Document Title	Technical Report on Operating System Tracing Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1083

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R25-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2025-11-27	R25-11	AUTOSAR Release Management	• No content changes
2024-11-27	R24-11	AUTOSAR Release Management	• Removed 'draft' state from Specification Items • Correction of API syntax (`ArtiVersionInfoType`, `CallingContext`) • Minor editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction

This technical report provides additional information to the Operating System Tracing Interface of the AUTOSAR Standard.

1.1 Objectives

The goal is to provide an API that can be used at a very low level to trace tasks and processes. It is at a very low level to have no or minimal impact on the runtime behavior of the application. The recorded information is used to determine timing information of the software.

Based on the timing information, the timing requirements, such as CPU time, deadlines, accuracy of periodicity can be analyzed. In addition, time consumption can be broken down to specific parts of the application, and timing dependencies and locks can be shown.

1.2 Scope

This report is related to the operating system of the adaptive platform. The API is used by stack and trace tool vendors. It is not intended to be used by an application engineer.

The API is intended to be used at driver level of the operating system. Processes and tasks cannot be traced at application level or middleware level because this would influence the runtime behavior of the system.

2 Definition of terms and acronyms

2.1 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
Adaptive Application	see [1] AUTOSAR Glossary
ARTI	see [1] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [1] AUTOSAR Glossary
Executable	see [1] AUTOSAR Glossary
Execution Management [2]	The element of the `AUTOSAR Adaptive Platform` responsible for the ordered startup and shutdown of the `AUTOSAR Adaptive Platform` and `Adaptive Applications`.
Execution Manifest	`Manifest` file to configure execution of an `Adaptive Application`. An `Execution Manifest` is created at integration time and deployed onto a `Machine` together with the `Executable` to which it is attached. It supports the integration of the `Executable` code and describes the configuration properties (startup parameters, resource group assignment etc.) of each `Process`, i.e. started instance of that `Executable`.
Machine	see [1] AUTOSAR Glossary
Manifest	see [1] AUTOSAR Glossary
Modelled Process	A `Modelled Process` is an instance of an `Executable` to be executed on a `Machine` and has a 1:1 association with the ARXML/Meta-Model element `Process`. This document also uses the term process (without the “modelled” prefix) to refer to the OS concept of a running process.
Operating System	Software responsible for managing `Processes` on a `Machine` and for providing an interface to hardware resources.
Process	see [1] AUTOSAR Glossary
Task	see [1] AUTOSAR Glossary In case of POSIX a task is called thread.

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related Documentation

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

4 Functional Specification

4.1 ARTI Tracing Interface

4.1.1 OS/ARTI Adapter

The so-called “OS/ARTI Adapter” provides the trace points at OS level. It is used to understand, verify and visualize the timing behavior of the OS. The ARTI trace hooks themselves form a standardized interface that is specified by the API below.

Figure 4.1 illustrates the Layout of the OS/ARTI driver containing the OS/ARTI Adapter.

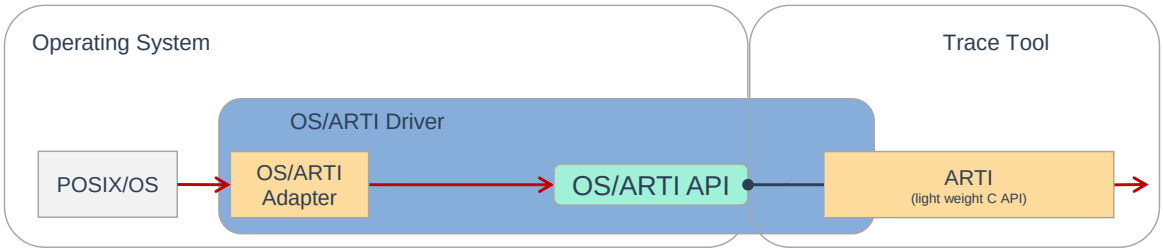

Figure 4.1: Layout of the OS/ARTI Driver

The implementation of the ARTI hooks themselves depends on the tracing mechanism and shall be provided by the tracing tool vendor.

