value reaches the new Match value.

Match is an absolute value at which the next counter action needs to be processed.

This is called from within Os_AdvanceCounter_<CounterID>() to set the match value appropriate for the next alarm or expiry point.

It can also be called from SetAbsAlarm() or SetRelAlarm() if the newly started alarm has to execute before the currently set time.

Care must be taken to cope with the following situations:

- Where intervals are short, it is possible for the hardware count to have already moved past the Match value at the point this get called. If so, it is important to ensure that the interrupt pending bit gets set in software.

- Where an alarm can be started with an interval shorter than one already set, the code must be able to reduce the match value and detect if this means that the hardware count has already passed this point.

The callback does not normally initialize the underlying hardware. This is noramlly done in initialization code before the OS is started.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_Set_MyCounter(TickType Match){
  /* Prevent match interrupts for maxallowedvalue+1 ticks*/
  HW_OUTPUT_COMPARE_VALUE = COUNTER - 1u;
  dismiss_interrupt();
  HW_OUTPUT_COMPARE = Match;
  enable_interrupt();
}
```

Required when

Required for each hardware counter configured.

See Also

Os_AdvanceCounter
SetAbsAlarm
SetRelAlarm
Os_Cbk_Cancel_<CounterID>
Os_Cbk_Now_<CounterID>
Os_Cbk_State_<CounterID>

3.10 Os_Cbk_StackOverrunHook

Callback routine to trap stack-related errors.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_StackOverrunHook(
    Os_StackSizeType Overrun,
    Os_StackOverrunType Reason
)
```

Parameters

Name	Type	Mode	Description
Overrun	Os_StackSizeType	in	The amount of the overrun.
Reason	Os_StackOverrunType	in	The cause of the overrun.

Return Values

None.

Description

This hook routine is called if:

(a) a stack allocation budget has been specified for a task/ISR and this budget has been exceeded.

(b) an ECC task failed to start because there was no space on the stack

(c) an ECC task failed to resume from wait because there was no space on the stack

(d) an ECC task failed to wait because it was using too much stack (and its state could not, therefore, be safely preserved)

GetTaskID() and GetISRID() can be used to determine which Task or ISR is involved.

A default version of the hook is present in the kernel that calls ShutdownOS() with the status E_OS_STACKFAULT. You can implement the callback within your application to override this behaviour.

Budget overruns are detected at preemption points (or when Os_GetStackUsage() is called) and are only be reported the first time that the overrun is first detected in a given run.

A budget overrun does not result in a Task/ISR being forcibly terminated. (Note that it is not permissible to call TerminateTask within the hook.)

ECC related overruns occur when lower priority tasks exceed their stack bud-

get, or when the stack preemption overheads are set to values that are too small.

An ECC overrun does result in the Task being forcibly terminated.

OS_BUDGET and OS_ECC_WAIT can only occur when Stack Monitoring is configured.

OS_ECC_START and OS_ECC_RESUME can occur independently of whether Stack Monitoring is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_StackOverrunHook(
    Os_StackSizeType Overrun, Os_StackOverrunType Reason) {
  switch (Reason) {
    case OS_BUDGET:
      /* The currently running task or ISR has exceeded its
         stack budget */
      break;
    case OS_ECC_START:
      /* An ECC task has failed to start because there is
         insufficient room on the stack */
      break;
    case OS_ECC_RESUME:
      /* An ECC task has failed to resume from wait because
         there is insufficient room on the stack */
      break;
    case OS_ECC_WAIT:
      /* An ECC task has failed to enter the waiting state
         because it is exceeding its stack budget */
      break;
  }
}
```

Required when

Optional when Stack Monitoring is configured and budgets are assigned, or when there are ECC tasks.

See Also

Os_GetStackUsage
Os_GetISRMaxStackUsage
Os_GetTaskMaxStackUsage
Os_ResetISRMaxStackUsage
Os_ResetTaskMaxStackUsage
GetISRID
GetTaskID

3.11 Os_Cbk_State_<CounterID>

Callback routine to read the current state of a hardware counter.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_State_<CounterID>(
    Os_CounterStatusRefType State
)
```

Parameters

Name	Type	Mode	Description
State	Os_CounterStatusRefType	out	The counter state.

Return Values

None.

Description

This function must update the counter status structure to indicate if it is running, whether a counter interrupt is pending, and how long the interval is to the next match.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_State_MyCounter(
   Os_CounterStatusRefType State) {
  State.Delay = HW_OUTPUT_COMPARE_VALUE - HW_COUNTER_NOW_VALUE;
  State.Pending = counter_interrupt_pending();
  State.Running = counter_interrupt_enabled();
}
```

Required when

Required for each hardware counter configured.

See Also

Os_Cbk_Cancel_<CounterID>
Os_Cbk_Now_<CounterID>
Os_Cbk_Set_<CounterID>

3.12 Os_Cbk_TimeOverrunHook

Callback routine to trap errors detected during time monitoring.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_TimeOverrunHook(
    Os_StopwatchTickType Overrun
)
```

Parameters

Name	Type	Mode	Description
Overrun	Os_StopwatchTickType	in	The amount of the overrun in stopwatch ticks.

Return Values

None.

Description

This hook routine is called if an execution budget has been specified for a task/ISR and the execution time has exceeded this budget.

Budget overruns are detected at preemption points or when the Task/ISR terminated. This hook is called once, when the overrun is first detected.

A budget overrun does not result in a Task/ISR being forcibly terminated. (Note that it is not permissible to call TerminateTask within the hook.)

GetTaskID() and GetISRID() can be used to determine which Task or ISR has overrun.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TimeOverrunHook(
   Os_StopwatchTickType Overrun) {
}
```

Required when

Required when Time Monitoring is configured and budgets are assigned.

See Also

Os_GetExecutionTime
Os_GetISRMaxExecutionTime
Os_GetTaskMaxExecutionTime
Os_ResetISRMaxExecutionTime
Os_ResetTaskMaxExecutionTime
GetISRID
GetTaskID

3.13 PostTaskHook

Callback routine called when context switching from a task.

Syntax

```
FUNC(void, OS_APPL_CODE) PostTaskHook(void)
```

Return Values

None.

Description

This hook routine is called by the operating system immediately before it leaves the running state.

This means it is safe to evaluate the TaskID.

The PostTaskHook is not called if a task is leaving the running state because the ShutdownOS() call has been made.

A sample PostTaskHook can be generated automatically by rtaosgen. See the RTA-OS3.0 Reference Guide for further details.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) PostTaskHook(void){
  TaskType LeavingTask;
  GetTaskID(&LeavingTask);
  if (LeavingTask == TaskA) {
    /* Do action for leaving A */
  } else if (LeavingTask == TaskB) {
    /* Do action for leaving B */
  }
  ...
}
```

Required when

Required when the PostTaskHook is configured.

See Also

PreTaskHook

3.14 PreTaskHook

Callback routine called when context switching into a task.

Syntax

```
FUNC(void, OS_APPL_CODE) PreTaskHook(void)
```

Return Values

None.

Description

This hook routine is called by the operating system immediately after it enters the running state but before the task itself begins to execute.

This means it is safe to evaluate the TaskID.

A sample PreTaskHook can be generated automatically by rtaosgen. See the RTA-OS3.0 Reference Guide for further details.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) PreTaskHook(void){
  TaskType EnteringTask;
  GetTaskID(&EnteringTask);
  if (EnteringTask == TaskA) {
    /* Do action for entering A */
  } else if (EnteringTask == TaskB) {
    /* Do action for entering B */
  }
  ...
}
```

Required when

Required when the PreTaskHook is configured.

See Also

PostTaskHook

3.15 ShutdownHook

Callback routine called during OS shutdown.

Syntax

```
FUNC(void, OS_APPL_CODE)ShutdownHook(
    StatusType Error
)
```

Parameters

Name	Type	Mode	Description
Error	StatusType	in	The reason for the shutdown.

Return Values

None.

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. The OS can be restarted from the ShutdownHook() using Os_Restart()

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) ShutdownHook(StatusType Error){
  if (Error == E_OS_STACKFAULT) {
    /* Attempt recovery by restart */
    Os_Restart();
    /* Never reach here... */
  } else if (Error == E_OK) {
    /* Normal shutdown prodecure */
  }
  ...
}
```

Required when

Required when the ShutdownHook is configured.

See Also

Os_Restart
StartupHook

3.16 StartupHook

Callback routine called during OS startup.

Syntax

```
FUNC(void, OS_APPL_CODE) StartupHook(void)
```

Return Values

None.

Description

If a StartupHook() is configured, this hook routine is called by the OS at the end of the OS initialization, but before the scheduler is running.

The application can start tasks, initialize device drivers and so on within StartupHook().

StartupHook() runs with Category2 ISRs disabled so it is safe to enable interrupt sources from the hook.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) StartupHook(void){
  /* Enable timer interrupt */
  CHANNEL0_CONTROL_REG |= ONE_MILLISECOND_TIMER;
  CHANNEL0_CONTROL_REG |= ENABLE;
}
```

Required when

Required when the StartupHook is configured.

See Also

ShutdownHook

4 RTA-OS3.0 Types

4.1 AlarmBaseRefType

A pointer to an object of AlarmBaseType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
AlarmBaseType AlarmBase;
AlarmBaseRefType AlarmBaseRef = &AlarmBase;
```

4.2 AlarmBaseType

Defines the configuration of a counter. The type is a C struct that contains the fields maxallowedvalue, ticksperbase and mincycle.

maxallowedvalue is the maximum allowed count value in ticks.

ticksperbase is the number of ticks required to reach a counter-specific (significant) unit.

mincycle is the smallest allowed value for the cycle-parameter of SetRelAlarm/SetAbsAlarm.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

All values are of type TickType.

Example

```
TickType         max,min,ticks;
AlarmBaseType    SomeAlarmBase;
AlarmBaseRefType PointerToSomeAlarmBase = &SomeAlarmBase;
max   = SomeAlarmBase.maxallowedvalue;
ticks = SomeAlarmBase.ticksperbase;
min   = SomeAlarmBase.mincycle;
```

4.3 AlarmType

The type of an Alarm.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
AlarmType SomeAlarm;
```

4.4 AppModeType

The type of an application mode.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

Symbolic names of the application modes declared at configuration time. (Must include OSDEFAULTAPPMODE)

Example

```
AppModeType SomeAppMode;
```

4.5 CounterType

The type of a Counter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
CounterType SomeCounter;
```

4.6 EventMaskRefType

A pointer to an object of EventMaskType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
EventMaskRefType SomeEventRef;
```

4.7 EventMaskType

The type of an event.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

Symbolic names of the EventMasks declared at configuration time.

Example

```
EventMaskType SomeEvent;
```

4.8 ISRRefType

A pointer to an object of ISRType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
ISRType SomeISR;
ISRRefType PointerToSomeISR = &SomeISR;
```

4.9 ISRType

The type of a ISR.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

The symbolic names of ISRs declared at configuration time.

Constants

```
INVALID_ISR
```

Example

```
ISRType SomeISR;
```

4.10 OSServiceIdType

The type of a OS API call. Used only in the ErrorHook(). The values take the form OSServiceId__APICallName_ where _APICallName_ represents the name of an API call (without any leading Os_).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
FUNC(void, OS_APPL_CODE) ErrorHook(StatusType Error){
  OSServiceIdType ServiceExecuting;
  ServiceExecuting = OSError_GetServiceID();
  switch ( ServiceExecuting ) {
    case OSServiceId_ActivateTask:
      ...
      break;
    case OSServiceId_CancelAlarm:
      ...
      break;
    case OSServiceId_ChainTask:
      ...
      break;
    ...
    default:
      ...
  }
}
```

4.11 Os_CounterStatusRefType

A pointer to an object of Os_CounterStatusType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_CounterStatusType MyHwCounterStatus;
do {
  Os_AdvanceCounter_MyHWCounter();
  Os_Cbk_State_MyHWCounter(&MyHwCounterStatus);
} while (MyHwCounterStatus.Running && MyHwCounterStatus.Pending
   );
```

4.12 Os_CounterStatusType

Defines the status of a hardware counter. The type is a C struct that contains the fields Running, Pending and Delay.

Running is TRUE only if the counter driver is running.

Pending is TRUE only if an expiry of an associated alarm and/or schedule table expiry point is pending.

Delay is a value that defines the number of ticks - relative to the last expiry - at which the next expiry is due. An Os_CounterStatusType.Delay value of zero represents maxallowedvalue+1 (the modulus) of the counter.

The Delay field is only valid when Running and Pending are TRUE.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_CounterStatusType CounterStatus;
```

4.13 Os_StackOverrunType

Enumerated type defining the reason for a stack overrun.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Values

OS_BUDGET
OS_ECC_START
OS_ECC_RESUME
OS_ECC_WAIT

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_StackOverrunHook(
   Os_StackSizeType Overrun, Os_StackOverrunType Reason) {
  switch (Reason) {
    case OS_BUDGET:
      /* The currently running task or ISR has exceeded its
         stack budget */
      break;
    case OS_ECC_START:
```

```
      /* An ECC task has failed to start because there is
         insufficient room on the stack */
      break;
    case OS_ECC_RESUME:
      /* An ECC task has failed to resume from wait because
         there is insufficient room on the stack */
      break;
    case OS_ECC_WAIT:
      /* An ECC task has failed to enter the waiting state
         because it is exceeding its stack budget */
      break;
  }
}
```

4.14 Os_StackSizeType

An unsigned value representing an amount of stack in bytes.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_StackSizeType stack_size;
stack_size = Os_GetStackSize(start_position, end_position);
```

4.15 Os_StackValueType

An unsigned value representing the position of the stack pointer (ESP).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_StackValueType start_position;
start_position = Os_GetStackValue();
```

4.16 Os_StopwatchTickType

Scalar representing a ticks of a stopwatch (time monitoring or protection) counter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
Os_StopwatchTickType Duration;
GetExecutionTime(&Duration);
```

4.17 PhysicalTimeType

Scalar representing a units of physical (wall clock) time.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
PhysicalTimeType Milliseconds = OS_TICKS2MS_MyCounter(42);
```

4.18 ResourceType

The type of a Resource.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

RES_SCHEDULER plus the symbolic names of Resources declared at configuration time.

Constants

RES_SCHEDULER

Example

```
ResourceType SomeResource;
```

4.19 ScheduleTableRefType

A pointer to an object of ScheduleTableType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
ScheduleTableType SomeScheduleTable;
ScheduleTableRefType PointerToSomeScheduleTable = &
    SomeScheduleTable;
```

4.20 ScheduleTableStatusRefType

A pointer to an object of ScheduleTableStatusType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
ScheduleTableStatusType SomeScheduleTableStatus;
GetScheduleTableStatus(&SomeScheduleTableStatus);
```

4.21 ScheduleTableStatusType

Enumerated type defining the runtime state of a schedule table.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

SCHEDULETABLE_STOPPED
SCHEDULETABLE_NEXT
SCHEDULETABLE_WAITING
SCHEDULETABLE_RUNNING
SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS

Example

```
ScheduleTableStatusType SomeScheduleTableStatus;
```

4.22 ScheduleTableType

The type of a ScheduleTable.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
ScheduleTableType SomeScheduleTable;
```

4.23 StatusType

Enumeration type defining the status of an API call.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

E_OK
E_OS_ACCESS
E_OS_CALLEVEL
E_OS_ID
E_OS_LIMIT
E_OS_NOFUNC
E_OS_RESOURCE
E_OS_STATE
E_OS_VALUE
E_OS_SERVICEID
E_OS_ILLEGAL_ADDRESS
E_OS_MISSINGEND
E_OS_DISABLEDINT
E_OS_STACKFAULT
E_OS_PROTECTION_MEMORY
E_OS_PROTECTION_TIME
E_OS_PROTECTION_ARRIVAL
E_OS_PROTECTION_LOCKED
E_OS_PROTECTION_EXCEPTION
E_OS_SYS_NO_RESTART
E_OS_SYS_RESTART
E_OS_SYS_OVERRUN

Example

```
StatusType ErrorCode;
ErrorCode = ActivateTask(MyTask);
```

4.24 Std_ReturnType

AUTOSAR's standard API service return type. This is NOT used by AUTOSAR OS. The type is an 8-bit unsigned integer whose top 6 bits may encode module-specific error codes.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

E_OK=0
E_NOT_OK=1

Example

```
Std_ReturnType ErrorCode;
Std_ReturnType ErrorMask = 0x03;
ErrorCode = Rte_Call_SomePort_SomeOperation(self, 42);
if ((ErrorCode & ErrorMask)== E_NOT_OK)  {
  /* call succeeded */
}
```

4.25 Std_VersionInfoType

A C struct whose fields contained AUTOSAR version information for a module. (Defined in Std_Types.h)

The fields are:

vendorID

moduleID

instanceID

sw_major_version

sw_minor_version

sw_patch_version

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

The field vendorID for ETAS is 11
The field moduleID for AUTOSAR OS is 1

Example

```
Std_VersionInfoType Version;
GetVersionInfo(&Version);
if (Version.vendorID == 11) {
  /* Make ETAS-specific API call */
  AdvanceCounter(HardwareCounter);
}
```

4.26 TaskRefType

A pointer to an object of TaskType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TaskType SomeTask;
TaskRefType TaskRef = &SomeTask;
```

4.27 TaskStateRefType

A pointer to an object of TaskStateType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TaskStateType TaskState;
TaskStateRefType TaskStateRef = &TaskState;
```

4.28 TaskStateType

Enumerated type defining the current state of a task.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

SUSPENDED
READY
WAITING
RUNNING

Example

```
TaskStateType TaskState;
GetTaskState(&TaskState);
```

4.29 TaskType

The type of a task.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Values

The symbolic names of tasks declared at configuration time.

Constants

`INVALID_TASK`

Example

```
TaskType SomeTask;
```

4.30 TickRefType

A pointer to an object of TickType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TickRefType SomeTick;
GetCounterValue(MyCounter,SomeTick);
```

4.31 TickType

Scalar representing a ticks of a counter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TickType StartTime = 42;
TickType NoRepeat = 0;
SetAbsAlarm(MyAlarm,StartTime,NoRepeat);
```

4.32 boolean

Addressable 8 bits only for use with TRUE/FALSE. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

0=FALSE
1=TRUE

Example

```
if (Condition == TRUE) {
  x = y;
}
```

4.33 float32

Single precision floating point number. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
float32 x;
```

4.34 float64

Double precision floating point number. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
float64 x;
```

4.35 sint16

Signed 16-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

-32768..32767

Example

```
sint16 x;
```

4.36 sint16_least

Signed integer at least 16-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least -32768..32767

Example

```
sint16_least x;
```

4.37 sint32

Signed 32-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

-2147483648..2147483647

Example

```
sint32 x;
```

4.38 sint32_least

Signed integer at least 32-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least -2147483648..2147483647

Example

```
sint32_least x;
```

4.39 sint8

Signed 8-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

-128..127

Example

```
sint8 x;
```

4.40 sint8_least

Signed integer at least 8-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least -128..127

Example

```
sint8_least x;
```

4.41 uint16

Unsigned 16-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

0..65535

Example

```
uint16 x;
```

4.42 uint16_least

Unsigned integer at least 16-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least 0..65535

Example

```
uint16_least x;
```

4.43 uint32

Unsigned 32-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

0..4294967295

Example

```
uint32 x;
```

4.44 uint32_least

Unsigned integer at least 32-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least 0..4294967295

Example

```
uint32_least x;
```

4.45 uint8

Unsigned 8-bit integer. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

0..255

Example

```
uint8 x;
```

4.46 uint8_least

Unsigned integer at least 8-bits wide. (Defined in Platform_Types.h)

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Values

At least 0..255

Example

```
uint8_least x;
```

5 RTA-OS3.0 Macros

5.1 ALARMCALLBACK

Declares an alarm callack. The only OS API calls that can be made in an alarm callback are SuspendAllInterrupts() and ResumeAllInterrupts().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
ALARMCALLBACK(MyCallback){...}
```

5.2 CAT1_ISR

Macro that should be used to create a Category 1 ISR entry function. This macro exists to help make your code portable between targets.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
CAT1_ISR(MyISR) {...}
```

5.3 DeclareAlarm

This is used to declare an alarm and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all Alarms in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareAlarm(MyAlarm);
```

5.4 DeclareCounter

This is used to declare a Counter and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all Counters in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareCounter(MyCounter);
```

5.5 DeclareEvent

This is used to declare an Event and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all Events in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareEvent(MyEvent);
```

5.6 DeclareISR

This is used to declare an ISR and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all ISRs in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareISR(MyISR);
```

5.7 DeclareResource

This is used to declare a Resource and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all Resources in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareResource(MyResource);
```

5.8 DeclareScheduleTable

This is used to declare a ScheduleTable and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all ScheduleTables in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareScheduleTable(MyScheduleTable);
```

5.9 DeclareTask

This is used to declare a Task and works similarly to external declaration of variables in C. You will not normally need to use this because RTA-OS automatically declares all Tasks in your configuration.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
DeclareTask(MyTask);
```

5.10 ISR

Macro that must be used to create a Category 2 ISR entry function.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
ISR(MyISR) {...}
```

5.11 OSCYCLEDURATION

Duration of an instruction cycle in nanoseconds.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
time_in_ns = CycleMeasurement * OSCYCLEDURATION;
```

5.12 OSCYCLESPERSECOND

The number of instruction cycles per second.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
time_in_secs = CycleMeasurement / OSCYCLESPERSECOND;
```

5.13 OSErrorGetServiceId

Returns the identifier of the service that generated an error.

Values are of OSServiceIdType.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
OSServiceIdType WhatServiceFailed = OSErrorGetServiceId();
```

5.14 OSMAXALLOWEDVALUE

Constant definition of the maximum possible value of the Counter called SystemCounter in ticks.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
SetAbsAlarm(MyAlarm,OSMAXALLOWEDVALUE,0)
```

5.15 OSMAXALLOWEDVALUE_<CounterID>

Constant definition of the maximum possible value of the Counter called CounterID in ticks.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
SetAbsAlarm(MyAlarm,OSMAXALLOWEDVALUE_SomeCounter,0)
```

5.16 OSMINCYCLE

Constant definition of the minimum number of ticks for a cyclic alarm on the Counter called SystemCounter.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
if (ComputedValue < OSMINCYCLE) {
    SetAbsAlarm(MyAlarm,42,OSMINCYCLE);
} else {
    SetAbsAlarm(MyAlarm,42,ComputedValue);
}
```

5.17 OSMINCYCLE_<CounterID>

Constant definition of the minimum number of ticks for a cyclic alarm on the Counter called CounterID.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
if (ComputedValue < OSMINCYCLE_SomeCounter) {
    SetAbsAlarm(MyAlarm,42,OSMINCYCLE_SomeCounter);
} else {
    SetAbsAlarm(MyAlarm,42,ComputedValue);
}
```

5.18 OSSWTICKDURATION

Duration of a stopwatch tick in nanoseconds.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
time_in_ns = StopwatchMeasurement * OSSWTICKDURATION;
```

5.19 OSSWTICKSPERSECOND

The number of stopwatch ticks per second.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
time_in_secs = CycleMeasurement / OSSWTICKSPERSECOND;
```

5.20 OSTICKDURATION

Duration of a tick of the Counter called SystemCounter in nanoseconds.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
uint32   RealTimeDeadline = 50000; /* 50 ms */
TickType Deadline = (TickType)RealTimeDeadline / OSTICKDURATION
    ;
SetRelAlarm(Timeout,Deadline,0);
```

5.21 OSTICKDURATION_<CounterID>

Duration of a tick of the Counter called CounterID in nanoseconds.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
uint32   RealTimeDeadline = 50000; /* 50 ms */
TickType Deadline = (TickType)RealTimeDeadline /
   OSTICKDURATION_SomeCounter;
SetRelAlarm(Timeout,Deadline,0);
```

5.22 OSTICKSPERBASE

Constant definition of the ticks per base setting of the Counter called SystemCounter in ticks.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

5.23 OSTICKSPERBASE_<CounterID>

Constant definition of the ticks per base setting of the Counter called CounterID in ticks.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

5.24 OS_EXTENDED_STATUS

Defined when extended status is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_EXTENDED_STATUS
CheckStatusType = ActivateTask(Task1);
if (CheckStatusType == E_OS_LIMIT) {
   /* Log an error */
}
#else
ActivateTask(Task1);
#endif
```

5.25 OS_MAIN

Declare the main program. Use of OS_MAIN() rather than main() is preferred for portable code, because different compilers have different requirements on the parameters and return type of main().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#include "Os.h"
OS_MAIN() {
  /* Initialize target hardware */
```

```
  StartOS(OSDEFAULTAPPMODE);
}
```

5.26 OS_NOAPPMODE

The value returned by GetActiveApplicationMode() when the OS is not running.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.27 OS_NUM_ALARMS

The number of alarms declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.28 OS_NUM_APPMODES

The number of AppModes declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.29 OS_NUM_COUNTERS

The number of counters declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.30 OS_NUM_EVENTS

The number of Events declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.31 OS_NUM_ISRS

The number of Category 2 ISRs declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.32 OS_NUM_RESOURCES

The number of resources declared (excludes internal).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.33 OS_NUM_SCHEDULETABLES

The number of schedule tables declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.34 OS_NUM_TASKS

The number of tasks declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

5.35 OS_REGSET_<RegisterSetID>_SIZE

This macro defines the size of the buffer needed to preserve Register Set <RegisterSetID> at run time. If no buffer is needed, then it is not declared. This can happen if no task/ISR that uses the register set can be preempted by another one that also uses it.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_REGSET_FP_SIZE
  fp_context_save_area fpsave[OS_REGSET_FP_SIZE];
#endif /* OS_REGSET_FP_SIZE */
```

5.36 OS_SCALABILITY_CLASS_1

Defined when AUTOSAR Scalability Class 1 is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_SCALABILITY_CLASS_1
ALARMCALLBACK(OnlyInSC1){
    ...
}
#endif
```

5.37 OS_SCALABILITY_CLASS_2

Defined when AUTOSAR Scalability Class 2 is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#if defined(OS_SCALABILITY_CLASS_2) || defined(
   OS_SCALABILITY_CLASS_4)
StartScheduleTableSynchron(Table);
#endif
```

5.38 OS_SCALABILITY_CLASS_3

Defined when AUTOSAR Scalability Class 3 is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#if defined(OS_SCALABILITY_CLASS_3) || defined(
   OS_SCALABILITY_CLASS_4)
FUNC(void, OS_APPL_CODE)
ErrorHook_MyApplication(StatusType Error){
 /* Handle OS-Application error */
}
#endif
```

5.39 OS_SCALABILITY_CLASS_4

Defined when AUTOSAR Scalability Class 4 is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#if defined(OS_SCALABILITY_CLASS_3) || defined(
   OS_SCALABILITY_CLASS_4)
FUNC(void, OS_APPL_CODE)
ErrorHook_MyApplication(StatusType Error){
 /* Handle OS-Application error */
}
#endif
```

5.40 OS_STACK_MONITORING

This macro is only defined if stack monitoring is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_Idle(void){
    #ifdef OS_STACK_MONITORING
        Os_StackSizeType Task1Stack, Task2Stack, Task3Stack;
        Task1Stack = Os_GetTaskMaxStackUsage(Task1);
        Task2Stack = Os_GetTaskMaxStackUsage(Task2);
        ...
        TaskNStack = Os_GetTaskMaxStackUsage(TaskN);
    #endif
    return TRUE;
}
```

5.41 OS_STANDARD_STATUS

Defined when standard status is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_STANDARD_STATUS
ActivateTask(Task1);
#else
CheckStatusType = ActivateTask(Task1);
if (CheckStatusType == E_OS_LIMIT) {
   /* Log an error */
}
#endif
```

5.42 OS_TICKS2<Unit>_<CounterID>(ticks)

Converts ticks on CounterID to Unit where Unit is: NS (nanosecond), MS (Millisecond), US (Microsecond), SEC(Second).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✓	✗

Example

```
time_in_ms = OS_TICKS2MS_SystemCounter(time);
```

5.43 OS_TIME_MONITORING

This macro is only defined if time monitoring is configured.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✗

Example

```
#ifdef OS_TIME_MONITORING
Os_StopwatchTickType start,end,function_duration;
start = Os_GetExecutionTime();
#endif
ThirdPartyFunction(x,y);
#ifdef OS_TIME_MONITORING
end = Os_GetExecutionTime();
function_duration = end - start;
#endif
```

5.44 TASK

Macro that must be used to create the task's entry function.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✓	✓	✗

Example

```
TASK(MyTask) {...}
```

6 RTA-TRACE API calls

6.1 Guide to Descriptions

All API calls have the following structure:

Syntax

```
/* C function prototype for the API call */
ReturnValue NameOfAPICall(Parameter Type, ...)
```

Parameters

A list of parameters for each API call and their mode:

in The parameter is passed in to the call

out The parameter is passed out of the API call by passing a reference (pointer) to the parameter into the call.

inout The parameter is passed into the call and then (updated) and passed out.

Return Values

Where API calls return a `StatusType` the values of the type returned and an indication of the reason for the error/warning are listed. The build column indicates whether the value is returned for both standard and extended status builds or for extended status build only.

Description

A detailed description of the behavior of the API call.

Portability

The RTA-OS3.0 API includes four classes of API calls:

OSEK OS calls are those specified by the OSEK OS standard. OSEK OS calls are portable to other implementations of OSEK OS and are portable to other implementations of AUTOSAR OS R3.0.

AUTOSAR OS calls are those specified by the AUTOSAR OS R3.0 standard. AUTOSAR OS calls are portable to other implementations of AUTOSAR OS R3.0. The calls are portable to OSEK OS only if the call is also an OSEK OS call.

RTA-TRACE calls are provided by RTA-OS3.0 for controlling the RTA-TRACE run-time profiling tool. These calls are only available when RTA-TRACE support has been configured.

RTA-OS3.0 calls include all those form the other three classes plus calls that provide extensions AUTOSAR OS functionality. These calls are unique to RTA-OS3.0 and are not portable to other implementations.

Example Code

```
A C code listing showing how to use the API calls
```

Calling Environment

The valid calling environment for the API call. A ✓ indicates that a call can be made in the indicated context. A ✗ indicates that the call cannot be made in the indicated context.

See Also

A list of related API calls.

6.2 Os_CheckTraceOutput

Checks for the presence of trace data.

Syntax

```
void Os_CheckTraceOutput(void)
```

Return Values

None.

Description

When tracing in free-running mode, this must be called regularly by the application. It is used to detect when the trace buffer has data to upload to RTA-TRACE.

It does not have to be called in Bursting or Triggering modes, though it is not harmful to do so.

It causes Os_Cbk_TraceCommDataReady() to be called when there is data to send.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_CheckTraceOutput();
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✗	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxEnd
Os_Cbk_TraceCommTxReady

6.3 Os_ClearTrigger

Clear all triggering conditions.

Syntax

```
void Os_ClearTrigger(void)
```

Return Values

None.

Description

This API call clears all trigger conditions that have been set using an Os_TriggerOnXXX() API.

Trace information will continue to be logged in the trace buffer, but no trace record will trigger the upload of the trace buffer to the host.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_ClearTrigger();
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartFreeRunningTrace
Os_SetTriggerWindow
Os_SetTraceRepeat

6.4 Os_DisableTraceCategories

Control which tracepoints are traced.

Syntax

```
void Os_DisableTraceCategories(
    Os_TraceCategoriesType CategoriesMask
)
```

Parameters

Name	Type	Mode	Description
CategoriesMask	Os_TraceCategoriesType	in	Mask of the trace categories to disable.

Return Values

None.

Description

Trace categories are used to filter whether tracepoints, task tracepoints and intervals get recorded and are typically used to control the volume of data that gets traced.

A category can be configured at build time to be active always, never or under run-time control. Categories that are under run-time control are enabled using Os_EnableTraceCategories and disabled using Os_DisableTraceCategories.

This call disables the specified run-time categories and therefore will inhibit the logging of all tracepoints, task tracepoints and intervals that are filtered by these categories.

Categories not listed in the call will be left in their current state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_DisableTraceCategories(DebugTracePoints | DataLogTracePoints
    ); /* Disable DebugTracePoints and DataLogTracePoints*/
Os_LogTracepoint(tpTest, DebugTracePoints); /* tpTest is not
    recorded: DebugTracePoints is disabled */
Os_LogTracepoint(tpTest, OS_TRACE_CATEGORY_ALWAYS); /* tpTest
    is recorded here */
```

```
Os_DisableTraceCategories(OS_TRACE_ALL_CATEGORIES); /* Disable
    all categories except for OS_TRACE_CATEGORY_ALWAYS */
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_EnableTraceCategories

6.5 Os_DisableTraceClasses

Control which types of objects are traced.

Syntax

```
void Os_DisableTraceClasses(
    Os_TraceClassesType ClassMask
)
```

Parameters

Name	Type	Mode	Description
ClassMask	Os_TraceClassesType	in	Mask of the trace classes to disable.

Return Values

None.

Description

Trace classes are used to filter whether complete types of trace events get recorded. They are typically used to control the volume of data that gets traced.

Trace classes can be configured at build time to be active always, never or under run-time control. Classes that are under run-time control are enabled using Os_EnableTraceClasses and disabled using Os_DisableTraceClasses.

This call disables the specified run-time classes and therefore will inhibit the tracing of events that are filtered by these classes.

Classes not listed in the call will be left in their current state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_DisableTraceClasses(OS_TRACE_TRACEPOINT_CLASS);
Os_LogTracepoint(tpTest, OS_TRACE_ALL_CATEGORIES);  /* Will not
    get recorded */
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_EnableTraceClasses

6.6 Os_EnableTraceCategories

Control which tracepoints are traced.

Syntax

```
void Os_EnableTraceCategories(
    Os_TraceCategoriesType CategoriesMask
)
```

Parameters

Name	Type	Mode	Description
CategoriesMask	Os_TraceCategoriesType	in	Mask of the trace categories to enable.

Return Values

None.

Description

Trace categories are used to filter whether tracepoints, task tracepoints and intervals get recorded and are typically used to control the volume of data that gets traced.

A category can be configured at build time to be active always, never or under run-time control. Categories that are under run-time control are enabled using Os_EnableTraceCategories and disabled using Os_DisableTraceCategories.

This call enables the specified run-time categories.

Categories not listed in the call will be left in their current state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_EnableTraceCategories(DebugTracePoints | DataLogTracePoints)
    ;
Os_LogTracepoint(tpTest, DebugTracePoints); /* tpTest is
    recorded */
Os_LogTracepoint(tpTest, FunctionProfileTracePoints); /* tpTest
     is not recorded - FunctionProfileTracePoints not enabled */
Os_LogTracepoint(tpTest, OS_TRACE_ALL_CATEGORIES); /* tpTest is
     recorded */
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_DisableTraceCategories

6.7 Os_EnableTraceClasses

Control which types of objects are traced.

Syntax

```
void Os_EnableTraceClasses(
    Os_TraceClassesType ClassMask
)
```

Parameters

Name	Type	Mode	Description
ClassMask	Os_TraceClassesType	in	Mask of the trace classes to enable.

Return Values

None.

Description

Trace classes are used to filter whether complete types of trace events get recorded. They are typically used to control the volume of data that gets traced.

Trace classes can be configured at build time to be active always, never or under run-time control. Classes that are under run-time control are enabled using Os_EnableTraceClasses and disabled using Os_DisableTraceClasses.

This call enables the specified run-time classes.

Classes not listed in the call will be left in their current state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_EnableTraceClasses(OS_TRACE_TRACEPOINT_CLASS);
Os_LogTracepoint(tpTest, OS_TRACE_ALL_CATEGORIES);  /* Will get
    recorded */
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_DisableTraceClasses

6.8 Os_LogCat1ISREnd

Log the end of a Category 1 ISR.

Syntax

```
void Os_LogCat1ISREnd(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Category 1 ISR identifier.