The ARTI hook interface is designed to be usable as a C macro expansion or as a C function. If no tracing mechanism is available, the ARTI hooks may be expanded to nothing (in case of a macro) or call an empty function.

The ARTI interface follows the two-level approach of AUTOSAR, where a “task” is a schedulable unit (in OSes often called “thread”), and a “process” is a mandatory environment holding several tasks. An example system is shown in Figure 4.2.

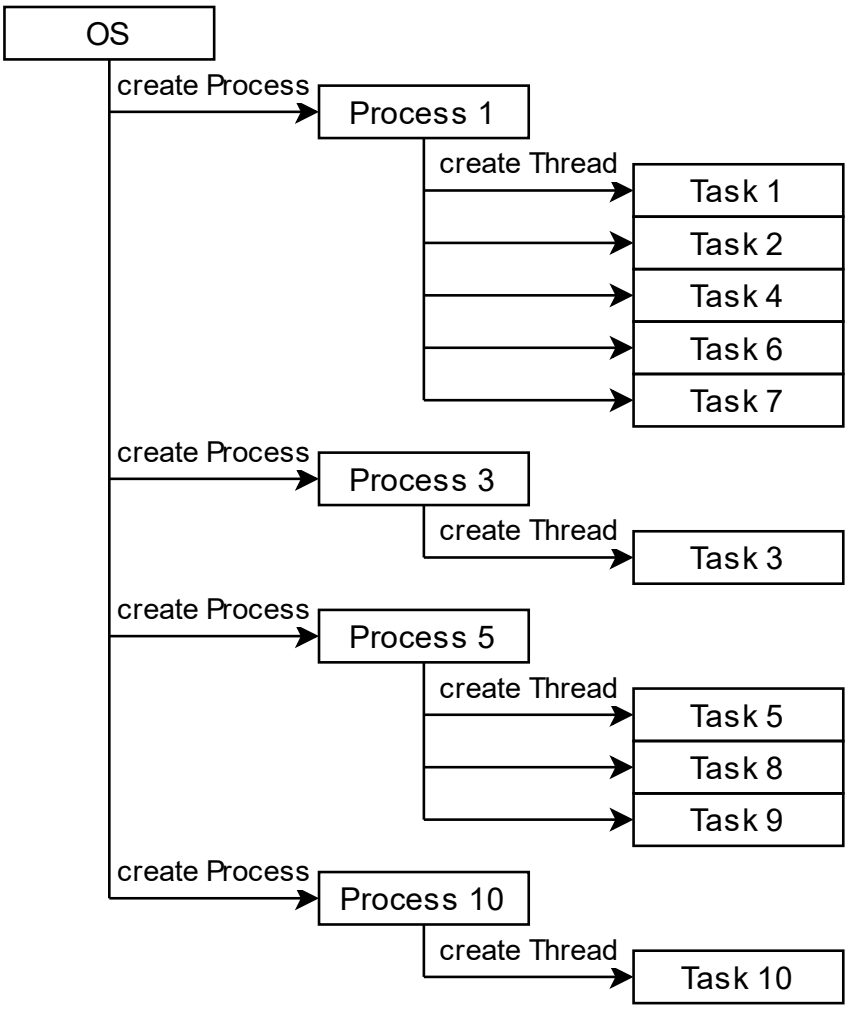

Figure 4.2: Example of Process - Task/Thread Model

An ARTI interface carries some of these parameters:

- callingContext: type CallingContext represents the current interrupt handling.
  - kInterruptsDisabled indicates that the hook gets called in a context where interrupts are disabled,
  - kInterruptsMayBeDisabled indicates that the called hook may disable interrupts,
  - kInterruptsMayNotBeDisabled indicates that the called hook cannot disable interrupts
- coreId: type uint32_t, specifies the ID of the core where the event happens
- taskId: type uint32_t, specifies the task ID of the task belonging to the hook
- processId: type uint32_t, specifies the process ID of the process belonging to the hook

Both taskId and processId are IDs representing a task or a process within the OS-/ARTI API. A taskId or processId is used by ARTI over a tracing run and is derived from the OS internal task or process ID. The derivation is a not specified implementation detail and should closely match the OS internal ID. The meaning of these IDs can be derived from the task/process name given by ArtiTaskInfo/ArtiProcessInfo or ArtiTaskRename/ArtiProcessRename. The processId can be mapped by a trace tool to AUTOSAR Adaptive Platform Modelled Processes using the Execution Manifest when also ExecutionManagerProccessStateChangeMsg messages of the Execution Management are traced.