Return Values

None.

Description

This call marks the end of a Category 1 ISR. This type of ISR is not controlled by the operating system so no automatic tracing of it can occur. If Category 1 ISRs need to be logged then it is necessary to do this manually using this call.

This event is only logged if the OS_TRACE_TASKS_AND_ISRS_CLASS trace class is active.

Take care to ensure that both the start and end of the Category 1 ISR logged, otherwise the resulting trace will be incorrect.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
CAT1_ISR(Category1Handler) {
  Os_LogCat1ISRStart(Category1Handler);
  ...
  Os_LogCat1ISREnd(Category1Handler);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✓	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_LogCat1ISRStart

6.9 Os_LogCat1ISRStart

Log the start of a Category 1 ISR.

Syntax

```
void Os_LogCat1ISRStart(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Category 1 ISR identifier.

Return Values

None.

Description

This call marks the start of a Category 1 ISR. This type of ISR is not controlled by the operating system so no automatic tracing of it can occur. If Category 1 ISRs need to be logged then it is necessary to do this manually using this call.

This event is only logged if the OS_TRACE_TASKS_AND_ISRS_CLASS trace class is active.

Take care to ensure that both the start and end of the Category 1 ISR are logged, otherwise the resulting trace will be incorrect.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
CAT1_ISR(Category1Handler) {
  Os_LogCat1ISRStart(Category1Handler);
  ...
  Os_LogCat1ISREnd(Category1Handler);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✗	PreTaskHook	✗	StackOverrunHook	✗
Category 1 ISR	✓	PostTaskHook	✗	TimeOverrunHook	✗
Category 2 ISR	✗	StartupTaskHook	✗		
		ShutdownHook	✗		
		ErrorHook	✗		
		ProtectionHook	✗		

See Also

Os_LogCat1ISREnd

6.10 Os_LogCriticalExecutionEnd

Log the completion of a critical execution event.

Syntax

```
void Os_LogCriticalExecutionEnd(
    Os_TraceInfoType CriticalExecutionID
)
```

Parameters

Name	Type	Mode	Description
CriticalExecutionID	Os_TraceInfoType	in	Critical execution profile identifier.

Return Values

None.

Description

Logs the end of a critical point of execution in the trace buffer. This is typically used to indicate that a task/ISR has completed a time-critical section of code. This might be needed if the deadline that needs to be met by the task/ISR occurs before the end of the task/ISR.

CriticalExecutionID is only logged if the OS_TRACE_TASKS_AND_ISRS_CLASS class is active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
TASK(MyTask){
  ...
  ReadSensor(X);
  Os_LogCriticalExecutionEnd(SensorRead);
  ...
  WriteActuator(Y);
  Os_LogCriticalExecutionEnd(SensorRead);
  ...
  TerminateTask();
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.11 Os_LogIntervalEnd

Log the end of a measurement interval.

Syntax

```
void Os_LogIntervalEnd(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the end of an interval in the trace buffer.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStart(EndToEndTime, SystemLoggingCategory);
...
Os_LogIntervalEnd(EndToEndTime, SystemLoggingCategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEndValue
Os_LogIntervalEndData
Os_LogIntervalStart
Os_LogIntervalStartValue
Os_LogIntervalStartData

6.12 Os_LogIntervalEndData

Log the end of a measurement interval together with associated data.

Syntax

```
void Os_LogIntervalEndData(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceDataPtrType DataPtr,
    Os_TraceDataLengthType Length,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
DataPtr	Os_TraceDataPtrType	in	A pointer to the start address of the data block to log.
Length	Os_TraceDataLengthType	in	The length of the data block in bytes.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the end of an interval in the trace buffer and associate some data with it.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStart(EndToEndTime, SystemLoggingCategory);
...
Os_LogIntervalEndData(EndToEndTime, &DataBlock, 4,
   SystemLoggingCategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEnd
Os_LogIntervalEndValue
Os_LogIntervalStart
Os_LogIntervalStartValue
Os_LogIntervalStartData

6.13 Os_LogIntervalEndValue

Log the end of a measurement interval together with an associated value.

Syntax

```
void Os_LogIntervalEndValue(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceValueType Value,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
Value	Os_TraceValueType	in	Numerical value to be logged with the interval.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the end of an interval in the trace buffer and associate a value with it.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStart(EndToEndTime, SystemLoggingCategory);
...
Os_LogIntervalEndValue(EndToEndTime, 42, SystemLoggingCategory)
    ;
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEnd
Os_LogIntervalEndData
Os_LogIntervalEndValue
Os_LogIntervalStartValue
Os_LogIntervalStartData

6.14 Os_LogIntervalStart

Log the start of a measurement interval.

Syntax

```
void Os_LogIntervalStart(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the start of an interval in the trace buffer.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStart(EndToEndTime, SystemLoggingCategory);
...
Os_LogIntervalEnd(EndToEndTime, SystemLoggingCategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEnd
Os_LogIntervalEndData
Os_LogIntervalEndValue
Os_LogIntervalStartData
Os_LogIntervalStartValue

6.15 Os_LogIntervalStartData

Log the start of a measurement interval together with associated data.

Syntax

```
void Os_LogIntervalStartData(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceDataPtrType DataPtr,
    Os_TraceDataLengthType Length,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
DataPtr	Os_TraceDataPtrType	in	A pointer to the start address of the data block to log.
Length	Os_TraceDataLengthType	in	The length of the data block in bytes.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the start of an interval in the trace buffer and associate some data with it.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStartData(EndToEndTime, &DataBlock, 4,
   SystemLoggingCategory);
...
Os_LogIntervalEnd(EndToEndTimeSystemLoggingCategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEnd
Os_LogIntervalEndData
Os_LogIntervalEndValue
Os_LogIntervalStart
Os_LogIntervalStartValue

6.16 Os_LogIntervalStartValue

Log the start of a measurement interval together with an associated value.

Syntax

```
void Os_LogIntervalStartValue(
    Os_TraceIntervalIDType IntervalID,
    Os_TraceValueType Value,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Interval Identifier.
Value	Os_TraceValueType	in	Numerical value to be logged with the interval.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the start of an interval in the trace buffer and associate a value with it.

The interval is only logged if the OS_TRACE_INTERVAL_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogIntervalStartValue(EndToEndTime, 42,
   SystemLoggingCategory);
...
Os_LogIntervalEnd(EndToEndTime, SystemLoggingCategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogIntervalEnd
Os_LogIntervalEndData
Os_LogIntervalEndValue
Os_LogIntervalStart
Os_LogIntervalStartData

6.17 Os_LogProfileStart

Log the start of a new execution profile.

Syntax

```
void Os_LogProfileStart(
    Os_TraceInfoType ProfileID
)
```

Parameters

Name	Type	Mode	Description
ProfileID	Os_TraceInfoType	in	Profile Identifier.

Return Values

None.

Description

Logs which execution profile is active in the trace buffer. Execution profiles can be used to identify which route is taken through a Task or ISR when this depends on external conditions.

The profile is only recorded if the OS_TRACE_TASKS_AND_ISRS_CLASS class is active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
TASK(MyTask){
  if (some_condition()) {
    Os_LogProfileStart(TrueRoute);
    ...
  } else {
    Os_LogProfileStart(FalseRoute);
    ...
  }
  TerminateTask();
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.18 Os_LogTaskTracepoint

Log a tracepoint in the specified categories.

Syntax

```
void Os_LogTaskTracepoint(
    Os_TraceTracepointIDType TaskTracepointID,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TaskTracepointID	Os_TraceTracepointIDType	in	Task Tracepoint Identifier.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the task tracepoint event in the trace buffer.

TaskTracepointID is recorded only if the OS_TRACE_TASK_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTaskTracepoint(MyTaskTracePoint, ACategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointData
Os_LogTracepointValue
Os_LogTaskTracepointData
Os_LogTaskTracepointValue

6.19 Os_LogTaskTracepointData

Log a tracepoint in the specified categories together with associated data.

Syntax

```
void Os_LogTaskTracepointData(
    Os_TraceTracepointIDType TracepointID,
    Os_TraceDataPtrType DataPtr,
    Os_TraceDataLengthType Length,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Tracepoint Identifier.
DataPtr	Os_TraceDataPtrType	in	A pointer to the start address of the data block to log.
Length	Os_TraceDataLengthType	in	The length of the data block in bytes.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the task tracepoint event in the trace buffer and associate some data with it.

TaskTracepointID is recorded only if the OS_TRACE_TASK_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTaskTracepointData(MyTracePoint, &DataBlock, 4, ACategory
    );
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointValue
Os_LogTracepointData
Os_LogTaskTracepoint
Os_LogTaskTracepointValue

6.20 Os_LogTaskTracepointValue

Log a tracepoint in the specified categories together with an associated value.

Syntax

```
void Os_LogTaskTracepointValue(
    Os_TraceTracepointIDType TracepointID,
    Os_TraceValueType Value,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Tracepoint Identifier.
Value	Os_TraceValueType	in	Numerical value to be logged with the tracepoint.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the task tracepoint event in the trace buffer and associate a value with it.

TaskTracepointID is recorded only if the OS_TRACE_TASK_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTaskTracepointValue(MyTracePoint, 99, ACategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointData
Os_LogTracepointValue
Os_LogTaskTracepoint
Os_LogTaskTracepointData

6.21 Os_LogTracepoint

Log a tracepoint in the specified categories.

Syntax

```
void Os_LogTracepoint(
    Os_TraceTracepointIDType TracepointID,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Tracepoint Identifier.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the tracepoint event in the trace buffer.

TracepointID is recorded only if the OS_TRACE_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTracepoint(MyTracepoint, ACategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointData
Os_LogTracepointValue
Os_LogTracepointData
Os_LogTracepointValue

6.22 Os_LogTracepointData

Log a tracepoint in the specified categories together with associated data.

Syntax

```
void Os_LogTracepointData(
    Os_TraceTracepointIDType TracepointID,
    Os_TraceDataPtrType DataPtr,
    Os_TraceDataLengthType Length,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Tracepoint Identifier.
DataPtr	Os_TraceDataPtrType	in	A pointer to the start address of the data block to log.
Length	Os_TraceDataLengthType	in	The length of the data block in bytes.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the tracepoint event in the trace buffer and associate some data with it.

TracepointID is recorded only if the OS_TRACE_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTracepointData(MyTracePoint, &DataBlock, 4, ACategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointValue
Os_LogTracepointData
Os_LogTracepoint
Os_LogTracepointValue

6.23 Os_LogTracepointValue

Log a tracepoint in the specified categories together with an associated value.

Syntax

```
void Os_LogTracepointValue(
    Os_TraceTracepointIDType TracepointID,
    Os_TraceValueType Value,
    Os_TraceCategoriesType CategoryMask
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Tracepoint Identifier.
Value	Os_TraceValueType	in	Numerical value to be logged with the tracepoint.
CategoryMask	Os_TraceCategoriesType	in	A category mask.

Return Values

None.

Description

Log the tracepoint event in the trace buffer and associate a value with it.

TracepointID is recorded only if the OS_TRACE_TRACEPOINT_CLASS class is active and one or more of the categories in CategoryMask are active.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_LogTracepointValue(MyTracePoint, 99, ACategory);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_LogTracepoint
Os_LogTracepointData
Os_LogTracepointValue
Os_LogTracepoint
Os_LogTracepointData

6.24 Os_SetTraceRepeat

Control whether trace repeats or not.

Syntax

```
void Os_SetTraceRepeat(
    boolean Repeat
)
```

Parameters

Name	Type	Mode	Description
Repeat	boolean	in	Control whether bursting/triggering traces repeat.

Return Values

None.

Description

When TRUE, bursting and triggering trace modes automatically restart once the most recent trace content has been transmitted from the trace buffer to the RTA-TRACE client.

The API has no effect in free-running trace mode.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_SetTraceRepeat(TRUE);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartTriggeringTrace

6.25 Os_SetTriggerWindow

Set the size of the trace buffer window to be uploaded in triggering mode.

Syntax

```
void Os_SetTriggerWindow(
    Os_TraceIndexType Before,
    Os_TraceIndexType After
)
```

Parameters

Name	Type	Mode	Description
Before	Os_TraceIndexType	in	Number of records to be recorded before the trigger event.
After	Os_TraceIndexType	in	Number of records to record after the trigger event.

Return Values

None.

Description

This call sets the number of records to be recorded before and after a trigger event.

When the trigger occurs, tracing events continue to be logged until After trace records have been written to the trace buffer, and the data is then uploaded.

The total number of records uploaded (Before + After) is limited by the size of the trace buffer.

Note that a trace event that contains data values may required multiple records to be written to the trace buffer. This means that the number of complete events seen before or after the trigger point may be less than the number of records requested.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) StartupHook(){
  ...
  Os_SetTriggerWindow(100,50);
  Os_StartTriggeringTrace();
```

```
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartFreeRunningTrace

6.26 Os_StartBurstingTrace

Starts tracing in bursting mode.

Syntax

```
void Os_StartBurstingTrace(void)
```

Return Values

None.

Description

Bursting trace mode logs trace information into the trace buffer until the buffer is full. When the trace buffer is full, tracing stops and data transfer begins. No attempt is made to upload data to the host until the trace buffer has filled.

Where Os_SetTraceRepeat() has been used to enable repeated bursting traces, tracing resumes once the buffer is empty (i.e. once data transfer is complete).

The trace buffer is cleared and tracing restarts again if this call is made whilst tracing.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) StartupHook(){
  ...
  Os_StartBurstingTrace();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_SetTraceRepeat
Os_StartFreeRunningTrace
Os_StartTriggeringTrace

6.27 Os_StartFreeRunningTrace

Starts tracing in free-running mode.

Syntax

```
void Os_StartFreeRunningTrace(void)
```

Return Values

None.

Description

Free running trace mode logs trace information while there is space in the trace buffer. Data is uploaded to the host from the buffer as soon as it is available, concurrently with capture.

If the trace buffer becomes full, logging of trace data is suspended until there is space in the buffer. When space in the buffer is available again, tracing resumes. The buffer might become full if the communications link is too slow for the desired volume of trace data.

The trace buffer is cleared and tracing restarts again if this call is made whilst tracing.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) StartupHook(){
  ...
  Os_StartFreeRunningTrace();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartTriggeringTrace

6.28 Os_StartTriggeringTrace

Starts tracing in triggering mode.

Syntax

```
void Os_StartTriggeringTrace(void)
```

Return Values

None.

Description

Triggering trace mode logs trace information into the buffer continously, waiting for a trigger condition. If the buffer overflows, then new trace information overwrites existing information.

A pre- and post-trigger number of buffer records must be specified using Os_SetTriggerWindow() so that only the set of events before and after the trigger event can be seen. Unpredictable behavior may occur if the trigger window is not set.

Trigger events are set using the Os_TriggerOnXXX() APIs.

When a triggering event occurs (for example, when a task starts executing), data collection continues until post-trigger number of trace records are logged. Data transfer to the host then begins.

Tracing resumes after the data transfer completes if Os_SetTraceRepeat() permits this.

The trace buffer is cleared and tracing restarts again if this call is made whilst tracing.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) StartupHook(){
  ...
  Os_SetTriggerWindow(100,50);
  Os_StartTriggeringTrace();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartFreeRunningTrace
Os_SetTriggerWindow
Os_SetTraceRepeat

6.29 Os_StopTrace

Stops tracing.

Syntax

```
void Os_StopTrace(void)
```

Return Values

None.

Description

Stops data logging to the trace buffer. Any data remaining in the trace buffer is uploaded to the host.

Note that the call does not stop the the data link.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_StopTrace();
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_StartBurstingTrace
Os_StartFreeRunningTrace
Os_StartTriggeringTrace

6.30 Os_TraceCommInit

Initializes external communication support for tracing.

Syntax

```
Os_TraceStatusType Os_TraceCommInit(void)
```

Return Values

The call returns values of type Os_TraceStatusType.

Description

This function is used to initialize a trace communications link. It should not be used if you use a debugger link to extract trace data.

It calls the callback Os_Cbk_TraceCommInitTarget() to initialize the appropriate target hardware and its return value indicates the return value from the call to Os_Cbk_TraceCommInitTarget().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) StartupHook(){
  ...
  Os_TraceCommInit();
  Os_StartFreeRunningTrace();
  ...
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_Cbk_TraceCommInitTarget

6.31 Os_TraceDumpAsync

Uses an asynchronous communication to upload trace data in a single operation.

Syntax

```
void Os_TraceDumpAsync(
    Os_AsyncPushCallbackType fn
)
```

Return Values

None.

Description

This API is normally called in response to Os_Cbk_TraceCommDataReady(). It gets passed a reference to a function that can transmit a single character. It will call this function for each character that needs to be transmitted before returning to the caller.

An appropriate asynchronous serial device must be available and previously initialized. A typical serial link might be set to 115200bps, 8 data bits, no parity and 1 stop bit.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
extern FUNC(void, OS_APPL_CODE) push_async_io(uint8 val) {
  while(!async_tx_ready) {/* wait for room */}
  async_transmit(val)  ;
}
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommDataReady(void) {
  Os_TraceDumpAsync(push_async_io);
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_Cbk_TraceCommDataReady

6.32 Os_TriggerNow

Trigger upload of the trace buffer.

Syntax

```
void Os_TriggerNow(void)
```

Return Values

None.

Description

This API call forces a trigger condition to occur. This will cause the trace buffer to be uploaded, regardless of any other trigger conditions.

The call does not modify the state of the trigger conditions.

The call only has an effect in triggering trace mode.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerNow();
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.33 Os_TriggerOnActivation

Trigger when a task is activated.

Syntax

```
void Os_TriggerOnActivation(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Identifier of the task to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when specified task is activated.

TaskID can be set to OS_TRIGGER_ANY, in which case activation of any task will cause the trigger to occur.

The trigger will occur when a task is activated through ActivateTask, StartOS, Alarms or ScheduleTables.

Note that ChainTask(TaskID) does not cause an activation trigger; see Os_TriggerOnChain().

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnActivation(InterestingTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnChain

6.34 Os_TriggerOnAdvanceCounter

Trigger when a counter is advanced.

Syntax

```
void Os_TriggerOnAdvanceCounter(
    CounterType CounterID
)
```

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	Identifier of the hardware counter that triggers on advance.

Return Values

None.

Description

Causes a trace trigger to occur when a specified hardware counter is advanced.

CounterID can be set to OS_TRIGGER_ANY, in which case advancing any counter will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnAdvanceCounter(HWCounter);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnIncrementCounter

6.35 Os_TriggerOnAlarmExpiry

Trigger when an alarm expires.

Syntax

```
void Os_TriggerOnAlarmExpiry(
    AlarmType AlarmID
)
```

Parameters

Name	Type	Mode	Description
AlarmID	AlarmType	in	Identifier of the alarm.

Return Values

None.

Description

Causes a trace trigger to occur when a specified alarm expires.

AlarmID can be set to OS_TRIGGER_ANY, in which case any alarm expiry or *expiry point* will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnAlarmExpiry(Alarm_10ms);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.36 Os_TriggerOnCat1ISRStart

Trigger when a Category 1 ISR starts.

Syntax

```
void Os_TriggerOnCat1ISRStart(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Identifier of the Category 1 ISR to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified Category 1 ISR starts running.

ISRID can be set to OS_TRIGGER_ANY, in which case any such ISR will cause the trigger to occur.

Note that Category 1 ISRs are not controlled by RTA-OS, so you are responsible for calling Os_LogCat1ISRStart() at the beginning of your interrupt handler.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnCat1ISRStart(InterestingCat1ISR);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnCat1ISRStop
Os_LogCat1ISREnd

6.37 Os_TriggerOnCat1ISRStop

Trigger when a Category 1 ISR stops.

Syntax

```
void Os_TriggerOnCat1ISRStop(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Identifier of the Category 1 ISR to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified Category 1 ISR stops running.

ISRID can be set to OS_TRIGGER_ANY, in which case any such ISR will cause the trigger to occur.

Note that Category 1 ISRs are not controlled by RTA-OS, so you are responsible for calling Os_LogCat1ISREnd() at the end of your interrupt handler.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnCat1ISRStop(InterestingCat1ISR);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnCat1ISRStart
Os_LogCat1ISRStart
Os_LogCat1ISREnd

6.38 Os_TriggerOnCat2ISRStart

Trigger when a Category 2 ISR starts.

Syntax

```
void Os_TriggerOnCat2ISRStart(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Identifier of the Category 2 ISR to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified Category 2 ISR starts running.

ISRID can be set to OS_TRIGGER_ANY, in which case any such ISR will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnCat2ISRStart(InterestingCat2ISR);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnCat2ISRStop

6.39 Os_TriggerOnCat2ISRStop

Trigger when a Category 2 ISR stops.

Syntax

```
void Os_TriggerOnCat2ISRStop(
    ISRType ISRID
)
```

Parameters

Name	Type	Mode	Description
ISRID	ISRType	in	Identifier of the Category 2 ISR to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified Category 2 ISR stops running.

ISRID can be set to OS_TRIGGER_ANY, in which case any such ISR will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnCat2ISRStop(InterestingCat2ISR);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnCat2ISRStart

6.40 Os_TriggerOnChain

Trigger when a task is chained.

Syntax

```
void Os_TriggerOnChain(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Identifier of the task to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when an attempt is made to chain a specified task. (Noting that chain attempts can fail.)

TaskID can be set to OS_TRIGGER_ANY, in which case chaining of any task will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnChain(InterestingTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnActivation

6.41 Os_TriggerOnError

Trigger when an error occurs.

Syntax

```
void Os_TriggerOnError(
    StatusType Error
)
```

Parameters

Name	Type	Mode	Description
Error	StatusType	in	Identifier of the error to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified error is raised.

Error can be set to OS_TRIGGER_ANY, in which case any error will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnError(E_OS_LIMIT);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.42 Os_TriggerOnGetResource

Trigger when a resource is locked.

Syntax

```
void Os_TriggerOnGetResource(
    ResourceType ResourceID
)
```

Parameters

Name	Type	Mode	Description
ResourceID	ResourceType	in	Identifier of the resource to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified resource is locked.

ResourceID can be set to OS_TRIGGER_ANY, in which case any resource lock will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnGetResource(CriticalSection);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnReleaseResource

6.43 Os_TriggerOnIncrementCounter

Trigger when a counter is incremented.

Syntax

```
void Os_TriggerOnIncrementCounter(
    CounterType CounterID
)
```

Parameters

Name	Type	Mode	Description
CounterID	CounterType	in	Identifier of the software counter.

Return Values

None.

Description

Causes a trace trigger to occur when a specified counter is incremented.

CounterID can be set to OS_TRIGGER_ANY, in which case any counter increment will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnIncrementCounter(SWCounter);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnAdvanceCounter

6.44 Os_TriggerOnIntervalEnd

Trigger when a trace interval ends.

Syntax

```
void Os_TriggerOnIntervalEnd(
    Os_TraceIntervalIDType IntervalID
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Identifier of the interval to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified interval ends.

IntervalID can be set to OS_TRIGGER_ANY, in which case any interval end will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnIntervalEnd(EndToEndTimeMeasurement);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnIntervalStart
Os_TriggerOnIntervalStop

6.45 Os_TriggerOnIntervalStart

Trigger when a trace interval is started.

Syntax

```
void Os_TriggerOnIntervalStart(
    Os_TraceIntervalIDType IntervalID
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Identifier of the interval to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified interval starts.

IntervalID can be set to OS_TRIGGER_ANY, in which case any interval start will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnIntervalStart(EndToEndTimeMeasurement);
```

See Also

Os_TriggerOnIntervalEnd
Os_TriggerOnIntervalStop

6.46 Os_TriggerOnIntervalStop

Trigger when a trace interval ends.

Syntax

```
void Os_TriggerOnIntervalStop(
    Os_TraceIntervalIDType IntervalID
)
```

Parameters

Name	Type	Mode	Description
IntervalID	Os_TraceIntervalIDType	in	Identifier of the interval to trigger on.

Return Values

None.

Description

This call is a synonym for Os_TriggerOnIntervalEnd.

It causes a trace trigger to occur when a specified interval ends.

IntervalID can be set to OS_TRIGGER_ANY, in which case any interval end will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnIntervalStop(EndToEndTimeMeasurement);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnIntervalEnd

6.47 Os_TriggerOnReleaseResource

Trigger when a resource is unlocked.

Syntax

```
void Os_TriggerOnReleaseResource(
    ResourceType ResourceID
)
```

Parameters

Name	Type	Mode	Description
ResourceID	ResourceType	in	Identifier of the resource to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified resource is unlocked.

ResourceID can be set to OS_TRIGGER_ANY, in which case any resource unlock will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnReleaseResource(CriticalSection);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnGetResource

6.48 Os_TriggerOnScheduleTableExpiry

Trigger when a specified expiry point expires.

Syntax

```
void Os_TriggerOnScheduleTableExpiry(
    ExpiryID
)
```

Parameters

Name	Type	Mode	Description
ExpiryID	Os_TraceExpiryIDType	in	Identifier of the expiry to trigger on. The ExpiryID is formed by combining the name of the ScheduleTable and Expiry with an underscore character.

Return Values

None.

Description

Causes a trace trigger to occur when a specific expiry point is reached.

ExpiryID can be set to OS_TRIGGER_ANY, in which case any expiry *or alarm* will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
StartScheduleTableRel(SchedTable, 1);
Os_TriggerOnScheduleTableExpiry(SchedTable_ep1);
IncrementCounter(SystemCounter);
...
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.49 Os_TriggerOnSetEvent

Trigger when an event is set for a task.

Syntax

```
void Os_TriggerOnSetEvent(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Identifier of the task to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when an event is set for a specified task.

TaskID can be set to OS_TRIGGER_ANY, in which case any event setting will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnSetEvent(ExtendedTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

None.

6.50 Os_TriggerOnShutdown

Trigger when the OS is shutdown.

Syntax

```
void Os_TriggerOnShutdown(
    StatusType Status
)
```

Parameters

Name	Type	Mode	Description
Status	StatusType	in	Identifier of the shutdown exit code.

Return Values

None.

Description

Causes a trace trigger to occur when a specific status is passed to ShutdownOS.

Status can be set to OS_TRIGGER_ANY, in which case status value passed to ShutdownOS will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnShutdown(E_OK); /* Trigger on normal shutdown */
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

ShutdownOS

6.51 Os_TriggerOnTaskStart

Trigger when a task is started.

Syntax

```
void Os_TriggerOnTaskStart(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Identifier of the task to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified task starts running.

TaskID can be set to OS_TRIGGER_ANY, in which case any task start will cause the trigger to occur.

Note that a TaskID is started when its entry function is called, or when it resumes from the WAITING state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnTaskStart(InterestingTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnTaskStop

6.52 Os_TriggerOnTaskStop

Trigger when a task is stopped.

Syntax

```
void Os_TriggerOnTaskStop(
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskID	TaskType	in	Identifier of the task to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified task stops running.

TaskID can be set to OS_TRIGGER_ANY, in which case any task stop will cause the trigger to occur.

Note that a TaskID is stopped when its entry function is called, or when it enters the WAITING state.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnTaskStop(InterestingTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnTaskStart

6.53 Os_TriggerOnTaskTracepoint

Trigger when a task tracepoint is logged.

Syntax

```
void Os_TriggerOnTaskTracepoint(
    Os_TraceTracepointIDType TaskTracepointID,
    TaskType TaskID
)
```

Parameters

Name	Type	Mode	Description
TaskTracepointID	Os_TraceTracepointIDType	in	Identifier of the tracepoint to trigger on.
TaskID	TaskType	in	Identifier of the task.

Return Values

None.

Description

Causes a trace trigger to occur when a specified task-tracepoint for a specified task is logged.

TaskID can be set to OS_TRIGGER_ANY, in which any task-tracepoint with the specified value will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnTaskTracepoint(MyTaskTracepoint,InterestingTask);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnTracepoint

6.54 Os_TriggerOnTracepoint

Trigger when a tracepoint is logged.

Syntax

```
void Os_TriggerOnTracepoint(
    Os_TraceTracepointIDType TracepointID
)
```

Parameters

Name	Type	Mode	Description
TracepointID	Os_TraceTracepointIDType	in	Identifier of the tracepoint to trigger on.

Return Values

None.

Description

Causes a trace trigger to occur when a specified tracepoint is logged.

TracepointID can be set to OS_TRIGGER_ANY, in which any tracepoint will cause the trigger to occur.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TriggerOnTracepoint(MyTracepoint);
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_TriggerOnTaskTracepoint

6.55 Os_UploadTraceData

Uses an asynchronous communication to upload trace data a byte at a time.

Syntax

```
void Os_UploadTraceData(void)
```

Return Values

None.

Description

This API is responsible for sending individual bytes of trace data over a serial communications link. It uses callbacks into the application code to manage access to the actual communications link.

In polled mode, it is necessary to call this function frequently enough to ensure data is transmitted in a timely manner.

As a special case in interrupt mode, this function should be called from the Os_Cbk_TraceCommDataReady() callback and the transmit-interrupt handler.

An appropriate asynchronous serial device must be available and previously initialized. A typical serial link might be set to 115200bps, 8 data bits, no parity and 1 stop bit.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
/* This callback occurs when a new frame is ready for upload */
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommDataReady(void) {
  Os_UploadTraceData(); /* Causes call to
      Os_Cbk_TraceCommTxStart() */
}
ISR(asyncio) {
  Os_UploadTraceData();
}
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxStart(void) {
  /* Called from UploadTraceData when the first byte of a frame
       is ready to send.
   * It is immediately followed by a call to
       Os_Cbk_TraceCommTxByte().
   * In interrupt mode, this is used to enable the transmit
       interrupt.
```

```
   */
  enable_asyncio_interrupt();
}
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxByte(uint8 val) {
  /* Called from UploadTraceData when there is a byte ready to
     send */
  async_transmit(val);
}
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxEnd(void)  {
  /* Called from UploadTraceData when the last byte of data has
      been sent*/
  disable_asyncio_interrupt();
}
FUNC(boolean, OS_APPL_CODE) Os_Cbk_TraceCommTxReady(void) {
  /* Called from UploadTraceData to determine whether there is
     room in the transmit buffer */
  /* This should always return true in interrupt mode, because
     the interrupt should only
   * fire when there is room to send the next byte. */
  return async_tx_ready();
}
```

Calling Environment

Tasks/ISRs		AUTOSAR OS Hooks		RTA-OS3.0 Hooks	
Task	✓	PreTaskHook	✓	StackOverrunHook	✓
Category 1 ISR	✗	PostTaskHook	✓	TimeOverrunHook	✓
Category 2 ISR	✓	StartupTaskHook	✓		
		ShutdownHook	✓		
		ErrorHook	✗		
		ProtectionHook	✓		

See Also

Os_CheckTraceOutput
Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxEnd
Os_Cbk_TraceCommTxReady

7 RTA-TRACE Callbacks

7.1 Guide to Descriptions

Callbacks are code that is required by RTA-TRACE but must be provided by the user. This section documents all the callbacks required for RTA-TRACE. The descriptions have the following structure:

Syntax

```
/* C function prototype for the callback */
ReturnValue NameOfCallback(Parameter Type, ...)
```

Parameters

A list of parameters for each callback and their mode:

in The parameter is passed in to the callback by the OS

out The parameter is passed out of the API callback by passing a reference (pointer) to the parameter into the call.

inout The parameter is passed into the callback and then (updated) and passed out.

Return Values

A description of the return value of the callback,

Description

A detailed description of the required functionality of the callback.

Portability

The portability of the call between OSEK OS, AUTOSAR OS, RTA-OS3.0 and RTA-TRACE.

Example Code

```
A C code listing showing how to implement the callback.
```

Required When

The configuration that means user code must implement the callback.

See Also

A list of related callbacks.

7.2 Os_Cbk_TraceCommDataReady

Callback routine that signals when there is trace data ready to be sent.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommDataReady(void)
```

Return Values

None.