4.1.1.1 Adapter Management

The following interfaces are used for managing the OS/ARTI Adapter.

[TR_OSTI_00001] ARTI Version Info

Upstream requirements: RS_OSI_00210

⌈If ARTI is used then the OS/ARTI Adapter shall call ArtiVersionInfo when the OS/ARTI Adapter is started in the system.

- The parameter callingContext shall be set to the CallingContext which represents the current interrupt handling.
- The parameter versionInfoPtr shall be set to the ArtiVersionInfoType provided by the OS.

It is used to confirm the version of API between OS and ARTI-driver.⌋

The OS/ARTI Adapter shall call this function just before ArtiInit is called. It is used to assure the compatibility of the OS and the ARTI-driver whereby the apiVersion of the OS and the returned apiVersion of the ARTI-driver must be equal for further using these hooks. When this function is called, versionInfoPtr is filled with the OS related values. The versionInfoPtr->apiVersion is filled by the OS with the highest supported version of the OS. The driver returns a pointer to a filled ArtiVersionInfoType with the values of the ARTI-driver. The returned apiVersion should be adapted to the version of the OS if possible. If this is not possible, then the highest supported version of the driver is filled. When the apiVersion of OS and ARTI-driver are

- identical, then tracing is possible and can start with ArtiInit
- OS apiVersion is higher than ARTI-driver apiVersion, then the OS checks whether this is also supported. In this case it calls ArtiVersionInfo again with an adapted major version. If it is not supported then there is a mismatch and tracing can not happen.
- OS apiVersion is lower than ARTI-driver apiVersion, then tracing is not possible.

ArtiVersionInfo is called once or twice. The ARTI-driver knows whether trace is possible when ARTI-driver returned the same apiVersion that it got from the OS.

[TR_OSTI_00002] ARTI Initialisation

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is used then the OS/ARTI Adapter shall call `ArtiInit` right after the version of API is being confirmed.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.

It may be used to initialize the trace driver implementing the adapter.⌋

[TR_OSTI_00003] ARTI Cleanup

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is used then the OS/ARTI Adapter shall call `ArtiCleanup` when the OS/ARTI Adapter is stopped.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.

⌋

4.1.1.2 Task Interface

The term `Task` applies to the object as defined in the AUTOSAR Glossary: “A Task is the smallest schedulable unit managed by the OS. The OS decides when which task can run on the CPU of the ECU.”

The trace events of a task shall follow the state machine in Figure 4.3.

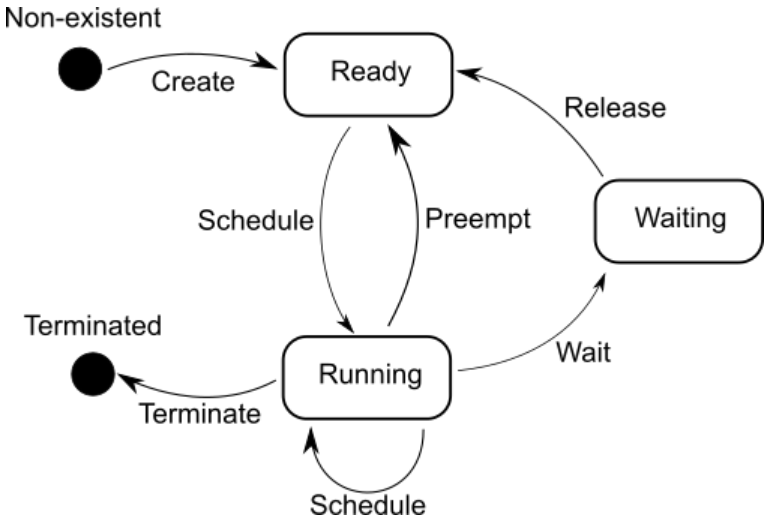

Figure 4.3: Minimal state machine of a task

The minimal state machine for a single task has the states:

Ready The task is ready and can be scheduled for running.