Description

When tracing in Bursting or Triggering modes, this gets called automatically when there is a new frame of data to be uploaded to RTA-TRACE.

When tracing in Free-running mode, this gets called from Os_CheckTraceOutput(), which must be called regularly by the application.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommDataReady(void) {
  Os_UploadTraceData(); /* Causes call to
     Os_Cbk_TraceCommTxStart() */
}
```

Required when

The callback may be provided if a communications link is used with tracing. A default version is present in the kernel library.

See Also

Os_UploadTraceData
Os_CheckTraceOutput
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxEnd
Os_Cbk_TraceCommTxReady

7.3 Os_Cbk_TraceCommInitTarget

Callback routine used to allow the application to perform initialization of external communication for tracing.

Syntax

```
FUNC(Os_TraceStatusType, OS_APPL_CODE)
    Os_Cbk_TraceCommInitTarget(void)
```

Return Values

The call returns values of type Os_TraceStatusType.

Description

Os_Cbk_TraceCommInitTarget supports the Os_TraceCommInit by providing application-specific code to initialize the communication link to RTA-TRACE. Typically it sets up an RS232 link.

E_OK should be returned if the initialization succeeded. Any other value will result in trace communication being disabled.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(Os_TraceStatusType, OS_APPL_CODE)
    Os_Cbk_TraceCommInitTarget(void){
  initialize_uart();
  return E_OK;
}
```

Required when

The callback must be provided if Os_TraceCommInit is used to initialize tracing using an external communications link.

See Also

Os_TraceCommInit

7.4 Os_Cbk_TraceCommTxByte

Callback routine that supplies a byte of trace data for sending.

Syntax

```
FUNC(void, OS_APPL_CODE)Os_Cbk_TraceCommTxByte(
    uint8 val
)
```

Return Values

None.

Description

This is called from UploadTraceData when there is a byte of data to send.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxByte(uint8 val) {
  /* Called from UploadTraceData when there is a byte ready to
     send */
  async_transmit(val);
}
```

Required when

The callback must be provided if Os_UploadTraceData is used.

See Also

Os_UploadTraceData
Os_CheckTraceOutput
Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxEnd
Os_Cbk_TraceCommTxReady

7.5 Os_Cbk_TraceCommTxEnd

Callback routine that signals that the last byte of trace data has been sent.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxEnd(void)
```

Return Values

None.

Description

This is called from UploadTraceData when the last byte of a frame has been sent.

In interrupt mode, this is used to disable the transmit interrupt.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxEnd(void)  {
  disable_asyncio_interrupt();
}
```

Required when

The callback must be provided if Os_UploadTraceData is used.

See Also

Os_UploadTraceData
Os_CheckTraceOutput
Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxReady

7.6 Os_Cbk_TraceCommTxReady

Callback routine used to discover if there is room to send the next trace data byte.

Syntax

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_TraceCommTxReady(void)
```

Return Values

The call returns values of type boolean.

Description

This is called from UploadTraceData to determine whether there is room in the transmit buffer to send the next byte.

This should always return true in interrupt mode, because the interrupt should only fire when there is room to send the next byte.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_TraceCommTxReady(void) {
  return async_tx_ready();
}
```

Required when

The callback must be provided if Os_UploadTraceData is used.

See Also

Os_UploadTraceData
Os_CheckTraceOutput
Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxStart
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxEnd

7.7 Os_Cbk_TraceCommTxStart

Callback routine that signals that the first byte of trace data is ready to be sent.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxStart(void)
```

Return Values

None.

Description

This is called from UploadTraceData when the first byte of a frame is ready to send.

It is immediately followed by a call to Os_Cbk_TraceCommTxByte().

In interrupt mode, this is used to enable the transmit interrupt.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_TraceCommTxStart(void) {
  enable_asyncio_interrupt();
}
```

Required when

The callback must be provided if Os_UploadTraceData is used.

See Also

Os_UploadTraceData
Os_CheckTraceOutput
Os_Cbk_TraceCommDataReady
Os_Cbk_TraceCommTxByte
Os_Cbk_TraceCommTxEnd
Os_Cbk_TraceCommTxReady

8 RTA-TRACE Types

8.1 Os_AsyncPushCallbackType

Type that represents a pointer to a void function that gets passed a single uint8 value. Used by Os_TraceDumpAsync()

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

8.2 Os_TraceCategoriesType

Type that is used to contain mask values relating to user-defined trace filter categories. An all and a non category are defined by default.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

OS_TRACE_NO_CATEGORIES
OS_TRACE_ALL_CATEGORIES

Example

```
Os_TraceCategoriesType ExtraTracing = DebugTracePoints |
    DataLogTracePoints;
```

8.3 Os_TraceClassesType

Type that is used to contain mask values relating to trace filter classes.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

OS_TRACE_ACTIVATIONS_CLASS
OS_TRACE_RESOURCES_CLASS
OS_TRACE_INTERRUPT_LOCKS_CLASS
OS_TRACE_SWITCHING_OVERHEADS_CLASS
OS_TRACE_TASKS_AND_ISRS_CLASS
OS_TRACE_ERRORS_CLASS
OS_TRACE_TASK_TRACEPOINT_CLASS
OS_TRACE_TRACEPOINT_CLASS
OS_TRACE_INTERVALS_CLASS
OS_TRACE_MESSAGE_DATA_CLASS
OS_TRACE_STARTUP_AND_SHUTDOWN_CLASS
OS_TRACE_ALARMS_CLASS
OS_TRACE_SCHEDULETABLES_CLASS
OS_TRACE_OSEK_EVENTS_CLASS
OS_TRACE_EXPIRY_POINTS_CLASS
OS_TRACE_NO_CLASSES
OS_TRACE_ALL_CLASSES

Example

```
Os_TraceClassesType AllTracepoints = OS_TRACE_TRACEPOINT_CLASS
    | OS_TRACE_TASK_TRACEPOINT_CLASS;
```

8.4 Os_TraceDataLengthType

The length of a data block (in bytes).

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TraceDataLengthType BlockLength = 8;
```

8.5 Os_TraceDataPtrType

A pointer to a block of data to log at a trace point or interval.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TraceDataPtrType DataPtr;
uint8 DataValues[10];
```

```
...
 DataPtr = &DataValue;
```

8.6 Os_TraceExpiryIDType

Enumerated type that defines Expiry points.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

The names of expiry points. These are generated using the pattern <schedeuletable_name>_<expiry_name>.

8.7 Os_TraceIndexType

An unsigned integer value of at least 16 bits representing a number of trace records.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
Os_TraceIndexType PreTriggerRecords = 100;
```

8.8 Os_TraceInfoType

An unsigned integer value representing a traced object.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

8.9 Os_TraceIntervalIDType

Enumerated type that defines RTA-TRACE trace intervals.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

The names of user defined trace intervals.

8.10 Os_TraceStatusType

Type containing the status of a trace API call.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

OS_TRACE_STATUS_OK
OS_TRACE_STATUS_COMM_INIT_FAILURE

8.11 Os_TraceTracepointIDType

Enumerated type that defines RTA-TRACE tracepoints.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Values

The names of user defined trace points.

8.12 Os_TraceValueType

An unsigned integer value representing either 16 or 32 bits depending on the configuration of compact time.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

9 RTA-TRACE Macros

9.1 OS_NUM_INTERVALS

The number of Trace Intervals declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

9.2 OS_NUM_TASKTRACEPOINTS

The number of TaskTracepoints declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

9.3 OS_NUM_TRACECATEGORIES

The number of Trace Categories declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

9.4 OS_NUM_TRACEPOINTS

The number of Tracepoints declared.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

9.5 OS_TRACE

This macro is only defined if tracing is enabled.

Portability

RTA-OS3.0	OSEK OS	AUTOSAR OS R3.0	RTA-TRACE
✓	✗	✗	✓

Example

```
#ifdef OS_TRACE
  ...
#endif
```

10 Coding Conventions

10.1 Namespace

The C programming language provides a single global scope for all names. This prevents any two names declared in an entire program code from being identical even if the names are declared in different compilation units. The AUTOSAR standard defines a naming convention for every basic software module to avoid problems with namespace clashes. This is defined by the “AUTOSAR General Requirements on Basic Software Modules”. RTA-OS3.0 has been implemented to satisfy these requirements. The namespace used by RTA-OS3.0 therefore reserves all names that are prefixed by:

- `OS*`
- `Os*`

Note however, that the interface provided by AUTOSAR OS R3.0 does not comply with the AUTOSAR naming convention. This means that the names used by AUTOSAR OS R3.0 for types, API calls, macros, constants, callbacks etc. are also reserved names and should not be duplicated in user code

RTA-OS3.0 defines OS API calls and macros internally according to the AUTOSAR general requirements and provides the AUTOSAR OS R3.0 names to the user through C macros. This does not apply to standard callbacks which retain their stanadrd name, for example `ErrorHook()`, `ShutdownHook()` etc.

This means the following forms are identical:

```
Os_StatusType Os_ActivateTask(Os_TaskType, Os_TaskId)

StatusType ActivateTask(TaskType, TaskId)
```

The two forms can be used interchangeably in user code if required, but only the second form represents standard AUTOSAR OS R3.0 API.

11 Configuration Language

11.1 Configuration Files

RTA-OS3.0 is configured using AUTOSAR's ECU Parameter description language. This section gives a short overview of AUTSOAR basic software module configuration in AUTOSAR XML and the extensions made by ETAS to the description language.

11.2 Understanding AUTOSAR XML Configuration

AUTOSAR uses eXtensible Markup Language (XML) as its configuration file format. AUTOSAR defines the tags and their semantics using an XML schema definition.

Every AUTOSAR XML file needs to reference the AUTOSAR schema instance that defines the structure of the XML elements for AUTOSAR XML files. In the simple case this is done as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR
xsi:schemaLocation="http://autosar.org/3.0.2 autosar.xsd" xmlns
   ="http://autosar.org/3.0.2" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance">
  ...
</AUTOSAR>
```

All element between <AUTOSAR> and </AUTOSAR> have the form <**ELEMENT**-NAME>. Only one AUTOSAR element is allowed per XML configuration file. All other AUTOSAR definitions are contained within this element.

If you need to mix AUTOSAR and non-AUTOSAR content within the same file then it is recommended that you use `autosar` as the namespace identifier. This is done as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<autosar:AUTOSAR
xmlns:autosar="http://autosar.org/3.0.2" xmlns:xsi="http://www.
   w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
   autosar.org/3.0.2 autosar.xsd">
...
</autosar:AUTOSAR>
```

In this case, all elements must now occur between <autosar:AUTOSAR> and </autosar:AUTOSAR> have the form <autosar:TAG-NAME>.

11.2.1 Packages

The <AUTOSAR> element is a container for exactly one <TOP-LEVEL-PACKAGES> element. The <TOP-LEVEL-PACKAGES> element represents the root of an XML object tree from which all objects in all configuration files can be accessed. The <TOP-LEVEL-PACKAGES> itself then contains one or more packages each defined with the <AR-PACKAGE> element. Each <AR-PACKAGE> defines a group of AUTOSAR elements or a set of sub-packages related to some part of AUTOSAR configuration.

Each <AR-PACKAGE> package definition is named using the <SHORT-NAME> element. Each package should have a unique name so that the elements contained within the package can be referenced by other packages. If two packages share the same name then they are assumed to be parts of the same package.

```
<AUTOSAR>
  <TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>MyPackage</SHORT-NAME>
      <DESC>This is one of my packages</DESC>
    </AR-PACKAGE>
    ...
    <AR-PACKAGE>
      <SHORT-NAME>MyOtherPackage</SHORT-NAME>
      <DESC>This is another</DESC>
    </AR-PACKAGE>
  </TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

The <AR-PACKAGE> element is used to define the package name as well as acting as a container for other elements, including<SUB-PACKAGES>.

Non basic software configuration can only be split at the <TOP-LEVEL-PACKAGES> level. When you need to work with multiple XML files you must therefore split them at the <TOP-LEVEL-PACKAGES> level. In the previous example, we might have decided to split this file into two different files, in which case in File 1 we would have:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR>
  <TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>SWCs</SHORT-NAME>
      <DESC>This is one of my packages</DESC>
      ...
```

```
    </AR-PACKAGE>
  </TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

In File 2 we would have the second AR-PACKAGE:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR>
  <TOP-LEVEL-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Interfaces</SHORT-NAME>
      <DESC>This is another</DESC>
      ...
    </AR-PACKAGE>
  </TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

11.3 ECU Configuration Description

AUTOSAR basic software uses a different configuration concept to the rest of AUTOSAR. Configuration uses an ECU configuration description file. This file is also an XML file, but the use of XML is significantly different to the rest of AUTOSAR configuration.

Rather than define a dedicated set of XML tags for the configuration of each basic software module, the ECU configuration description defines a <MODULE-CONFIGURATION> that contains CONTAINERS that hold configuration data in a <CONTAINER>.

Each <CONTAINER> holds <PARAMETER-VALUES>, <REFERENCE-VALUES> and <SUB-CONTAINERS>. <SUB-CONTAINERS> hold <CONTAINER> definitions, allowing a hierarchy of configuration containers to be formed.

This structure is common to all AUTOSAR basic software modules. The same format is used for the OS as for COM, NM, etc. The structure is customized for different basic software modules using a <DEFINITION-REF>. Each <MODULE-CONFIGURATION> and <CONTAINER> has a <DEFINITION-REF> which references the AUTOSAR ECU Configuration Definition. The <DEFINITION-REF> is an absolute reference to the definition of a configuration item in the AUTOSAR ECU Configuration Definition. This is also an XML file and defines the type of the container and what configuration elements are allowed.

By default, references are rooted at /AUTOSAR. For the OS there are things like:

- /AUTOSAR/Os/OsTask
- /AUTOSAR/Os/OsTask/OsTaskPriority
- /AUTOSAR/Os/OsResource
- /AUTOSAR/Os/OsIsr

Each definition in the definition file specifies:

- how many instance of the <CONTAINER> can exist in the <MODULE-CONFIGURATION>
- how many of each of the <PARAMETER-VALUES>, <REFERENCE-VALUES> and <SUB-CONTAINERS> the container can hold. This is called the *multiplicity* and the definition file specifies a <LOWER-MULTIPLICITY> and an <UPPER-MULTIPLICITY>.
- the definitions of the <PARAMETER-VALUES>, <REFERENCE-VALUES> and <SUB-CONTAINERS> the <CONTAINER> can hold

The description files used to configure AUTOSAR OS is written according to the rules specified in the definition file. The following example shows a valid description file for the OS that includes a single task called MyTask:

```
<ELEMENTS>
  <MODULE-CONFIGURATION>
    <SHORT-NAME>MyOSConfiguration</SHORT-NAME>
    <DEFINITION-REF>/AUTOSAR/Os</DEFINITION-REF>
    <CONTAINERS>
      <!-- Configuration containers -->
      <CONTAINER>
        <SHORT-NAME>MyTask</SHORT-NAME>
        <DEFINITION-REF>/AUTOSAR/Os/OsTask</DEFINITION-REF>
        <!-- Parameters (or sub-containers) as defined by the
           DEFINTION-REF -->
        <PARAMETER-VALUES>
            <INTEGER-VALUE>
                <DEFINITION-REF DEST="INTEGER-PARAM-DEF">/
                   AUTOSAR/Os/OsTask/OsTaskPriority</DEFINITION
                   -REF>
                <VALUE>27</VALUE>
            </INTEGER-VALUE>
            <INTEGER-VALUE>
                <DEFINITION-REF DEST="INTEGER-PARAM-DEF">/
                   AUTOSAR/Os/OsTask/OsTaskActivation</
                   DEFINITION-REF>
```

```
            <VALUE>1</VALUE>
          </INTEGER-VALUE>
          <ENUMERATION-VALUE>
            <DEFINITION-REF DEST="ENUMERATION-PARAM-DEF">/
              AUTOSAR/Os/OsTask/OsTaskSchedule</DEFINITION
              -REF>
            <VALUE>FULL</VALUE>
          </ENUMERATION-VALUE>
        </PARAMETER-VALUES>
      </CONTAINER>
      ...
    <CONTAINERS>
  </MODULE-CONFIGURATION>
</ELEMENTS>
```

Standard AUTOSAR configuration elements for the OS are documented in the *AUTOSAR Specification of Operating System Release Release 3.0 Version 3.0.1 Revision 0003* .

11.4 RTA-OS3.0 Configuration Language Extensions

In addition to the standard AUTOSAR configuration elements, each AUTOSAR OS vendor will also define their own pieces of ECU configuration to capture things that are not standardized in AUTOSAR - for example the allocation of vector addresses and priorities to interrupts.

Vendor extensions to AUTOSAR configuration take the standard AUTOSAR Standard Module Definition (called the StMD) and produce a Vendor Specific Module Definition (VSMD). This includes all the elements from AUTOSAR plus those defined by the vendor. More information about this process can be found in *AUTOSAR Specification of ECU Configuration Release Release 3.0 Version 2.0.1 Revision 0002*.

The AUTOSAR <PACKAGE> name for the VSMD must not be `AUTOSAR` so that tools can distinguish between standard configuration and vendor-specific configuration. In RTA-OS3.0, the VSMD <PACKAGE> is called `RTAOS` and all references to RTA-OS configuration objects have the form `/RTAOS/path to configuration element`. References to standard AUTOSAR objects retain the the form `/AUTOSAR/path to configuration element`. For example:

```
<CONTAINER>
  <!-- Top-level container for global OS configuration
     parameters -->
  <SHORT-NAME>OsInfo</SHORT-NAME>
```

```
  <DEFINITION-REF DEST="PARAM-CONF-CONTAINER-DEF">/AUTOSAR/Os/
     OsOS</DEFINITION-REF>

  <PARAMETER-VALUES>
    <!-- Standard AUTOSAR configuration parameters -->
    <ENUMERATION-VALUE>
      <DEFINITION-REF DEST="ENUMERATION-PARAM-DEF">/AUTOSAR/Os/
         OsOS/OsStatus</DEFINITION-REF>
      <VALUE>...</VALUE>
    </ENUMERATION-VALUE>
    <ENUMERATION-VALUE>
      <DEFINITION-REF DEST="ENUMERATION-PARAM-DEF">/AUTOSAR/Os/
         OsOS/OsScalabilityClass</DEFINITION-REF>
      <VALUE>...</VALUE>
    </ENUMERATION-VALUE>
    <BOOLEAN-VALUE>
      <DEFINITION-REF DEST="BOOLEAN-PARAM-DEF">/AUTOSAR/Os/OsOS
         /OsStackMonitoring</DEFINITION-REF>
      <VALUE>...</VALUE>
    </BOOLEAN-VALUE>
    <!-- ... -->

    <!-- RTA-OS-specific configuration parameters -->
    <STRING-VALUE>
      <DEFINITION-REF DEST="STRING-PARAM-DEF">/RTAOS/Os/OsOS/
         OsDefTaskStack</DEFINITION-REF>
      <VALUE>...</VALUE>
    </STRING-VALUE>
    <STRING-VALUE>
      <DEFINITION-REF DEST="STRING-PARAM-DEF">/RTAOS/Os/OsOS/
         OsDefCat1Stack</DEFINITION-REF>
      <VALUE>...</VALUE>
    </STRING-VALUE>
    <STRING-VALUE>
      <DEFINITION-REF DEST="STRING-PARAM-DEF">/RTAOS/Os/OsOS/
         OsDefCat2Stack</DEFINITION-REF>
      <VALUE>...</VALUE>
    </STRING-VALUE>
    <!-- ... -->
  </PARAMETER-VALUES>
</CONTAINER>
```

The following sections define the extensions to the standard AUTOSAR configuration attributes that are supported by RTA-OS3.0. Each sec-

tion defines (or extends) a <CONTAINER> and the <PARAMETER-VALUES>, <REFERENCE-VALUES> and <SUB-CONTAINERS> that the <CONTAINER> can hold.

The presence of vendor specific extensions to AUTOSAR is portable to 3rd party AUTOSAR configuration tooling. However, this applies only to the syntax of extensions. The semantics of extensions is, of course, not portable. For example, if one vendor defines a configuration element called `OsEnableSpecialOptimization` then another vendor will not be able to do anything with this configuration because their implementation cannot know the meaning of a "special optimization".

11.4.1 Container: OsRTATarget

Description

Parameters to represent a specific piece of target hardware.

Multiplicity

0..1

String Parameters

Name	Occurs	Description
OsRTATargetName	1..1	The name of the target system.
OsRTATargetVersion	0..1	The version number of the OS on the target system.
OsRTATargetVariant	0..1	The variant of the OS for this target system.

Sub-container: Param

Description

Target-specific parameter representation.

Multiplicity

0..*

String Parameters

Name	Occurs	Description
Value	1..1	Value of the parameter

11.4.2 Container: OsAppMode

Sub-container: OsTbRate

Description

Specifies the duration of a tick on a non-default timebase

Multiplicity

0..*

String Parameters

Name	Occurs	Description
OsRate	1..1	The duration of one tick of the container-specified timebase

11.4.3 Container: OsCounter

String Parameters

Name	Occurs	Description
OsFormat	0..1	A string that specifies a format for each tracepoint.

Reference Parameters

Name	Occurs	Destination
OsTimebaseRef	0..1	/AUTOSAR/Os/OsTimebase

11.4.4 Container: OsIsr

Boolean Parameters

Name	Occurs	Description
OsIsrFifo	0..1	Defines whether or not the interrupt is FIFO if retrig-gering or looping

Enumeration Parameters

Name	Occurs	Description
OsTraceFilter	0..1	Describes whether this ISR is traced with RTA-TRACE. Permitted values are: **ALWAYS** Always trace this ISR **NEVER** Never trace this ISR **RUNTIME** Allow the user to control tracing of this ISR at runtime.
OsIsrBehavior	0..1	Describes the interrupt as simple, retriggering or looping Permitted values are: **SIMPLE** Specifies that an interrupt from the same source cannot be generated during the execution of the ISR. **LOOPING** Specifies that the interrupt handler will loop within the ISR until all pending interrupts have been serviced. **RETRIGGERING** Specifies that the interrupt handler will either leave the interrupt pending or reassert it.

Integer Parameters

Name	Occurs	Description
OsIsrPriority	1..1	The Interrupt Priority **Range:** 0..maxint
OsIsrArbitrationOrder	0..1	The interrupt arbitration order - lower means interrupt is taken in preference. **Range:** 0..maxint
OsIsrBufferSize	0..1	Defines how many of this interrupt can occur at once. Zero means the buffering is done per profile. **Range:** 0..maxint

String Parameters

Name	Occurs	Description
OsIsrBudget	0..1	Execution budget expressed as a float, then timebase name, then units.
OsIsrStackAllocation	0..1	ISR manual stack allocation in bytes
OsIsrAddress	1..1	The Interrupt Vector

Reference Parameters

Name	Occurs	Destination
OsRegSetRef	0..*	/AUTOSAR/Os/OsRegSet

11.4.5 Container: OsOS

Boolean Parameters

Name	Occurs	Description
OsSuppressVectorGen	0..1	Suppresses generation of the vector table.

Integer Parameters

Name	Occurs	Description
OsCyclesPerSecond	0..1	Defines the clock speed of the target **Range:** 0..maxint
OsTicksPerSecond	0..1	Defines the stopwatch speed of the target **Range:** 0..maxint

String Parameters

Name	Occurs	Description
OsDefTaskStack	0..1	Default stack values
OsDefCat1Stack	0..1	Default category 1 stack values
OsDefCat2Stack	0..1	Default category 2 stack values

Sub-container: Param

Description

Representation of parameters

Multiplicity

0..*

String Parameters

Name	Occurs	Description
Value	1..1	Value of the parameter

Sub-container: OsHooks

Boolean Parameters

Name	Occurs	Description
OsStackFaultHook	0..1	Use stack fault hook

11.4.6 Container: OsRegSet

Description

Target specific register sets that can be associated with a task or ISR. By association with a task or ISR, the integrator is specifying that a specific task or ISR uses this register set. Having no association defined allows potential optimization.

Multiplicity

0..*

11.4.7 Container: OsTask

Enumeration Parameters

Name	Occurs	Description
OsTraceFilter	0..1	Describes whether this Task is traced with RTA-TRACE. Permitted values are: **ALWAYS** Always trace this ISR **NEVER** Never trace this ISR **RUNTIME** Allow the user to control tracing of this ISR at runtime.

String Parameters

Name	Occurs	Description
OsTaskStackAllocation	0..1	Task manual stack allocation
OsTaskWaitStack	0..1	Task stack usage when invoking WaitEvent
OsTaskBudget	0..1	Execution budget expressed as a float, then timebase name, then units.

Reference Parameters

Name	Occurs	Destination
OsRegSetRef	0..*	/AUTOSAR/Os/OsRegSet
OsHigherThanTaskRef	0..*	/AUTOSAR/Os/OsTask

11.4.8 Container: OsTimebase

Description

An OsTimebase is a set of related timeunits, usually multiples of each other.

Multiplicity

0..*

Boolean Parameters

Name	Occurs	Description
OsStopwatch	1..1	Defines whether this is the stopwatch timebase.

Integer Parameters

Name	Occurs	Description
OsMaxValue	1..1	Defines the maximum value of the timebase **Range:** 0..maxint

Sub-container: OsTbUnit

Description

An OsTbUnit is a time unit (eg. seconds)

Multiplicity

0..*

Float Parameters

Name	Occurs	Description
OsUnitAmount	1..1	Reserved for future use.
OsBaseAmount	1..1	Reserved for future use.

String Parameters

Name	Occurs	Description
OsBaseUnit	1..1	Reserved for future use.

11.4.9 Container: OsTrace

Description

RTA-TRACE Data

Multiplicity

0..1

Boolean Parameters

Name	Occurs	Description
OsTraceEnabled	0..1	Enables or disables tracing.
OsTraceCompactID	0..1	Trace Compact Identifiers
OsTraceCompactTime	1..1	Use compact time format
OsTraceTgtStack	1..1	Enable stack recording.
OsTraceTgtTrigger	1..1	Runtime target triggering.
OsTraceAutoComms	1..1	Initialise trace comms link at startup
OsTraceAutoRepeat	1..1	Call set trace repeat at startup

Enumeration Parameters

Name	Occurs	Description
OsTraceAuto	1..1	The autostart type for RTA-TRACE Permitted values are: **NONE** Don't automatically start tracing **BURSTING** Start tracing in bursting mode (wait till buffer is full before uploading) **TRIGGERING** Start tracing, waiting for a trigger **FREE_RUNNING** Start tracing continuously

Integer Parameters

Name	Occurs	Description
OsTraceBufferSize	1..1	The trace buffer size (in number of trace records) **Range:** 0..maxint

Sub-container: OsEnumeration

Description

Specifies an enumeration for tracing.

Multiplicity

0..*

Sub-container: OsEnumeration/Param

Description

Representation of name-value pairs

Multiplicity

0..*

String Parameters

Name	Occurs	Description
Value	1..1	Value of the parameter

Sub-container: OsTraceTracepoint

Description

Specifies a tracepoint

Multiplicity

0..*

Integer Parameters

Name	Occurs	Description
OsTraceTracepointID	1..1	Specifies a tracepoint ID (1-n, 0 indicates auto) **Range:** 0..maxint

String Parameters

Name	Occurs	Description
OsTraceTracepointFormat	0..1	A string that specifies a format for each tracepoint.

Sub-container: OsTraceTaskTracepoint

Description

Specifies a task tracepoint

Multiplicity

0..*

Integer Parameters

Name	Occurs	Description
OsTraceTaskTracepointID	1..1	Specifies a tracepoint ID (1-n, 0 indicates auto) **Range:** 0..maxint

String Parameters

Name	Occurs	Description
OsTraceTaskTracepointFormat	0..1	A string that specifies a format for each tracepoint.

Reference Parameters

Name	Occurs	Destination
OsTaskRef	0..1	/AUTOSAR/Os/OsTask
OsIsrRef	0..1	/AUTOSAR/Os/OsIsr

Sub-container: OsInterval

Description

Specifies a named interval.

Multiplicity

0..*

Integer Parameters

Name	Occurs	Description
OsIntervalID	1..1	Specifies a interval identifier (1-n, 0 indicates auto) **Range:** 0..maxint

String Parameters

Name	Occurs	Description
OsIntervalFormat	0..1	A string that specifies a format for each interval

Sub-container: Param

Description

Representation of name-value pairs

Multiplicity

0..*

String Parameters

Name	Occurs	Description
Value	1..1	Value of the parameter

Sub-container: OsClass

Description

Specifies an unnamed trace class.

Multiplicity

0..*

Boolean Parameters

Name	Occurs	Description
OsClassAutostart	0..1	For a run-time trace class, this determines whether it is started automatically at runtime.

Enumeration Parameters

Name	Occurs	Description
OsClassFilter	1..1	Specifies the filtering for a class. Permitted values are: **ALWAYS** Always trace this class **NEVER** Never trace this class **RUNTIME** Allow the user to control tracing of this class at runtime.

Sub-container: OsCategory

Description

Specifies a named trace class.

Multiplicity

0..*

Boolean Parameters

Name	Occurs	Description
OsCategoryAutostart	0..1	For a run-time trace category, this determines whether it is started automatically at runtime.

Enumeration Parameters

Name	Occurs	Description
OsCategoryFilter	1..1	Specifies the filtering for a category. Permitted values are: **ALWAYS** Always trace this category **NEVER** Never trace this category **RUNTIME** Allow the user to control tracing of this category at runtime.

Integer Parameters

Name	Occurs	Description
OsCategoryMask	1..1	Specifies a category mask. 0 represents auto. **Range:** 0..maxint

11.5 Project Description Files

A single logical OS configuration can be split across multiple XML configuration files. The files can be edited individually or simultaneously by the **rtaoscfg** configuration tool.

To help with the management of large, complex, configurations, RTA-OS3.0 provides a convenient shorthand for you to group a set of multiple files that represent a single logical OS configuration. This is called a "project". The files that comprise the project are referenced from a project file.

ETAS *Project files are specific to the RTA-OS3.0 tools and may not be portable to third party AUTOSAR tooling.*

A project file is an XML file that has the following structure:

```
    file ::=  <?xml version="1.0"?>
              <RTAOS_Project version="1.0">
              [<Working name="filename"/>]
              {<File name="filename"/>}
              [options]
              </RTAOS_Project>
 options ::=  <Options>
              {<Option name="filename">value</Option>}
              </Options>
   value ::=  booleanvalue | stringvalue |integervalue
```

12 Command Line

The tools shipped with RTA-OS3.0 can be invoked from the command line, making them easy to integrate into a build process. All commands accept any number of XML input files together with tool-specific options as parameters. The ordering of command line parameters is unimportant: options and XML files can be mixed freely.

Some command line options can be specified using either short or long (POSIX style) names. The two options forms provide identical functionality and can be used interchangeably.

When a command line option takes an argument, the argument appears immediately following short name options and after a colon following long name options. For example, an option with argument arg could appear as either

`command -oarg` or `command --option:arg`

The two forms are equivalent and can be mixed on the command line.

Optional settings for arguments are placed in brackets immediately before the argument itself. For example, assuming argument arg had a setting s, it would appear as either:

`command -o[s]arg` or `command --option:[s]arg`

12.1 rtaoscfg

The command **rtaoscfg** runs the graphical RTA-OS3.0 configuration editor.

```
rtaoscfg [options] <files>
```

12.1.1 Options

Option	Description
`--diagnostic`	Display the diagnostic information on the standard output. Diagnostic information includes: • The version of the tool executable • The names and versions of all tool plug-ins • The names and version of all target plug-ins • The location and contents of the license file
`-h, -?, --help`	Display usage information on the standard output.
`--nomsgbox`	Do not prompt the user with a message box when an error causes the configuration tool to exit.
`-o[<EXPS>]<DIR>` `--output:[<EXPS>]<DIR>`	Place all generated output files into the directory <DIR>. The optional <EXPS> clause places all generated files whose names match the comma-separated lost of expressions in <EXPS> in the directory <DIR>. Expressions can include the following wildcards: **?** matches a single character * matches a sequence of 1 or more characters

Option	Description
`--status:<STATUS>`	Generate a kernel library for the specified <STATUS> level. <STATUS> has two valid options: 1. STANDARD 2. EXTENDED If the OsStatus value is set in the input configuration then this option overrides the setting.
`--target:[<VARIANT>]<TARGET>`	Generate a kernel library for the specified <TARGET>. If multiple versions of <TARGET> are installed then the most recent version of the <TARGET> is selected. Selection of a specific version is possible using <TARGET>_<VERSION>. The option <VARIANT> selects a variant of <TARGET>. Both <TARGET> and <VARIANT> override the OsTarget and OsTargetVariant settings in the configuration file. A list of available targets and their associated versions and variants can be generated using `--target:?`
`--target_option:<NAME=VALUE>`	Override target option <NAME> with <VALUE>. A list of options is obtained using `--target_option:?`.
`--trace:<OPTION>`	Enable or disaable RTA-TRACE. <OPTION> may be one of: **on** enables RTA-TRACE (equivalent to setting `OsTraceEnabled` to `true`) **off** disables RTA-TRACE (equivalent to setting `OsTraceEnabled` to `false`

Option	Description
`--xml:<OPTION>`	Control the behavior of the XML processor when reading `<files>`. `<OPTION>` can be one of: **Novalidate** do not validate the input against the XML schema.
`--xmlschema:<SCHEMA>`	If validating the XML against a schema (`--xml:novalidate` is not set) then use the `<SCHEMA>` for the validation.

12.1.2 Generated Files

rtaoscfg does not generate any files directly. When the Builder is used in rtaoscfg this calls **rtaosgen**. See Section 12.2.2 for details of the files generated by **rtaosgen**.

12.1.3 Examples

Open a single file Config.xml for editing:

```
rtaoscfg Config.xml
```

Open an RTA-OS3.0 project file for editing:

```
rtaoscfg MyProject.rtaos
```

12.2 rtaosgen

The command **rtaosgen** runs the RTA-OS3.0 kernel library generator.

```
rtaosgen [options] <files>
```

12.2.1 Options

Option	Description
`@<FILE>`	Read command line parameters from `<FILE>`. Each command in `<FILE>` must appear on a separate. Quotation marks are not required to escape white space for filenames inside a command file. The `@<FILE>` option can itself appear multiple times inside `<FILE>`.

Option	Description
`--build:<OPTION>`	Pass `<OPTION>` to the build environment. `<OPTION>` may be one of: **verbose** display all build messages on the standard output **quiet** display no build messages on the standard output **clean** clean the build directory before building
`--debug:<OPTION>`	Keep generated assembler or source code[1]
`--diagnostic`	Display the diagnostic information on the standard output. Diagnostic information includes: • The version of the tool executable • The names and versions of all tool plug-ins • The names and version of all target plug-ins • The location and contents of the license file
`-h, -?, --help`	Display usage information on the standard output.
`-I<PATHS>` `--include:<PATHS>`	Add the each path in the comma-separated list `<PATHS>` to the include path for the builder.
`--nobuild`	Perform checks on the input configuration but does not build an RTA-OS3.0 library.
`--noinfo`	Suppress all information messages.
`--nowarnings`	Suppress all warning messages.

[1] Keeping source code is only possible with a valid source code license

Option	Description
`-o[<EXPS>]<DIR>` `--output:[<EXPS>]<DIR>`	Place all generated output files into the directory <DIR>. The optional <EXPS> clause places all generated files whose names match the comma-separated lost of expressions in <EXPS> in the directory <DIR>. Expressions can include the following wildcards: **?** matches a single character * matches a sequence of 1 or more characters
`--report:<REPORT>`	Generate the REPORT. A list of available reports is displayed on the standard output using `--report:?`
`--samples:[<SAMPLE>]<OPTION>`	Generate example code for <SAMPLE>. Use `--samples:[<SAMPLE>]overwrite` to write over existing samples. Use `--samples:?` to view available samples.
`--status:<STATUS>`	Generate a kernel library for the specified <STATUS> level. <STATUS> has two valid options: 1. STANDARD 2. EXTENDED If the OsStatus value is set in the input configuration then this option overrides the setting.

Option	Description
`--target:[<VARIANT>]<TARGET>`	Generate a kernel library for the specified <TARGET>. If multiple versions of <TARGET> are installed then the most recent version of the <TARGET> is selected. Selection of a specific version is possible using <TARGET>_<VERSION>. The option <VARIANT> selects a variant of <TARGET>. Both <TARGET> and <VARIANT> override the OsTarget and OsTargetVariant settings in the configuration file. A list of available targets and their associated versions and variants can be generated using `--target:?`
`--target_option:<NAME=VALUE>`	Override target option <NAME> with <VALUE>. A list of options is obtained using `--target_option:?`.
`--trace:<OPTION>`	Enable or disaable RTA-TRACE. <OPTION> may be one of: **on** enables RTA-TRACE (equivalent to setting `OsTraceEnabled` to `true`) **off** disables RTA-TRACE (equivalent to setting `OsTraceEnabled` to `false`
`--using:<FILES>`	**`#include`** each file in the comma-separated list <FILES> at the start of each library source file.
`--verbose`	Generate additional information when running.
`--version`	Show version information in compact form. More detailed information can be obtained using `--diagnostic`.
`--xml:<OPTION>`	Control the behavior of the XML processor when reading <files>. <OPTION> can be one of: **Novalidate** do not validate the input against the XML schema.

Option	Description
`--xmlschema:<SCHEMA>`	If validating the XML against a schema (`--xml:novalidate` is not set) then use the <SCHEMA> for the validation.

12.2.2 Generated Files

When **rtaosgen** runs and terminates without generating any errors or fatal messages then it will have generated the following files:

Filename	Contents
`Os.h`	The main include file for the OS.
`Os_Cfg.h`	Declarations of the objects you have configured. This is included by `Os.h`.
`Os_MemMap.h`	AUTOSAR memory mapping configuration used by RTA-OS3.0 to merge with the system-wide `MemMap.h` file.
`RTAOS.<lib>`	The RTA-OS3.0 library for your application. The extension <lib> depends on your target.
`RTAOS.<lib>.sig`	A signature file for the library for your application. The extension <lib> depends on your target.

There may be other files which are generated that are specific to your port. A list of additional files that can be generated can be found in the *RTA-OS3.0 Target/Compiler Port Guide* for your port.

12.2.3 Examples

Display the usage information

```
rtaosgen --help
```

Generate the OS described by `Config.xml` and generate sample AUTOSAR header files that will work with the OS. Create the library including both these generated files and the OS-specific generated files that are placed in the current directory. This is the standard command line you use when you will *not* be integrating RTA-OS3.0 with 3rd party AUTOSAR software:

```
rtaosgen --samples:[Includes] --include:Samples\\Includes
   Config.xml
```

Generate the OS described in BigConfig.rtaos using the AUTOSAR header files located at `PathToAutosarHeaderFiles` and the OS-specific header files that will be generated in the current directory. This is the standard command line you use when integrating RTA-OS3.0 with 3rd party AUTOSAR software:

```
rtaosgen --include:PathToAutosarHeaderFiles BigConfig.rtaos
```

List which sample files can be generated for the ManchesterMk1 target:

```
rtaosgen --target:ManchesterMk1 --samples:?
```

List which reports can be generated for the ManchesterMk1 target:

```
rtaosgen --target:ManchesterMk1 --report:?
```

Generate the OS as in the first example, but overwrite the existing sample includes files and override the target to be ManchesterMk1:

```
rtaosgen --samples:[Includes]overwrite --include:Samples\\
   Includes --target:ManchesterMk1 Config.xml
```

Generate the OS from a description split between `CoreConfig.xml` and `TargetConfig.xml`:

```
rtaosgen --include:PathToAutosarHeaderFiles CoreConfig.xml
   TargetConfig.xml
```

Generate the OS described in `Config.xml`, place the header files in `C:\working\OS\inc` and the library (plus the associated signature file) in `C:\working\OS\lib`

```
rtaosgen --include:PathToAutosarHeaderFiles --output:[*.h]C:\\
   working\\OS\\inc --output:[*.lib,*.sig]C:\\working\\OS\\inc
   Config.xml
```

13 Output File Formats

13.1 RTA-TRACE Configuration files

RTA-OS3.0 generates an RTA-TRACE configuration file when RTA-TRACE is enabled. The format of this file is similar to the ORTI format and is described in detail in the *RTA-TRACE OS Instrumenting Kit Manual*.

13.2 ORTI Files

This section describes the ORTI objects output by RTA-OS3.0.

When ORTI output is supported by a port and ORTI generation is configured then a file called `RTAOS.orti` is generated when the kernel is built using **rtaosgen**.

An ORTI object encapsulates information about OS objects in RTA-OS3.0, for example tasks, ISRs, alarms etc. An application may contain zero or more instances of each ORTI objects, each of which has a unique name. Each ORTI object has a number of attributes and each attribute has a value.

For example, the OS has a `RUNNINGTASK` attribute that shows the task that is currently running.

The following sections present the ORTI objects generated. Each section has the following structure:

Object

Name of the ORTI object

Description

A description of the ORTI object.

Attributes

The attributes for the ORTI object.

Attribute	Description
Attribute Name	*Attribute ORTI file description* - Description of attribute

Each row of the table names the attribute being described and gives a brief explanation of it. The name of each attribute is given in the Attribute column. Attributes that are prefix with vs_ have been added for RTA-OS3.0 support and are not standard ORTI attributes. Your debugger may or may not be able to display these attributes depending on how well it conforms to the ORTI standard.

Many debuggers display the attribute name. However, some debuggers choose to display the attribute description that is present in the ORTI file

instead. The descriptions used in RTA-OS3.0 appear in quotation marks at the start of the Description column.

13.2.1 OS

Object

OS

Description

There is only one OS object. It takes the name of the RTA-OS3.0 project.

Attributes

Attribute	Description
RUNNINGTASK	*Running task* - The name of the TASK that is currently running. If an ISR interrupts a task this attribute will continue to display the name of the task that was interrupted while the ISR is executing.
RUNNINGTASKPRIORITY	*Running task priority* - The current priority of the running task, using the same terms as in the OIL file. RUNNINGTASKPRIORITY does not show the effect of locking a resource shared by tasks and ISRs.
RUNNINGISR2	*Running cat 2 ISR* - The value of this attribute is the name of the Category 2 ISR that is currently running (if there is one). NO_ISR is displayed if no Category 2 ISR is currently running.
SERVICETRACE	*OS Services Watch*- Indicates the entry or exit of a service routine (an RTA-OS3.0 Component API call) and the name of this routine. Some debuggers recognize this attribute as a special trace attribute and can provide additional diagnostic support. On other debuggers, you will be shown which API call was most recently entered or completed.
LASTERROR	*Last OSEK error* - Gives the name of the last error that has occurred. Initially set to E_OK.

Attribute	Description
CURRENTAPPMODE	*Current AppMode* - Current application mode using the names stated in the XML file. The value *unknown AppMode* is reported if the application mode does not conform to a value in the XML file.

13.2.2 Task

Object

TASK

Description

Generated in response to task declarations in the configuration file.

Attributes

Attribute	Description
STATE	*State* - The task state. One of SUSPENDED, RUNNING, READY and WAITING.
vs_BASEPRIORITY	*Base priority* - Gives the base priority of the task. The base priority is the priority of the task as defined in the OIL file.
PRIORITY	*Dispatch priority* - Gives the dispatch priority of the task. The dispatch priority is the priority that the task starts running at. This can be higher than the base priority if internal resources are used or if the task is non-preemptable.
CURRENTACTIVATIONS	*Activations* - Gives the maximum number of activations allowed.

13.2.3 Category 1 ISR

There are no ORTI objects generated for Category 1 ISRs.

13.2.4 Category 2 ISR

There are no ORTI objects generated for Category 2 ISRs.

13.2.5 Resource

Object

RESOURCE

Description

Generated in response to resource declarations in the configuration file.

Attributes

Attribute	Description
PRIORITY	*Ceiling Priority* - Gives the ceiling priority of the resource in terms of the task priority that defines the ceiling.
LOCKER	*Resource locker* - Shows the current holder of the resource.
STATE	*Resource State* - Shows the state of the resource as locked or not locked.

13.2.6 Events

There are no ORTI objects generated for events.

13.2.7 Counter

There are no ORTI objects generated for counters.

13.2.8 Alarm

Object

ALARM

Description

Only generated in response to alarm declarations in the configuration file.

Attributes

Attribute	Description
ALARMTIME	*Alarm Time* - Shows when the alarm expires next. Refer to the Count value in the COUNTER object to establish the current count value.
CYCLETIME	*Cycle Time* - Gives the period of the cycle for a cyclic alarm. CYCLETIME will be zero for a single-shot alarm.

Attribute	Description
ACTION	*Action* - The action to perform when the alarm expires. This can include the following: • Activate a task. • Set an event. • Execute a callback function.
STATE	*Alarm state* - Indicates whether the alarm is running. Takes the value RUNNING or STOPPED.
COUNTER	*Counter* - Gives the name of the counter to which this alarm is attached.

13.2.9 Schedule Table

Object

SCHEDULETABLE

Description

Only generated in response to alarm declarations in the configuration file.

Attributes

Attribute	Description
COUNTER	*Counter* - Gives the name of the counter to which this alarm is attached.
STATE	*State* - Indicates the state of the schedule table.
EXPIRYTIME	*Expiry Time* - The tick at which the next expiry point is due to be processed.
NEXT	*Next table* - The next schedule table (if set).

14 Compatibility and Migration

This chapter provides compatibility information for RTA-OS3.0 with other ETAS tooling and outlines the major changes between RTA-OS3.0 and the earlier RTA-OSEK series of operating systems to assist users migrating to RTA-OS3.0.

14.1 ETAS Tools

The following table outlines the compatibility between RTA-OS3.0 and other ETAS software tools. Compatibility is split into two parts - the configuration language and the use of the API. The following indications are given:

- ✓ Fully compatible
- ✓ Partially compatible, see the notes for more details
- ✗ Not compatible

For a more detailed discussion of specific cases, please contact ETAS.

Product	Version	Compatibility		Notes
		Config	API	
ERCOSEK	4.x	✗	✓	1
RTA-OSEK	4.x	✗	✓	2
	5.x	✗	✓	2
RTA-TRACE	2.x	✓	✓	3
RTA-RTE2.x	1.x	✗	✓	4
ASCET	4.x	✗	✗	5
	5.x	✗	✓	6
	6.x	✗	✓	6

Notes on ETAS Tool compatibility

1. OSEK API calls are portable with the exceptions which have slightly modified behavior in AUTOSAR OS:
   - `StartOS()` does not return.
   - `SetRelAlarm()` cannot use zero as the offset parameter.
2. See Section 14.2 for specific details
3. RTA-OS3.0 adopts the AUTOSAR guidelines for namespaces in basic software modules for all internal names. However, the external names of all AUTOSAR OS R3.0 API calls, macros and type definitions are provided in the external API for compatibility with the AUTOSAR standard.

   Any RTA-OS3.0 functionality which is not part of the AUTOSAR OS R3.0 adopts the AUTOSAR naming convention for both internal and external names. For the OS this means:

   - Variable, API call names and constants are prefixed 0s_

- Macros are prefixed OS_

The AUTOSAR naming convention has also been applied to RTA-TRACE2.1 instrumentation code. While this does not change the behavior of RTA-TRACE2.1 and it transparent to users, it does mean that the documentation that ships with RTA-TRACE2.1 does not accurately reflect the names of API calls and types used in RTA-OS3.0. However, conversion between the documented names and those generated is trivial:

- All API calls, types and variables are prefixed Os_. For example:

  `LogTracepoint(MyTracepoint)`

  becomes

  `Os_LogTracepoint(MyTracepoint).`

- All macros are prefixed OS_. For example:

  `TRACE_ERRORS_CLASS`

  becomes

  `OS_TRACE_ERRORS_CLASS.`

4. RTA-RTE2.x generates an OS configuration in OIL that is compatible with AUTOSAR OS R2.x but this is not compatible with AUTOSAR OS R3.x. Most OS calls generated by RTA-RTE2.x are compatible with AUTOSAR OS R3.0. The API call `StartScheduleTable()` is used by the RTE library when integrating with an AUTOSAR OS R1.0. This call needs to be replaced by the `StartScheduleTableRel()` call to work with AUTOSAR OS R3.0. Full compatibility with the RTE is possible if the `OSENV_UNSUPPORTED` is defined instead. You should consult the *RTA-RTE2x Toolchain Integration Guide* for further details.

5. ASCET uses non-OSEK calls from the ERCOSEK API and is therefore not compatible with RTA-OS3.0.

6. ASCET uses the RTA-OSEK API such that generated code is compatible with RTA-OS3.0. The generated OIL file is not compatible with RTA-OS3.0 and will need to be converted to XML.

14.2 API Call Compatibility

The following table shows the compatibility between RTA-OS3.0 and the AUTOSAR OS R3.0 standard. In addition, compatibility between RTA-OS3.0 and the earlier RTA-OSEK family of operating systems (and compatibility with the OSEK OS and AUTOSAR OS R1.0 standards) is also shown.

API call	OSEK OS v2.2.x	RTA-OSEK v4.x	AUTOSAR R1.0 SC1	RTA-OSEK v5.x	AUTOSAR R3.0 SC1	AUTOSAR R3.0 SC2	AUTOSAR R3.0 SC3	AUTOSAR R3.0 SC4	RTA-OS3.0 v1.0	See Section
ActivateTask	✓	✓	✓	✓	✓	✓	✓	✓	✓	
ActivateTaskset		✓		✓						14.2.1
AdvanceSchedule		✓		✓						14.2.3
AssignTaskset		✓		✓						14.2.1
CallTrustedFunction							✓	✓		
CancelAlarm	✓	✓	✓	✓	✓	✓	✓	✓	✓	
ChainTask	✓	✓	✓	✓	✓	✓	✓	✓	✓	
ChainTaskset		✓		✓						14.2.1
CheckISRMemoryAccess							✓	✓		
CheckObjectAccess							✓	✓		
CheckObjectOwnership							✓	✓		
CheckTaskMemoryAccess							✓	✓		
ClearEvent	✓	✓	✓	✓	✓	✓	✓	✓	✓	
CloseCOM	✓	✓	✓	✓						14.2.4
DisableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
EnableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetActiveApplicationMode	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetAlarm	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetAlarmBase	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetApplicationID							✓	✓		
GetArrivalpointDelay		✓		✓						14.2.3
GetArrivalpointNext		✓		✓						14.2.3
GetArrivalpointTasksetRef		✓		✓						14.2.3
GetCounterValue		✓		✓	✓	✓	✓	✓	✓	
GetElapsedCounterValue					✓	✓	✓	✓	✓	
GetEvent	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetExecutionTime		✓		✓						14.2.2
GetISRID			✓	✓	✓	✓	✓	✓	✓	
GetLargestExecutionTime		✓		✓						14.2.2
GetMessageResource	✓	✓	✓	✓						14.2.4
GetMessageStatus	✓	✓	✓	✓						14.2.4
GetResource	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetScheduleNext		✓		✓						14.2.3
GetScheduleStatus		✓		✓						14.2.3
GetScheduleTableStatus		✓	✓	✓	✓	✓	✓	✓	✓	
GetScheduleValue		✓		✓						14.2.3
GetStackOffset		✓		✓						14.2.11
GetTaskID	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GetTasksetRef		✓		✓						14.2.1
GetTaskState	✓	✓	✓	✓	✓	✓	✓	✓	✓	
IncrementCounter			✓	✓	✓	✓	✓	✓	✓	

API call	OSEK OS v2.2.x	RTA-OSEK v4.x	AUTOSAR R1.0 SC1	RTA-OSEK v5.x	AUTOSAR R3.0 SC1	AUTOSAR R3.0 SC2	AUTOSAR R3.0 SC3	AUTOSAR R3.0 SC4	RTA-OS3.0 v1.0	See Section
InitCOM	✓	✓	✓	✓						14.2.4
InitCounter				✓						
MergeTaskset		✓		✓						14.2.1
NextScheduleTable			✓	✓	✓	✓	✓	✓	✓	
Os_AdvanceCounter									✓	
Os_GetExecutionTime									✓	14.2.2
Os_GetISRMaxExecutionTime									✓	14.2.2
Os_GetISRMaxStackUsage									✓	14.2.11
Os_GetStackUsage									✓	14.2.11
Os_GetStackValue									✓	14.2.11
Os_GetTaskMaxExecutionTime									✓	14.2.2
Os_GetTaskMaxStackUsage									✓	14.2.11
Os_ResetISRMaxExecutionTime									✓	14.2.2
Os_ResetISRMaxStackUsage									✓	14.2.11
Os_ResetTaskMaxExecutionTime									✓	14.2.2
Os_ResetTaskMaxStackUsage									✓	14.2.11
Os_Restart									✓	
Os_SetRestartPoint									✓	
osAdvanceCounter				✓						14.2.7
osResetOS				✓						14.2.12
ReadFlag	✓	✓	✓	✓						14.2.4
ReceiveMessage	✓	✓	✓	✓						14.2.4
ReleaseMessageResource	✓	✓	✓	✓						14.2.4
ReleaseResource	✓	✓	✓	✓	✓	✓	✓	✓	✓	
RemoveTaskset		✓		✓						14.2.1
ResetFlag	✓	✓	✓	✓						14.2.4
ResetLargestExecutionTime		✓		✓						14.2.2
ResumeAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
ResumeOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Schedule	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SendMessage	✓	✓	✓	✓						14.2.4
SetAbsAlarm	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SetArrivalpointDelay		✓		✓						14.2.3
SetArrivalpointNext		✓		✓						14.2.3
SetEvent	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SetRelAlarm	✓	✓	✓	✓	✓	✓	✓	✓	✓	14.2.8
SetScheduleNext		✓		✓						
SetScheduleTableAsync						✓		✓		
ShutdownOS	✓	✓	✓	✓	✓	✓	✓	✓	✓	14.2.6
StartCOM	✓	✓	✓	✓						14.2.4
StartOS	✓	✓	✓	✓	✓	✓	✓	✓	✓	14.2.5
StartSchedule		✓		✓						14.2.3

API call	OSEK OS v2.2.x	RTA-OSEK v4.x	AUTOSAR R1.0 SC1	RTA-OSEK v5.x	AUTOSAR R3.0 SC1	AUTOSAR R3.0 SC2	AUTOSAR R3.0 SC3	AUTOSAR R3.0 SC4	RTA-OS3.0 v1.0	See Section
StartScheduleTable			✓	✓						14.2.9
StartScheduleTableAbs					✓	✓	✓	✓	✓	
StartScheduleTableRel					✓	✓	✓	✓	✓	
StartScheduleTableSynchron						✓		✓		
StopCOM	✓	✓	✓	✓						14.2.4
StopSchedule		✓		✓						14.2.3
StopScheduleTable			✓	✓	✓	✓	✓	✓	✓	
SuspendAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SuspendOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SyncScheduleTable						✓		✓		
TerminateApplication							✓	✓		
TerminateTask	✓	✓	✓	✓	✓	✓	✓	✓	✓	
TestArrivalpointWriteable		✓		✓						14.2.3
TestEquivalentTaskset		✓		✓						14.2.1
TestSubTaskset		✓		✓						14.2.1
Tick_<CounterID>		✓		✓						14.2.10
TickSchedule		✓		✓						14.2.3
WaitEvent	✓	✓	✓	✓	✓	✓	✓	✓	✓	

14.2.1 Tasksets

RTA-OSEK taskset API. Tasksets are deprecated in RTA-OS3.0. The functionality can be implemented by executing multiple `ActivateTask()` calls in sequence. However, note that this only provides the same run-time behavior when the set of tasks that are activated are all of equal or lower priority than the task making the calls.

14.2.2 Time Monitoring

The RTA-OSEK timing build is replaced by the configuration option 'Time Monitoring'. This provides means that it is possible to use EXTENDED status without needing to provide stub implementations for time monitoring.

In RTA-OS3.0 the API calls are modified to take the `Os_` prefix but have identical behavior as the old RTA-OSEK calls. However, there are now specific calls for tasks and ISRs that replace the `GetLargestExecutionTime` and `GetLargestExecutionTime` calls.

- `Os_GetLargestExecutionTime` is replaced by `Os_Get[Task|ISR]MaxExecutionTime`

- `Os_ResetLargestExecutionTime` is replaced by `Os_Reset[Task|ISR]MaxExecutionTime`

14.2.3 Schedules

The RTA-OSEK schedule mechanism is replaced by AUTOSAR's ScheduleTable Mechanism. Note that it is not possible in RTA-OS3.0 to modify the schedule at runtime (this functionality is not supported by AUTOSAR OS). If runtime modification is required then alarms should be used instead.

14.2.4 OSEK COM

In OSEK OS, OSEK COM features may be provided by the OS (when OSEK COM is not used). This feature is deprecated in AUTOSAR OS R3.0 as internal communication for applications is provided by the AUTOSAR RTE.

14.2.5 Behavior of `StartOS()`

The `StartOS()` call may return in OSEK OS. This is the behavior provided in RTA-OSEK. In AUTOSAR OS this behavior is prohibited - the call *must not* return. This is the behavior provided by RTA-OS3.0. This means that it is no longer possible to use an idle loop placed after `StartOS()` as the idle mechanism. In RTA-OS3.0 the kernel will busy wait by default when there are no tasks and ISRs to run. You can replace this default behavior by providing a function called `Os_Cbk_Idle()` that implements your own idle (background task) functionality.

14.2.6 Behavior of `ShutdownOS()`

OSEK OS allows implementations to return from `ShutdownOS()`. In AUTOSAR OS `ShutdownOS()` must not return. RTA-OSEK always had the behavior specific by AUTOSAR OS, but it you are migrating from another OS then you may need to modify your application to reflect this change.

14.2.7 Hardware Counter Driver

The RTA-OSEK hardware counter driver call, `osAdvanceCounter()` is renamed `Os_AdvanceCounter` in RTA-OS3.0. The behavior of the call is also modified. In RTA-OSEK the call returned the status of the counter so the user could set up the next expiry. In RTA-OS3.0 this operation is performed internally (via a call to the user-provided `Os_Cbk_Set_CounetrID` API callback) for the first setup and application code must then call the `Os_Cbk_Status_CounterID` to check for multiple expiries.

14.2.8 Forbidding of Zero for `SetRelAlarm()`

`SetRelAlarm(_, 0, _)` is allowed in OSEK OS but forbidden in AUTOSAR OS.

14.2.9 Changes to Schedule Table API

The AUTOSAR OS standard has modified the call to start a schedule table so that the mechanism has the same concepts of absolute and relative start that are found with OSEK OS alarms. The API call `StartScheduleTable()` has been removed from the standard and is replaced by `StartScheduleTable[Rel|Abs]` in AUTOSAR R3.0. If you need to replicate the behavior of the `StartScheduleTable(Tbl,At)` call then you should used `StartScheduleTableRel(Tbl,At)`.

14.2.10 Software Counter Driver

The RTA-OSEK `Tick_CounterID()` calls have been replaced by AUTOSAR standard `IncrementCounter()` counter call which takes the `CounterID` as a parameter. However, to replicate the performance improvements that the 'static' version of the call provides, RTA-OS3.0 includes a 'static' version of the AUTOSAR call - `IncrementCounter_CounterID()` - which has identical behavior to the `Tick_CounterID()` call.

14.2.11 Stack Monitoring

The behavior of stack measurement is modified between RTA-OSEK and RTA-OS3.0. In RTA-OSEK stack measurements are made from the base address of the stack using the `GetStack0ffset()`. Typically the base address of the stack was given to RTA-OSEK at link time by defining label called `SP_INIT`.

In RTA-OS3.0 the `GetStack0ffset()` call is replaced by `0s_GetStackValue()`.

RTA-OSEK required you to calculate the amount of stack used by each task or ISR. You can still do this with RTA-OS3.0, but an additional API call, `0s_GetStackUsage()`, has been provided that returns the stack consumed by the calling task/ISR alone at the point of the call. The avoids the need to do any stack calculations yourself.

RTA-OS3.0 also logs the worst-case observed stack usage for each task/ISR when a context switch (or a call to `0s_GetStackUsage()` is made. Additional API calls are provided to get the largest observed stack usage for each task/ISR and to reset the largest observed value.

This model parallels the time monitoring functionality provided by RTA-OS3.0.

14.2.12 Restarting the OS

Neither OSEK OS or AUTOSAR OS provide facilities to re-start the OS at run-time. As this is commonly required functionality, RTA-OSEK provided a the `osReset0S()` API call that allowed a restart to be performed.

In RTA-OS3.0 this is replaced by a general-purpose restarting mechanism. The API call `Os_SetRestartPoint` is provided that can be made anywhere before you call `StartOS()` to place a marker from where the restart should happen. This means you can re-initialize any hardware required before the call to `StartOS()`. A restart is then achieved by calling `Os_Restart` which jumps to the marker you have set.

15 Contacting ETAS

15.1 Technical Support

Technical support is available to all RTA-OS3.0 users with a valid support contract. If you do not have such a contract then please contact ETAS through one of the addresses listed in Section 15.2.

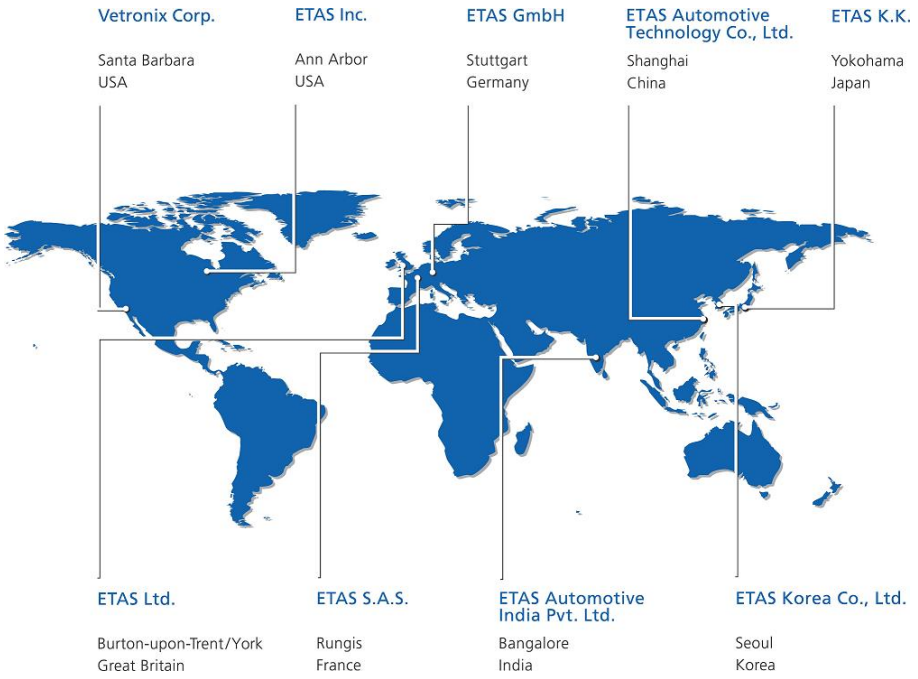
The best way to get technical support is by email. Any problems or questions should be sent to: rta.hotline.uk@etas.com

It is helpful if you can provide support with the following information:

- your support contract number.
- your .xml/.rtaos configuration files.
- the error message you received and the file `Diagnostic.dmp` if it was generated.
- the command line that results in an error message.
- the version of the ETAS tools you are using.
- the version of your compiler tool chain you are using.

If you prefer to discuss your problem with the technical support team you can contact them by telephone during normal office hours (0900-1730 GMT/BST). The telephone number for the RTA-OS3.0 support hotline is: +44 (0)1904 562624.

15.2 General Enquiries



Europe

Excluding France, Belgium, Luxembourg, United Kingdom and Scandinavia

ETAS GmbH

Borsigstrasse 14	Phone:	+49 711 89661-0
70469 Stuttgart	Fax:	+49 711 89661-300
Germany	E-mail:	sales.de@etas.com
	WWW:	www.etas.com

France, Belgium and Luxemburg

ETAS S.A.S.

1, place des États-Unis	Phone:	+33 1 56 70 00 50
SILIC 307	Fax:	+33 1 56 70 00 51
94588 Rungis Cedex	E-mail:	sales.fr@etas.com
France	WWW:	www.etas.com

United Kingdom and Scandinavia

ETAS Ltd.

Studio 3, Waterside Court	Phone:	+44 1283 54 65 12
Third Avenue, Centrum 100	Fax:	+44 1283 54 87 67
Burton-upon-Trent	E-mail:	sales.uk@etas.com
Staffordshire DE14 2WQ	WWW:	www.etas.com
United Kingdom		

USA

ETAS Inc.		
3021 Miller Road	Phone:	+1 888 ETAS INC
Ann Arbor	Fax:	+1 734 997-9449
MI 48103	E-mail:	sales.us@etas.com
USA	WWW:	www.etas.com

Japan

ETAS K.K.		
Queen's Tower C-17F	Phone:	+81 45 222-0900
2-3-5, Minatomirai, Nishi-ku	Fax:	+81 45 222-0956
Yokohama 220-6217	E-mail:	sales.jp@etas.com
Japan	WWW:	www.etas.com

Korea

ETAS Korea Co. Ltd.		
4F, 705 Bldg. 70-5	Phone:	+82 2 5747-016
Yangjae-dong, Seocho-gu	Fax:	+82 2 5747-120
Seoul 137-889	E-mail:	sales.kr@etas.com
Korea	WWW:	www.etas.com

P.R.China

ETAS (Shanghai) Co., Ltd.		
2404 Bank of China Tower	Phone:	+86 21 5037 2220
200 Yincheng Road Central	Fax:	+86 21 5037 2221
Shanghai 200120	E-mail:	sales.cn@etas.com
P.R. China	WWW:	www.etas.com

India

ETAS Automotive India Pvt. Ltd.		
No. 690, Gold Hill Square, 12F	Phone:	+91 80 4191 2585
Hosur Road, Bommanahalli	Fax:	+91 80 4191 2586
Bangalore, 560 068	E-mail:	sales.in@etas.com
India	WWW:	www.etas.com

Index