Running The task is being executed.

Waiting The task is waiting for an event, semaphore, a different thread or different OS object. The task can not be scheduled for running.

For an OS that does not support or differentiate between Ready state and Waiting state, the `ARTI` trace hooks for tracing switches between Ready and Running shall be mandatory, and `ARTI` trace hooks for switching to Waiting state are optional.

Hooks to be called on events related to tasks:

[TR_OSTI_00004] ARTI Task Switch Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter shall call `ArtiTaskSwitch` whenever an OS task enters the running state.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the current task is scheduled on.
- The parameter `nextId` shall be set to the operating system specific task ID of the next task.

⌋

On a single CPU there can be only one task in running state. The other tasks have to be terminated or have to be in waiting or ready state. This implies that at a task switch the previous task that was running left the running state and the OS/ARTI Adapter called the related API `ArtiTaskWait`, `ArtiTaskPreempt` or `ArtiTaskExit` before.

[TR_OSTI_00005] ARTI Task Wait Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter should call `ArtiTaskWait` whenever an OS task is entering waiting state.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the task is scheduled on.
- The parameter `taskId` shall be set to the operating system specific task ID of the task.

⌋

[TR_OSTI_00006] ARTI Task Release Notification

Upstream requirements: RS_OSI_00210

⌈If ARTI is enabled then the OS/ARTI Adapter should call ArtiTaskRelease whenever an OS task state changes from waiting to ready.

- The parameter callingContext shall be set to the CallingContext which represents the current interrupt handling.
- The parameter coreId shall be set to the coreId the task is scheduled on.
- The parameter taskId shall be set to the operating system specific task ID of the task.

⌋

[TR_OSTI_00007] ARTI Task Preempt Notification

Upstream requirements: RS_OSI_00210

⌈If ARTI is enabled then the OS/ARTI Adapter should call ArtiTaskPreempt whenever an OS task state changes from running to ready.

- The parameter callingContext shall be set to the CallingContext which represents the current interrupt handling.
- The parameter coreId shall be set to the coreId the task is scheduled on.
- The parameter taskId shall be set to the operating system specific task ID of the task.

⌋

[TR_OSTI_00008] ARTI Task Exit Notification

Upstream requirements: RS_OSI_00210

⌈If ARTI is enabled then the OS/ARTI Adapter shall call ArtiTaskExit whenever an OS task terminates.

- The parameter callingContext shall be set to the CallingContext which represents the current interrupt handling.
- The parameter coreId shall be set to the coreId the task is scheduled on.
- The parameter taskId shall be set to the operating system specific task ID of the task.

⌋

[TR_OSTI_00009] ARTI Task Creation Notification

Upstream requirements: RS_OSI_00210

⌈If ARTI is enabled then the OS/ARTI Adapter shall call ArtiTaskCreate whenever an OS task is created.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the task is scheduled on.
- The parameter `processId` shall be set to the operating system specific process ID of the process that is the parent of the task.
- The parameter `taskId` shall be set to the operating system specific task ID of the task that is being created.

⌋

[TR_OSTI_00010] ARTI Task Renaming Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter should call `ArtiTaskRename` whenever an OS task is named or renamed.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `taskId` shall be set to the operating system specific task ID of the task.
- The parameter `taskName` shall be set to the operating system specific task name.

⌋

Additional interfaces to tasks:

[TR_OSTI_00011] ARTI Task Information Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter shall call `ArtiTaskInfo` for each existing task directly after calling `ArtiInit` or whenever tracing is started.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `taskId` shall be set to the operating system specific task ID of the task.
- The parameter `processId` shall be set to the operating system specific process ID of the process that is the parent of the task.
- The parameter `taskName` shall be set to the operating system specific task name.

This function provides information about task name and parent process. This will build up the initial task list.⌋

4.1.1.3 Process Interface

The term `Process` applies to the object as defined in the AUTOSAR Glossary: “An executable unit managed by an operating system scheduler that has its own name space and resources (including memory) protected against the use by other processes.”

Hooks to be called on events related to processes:

[TR_OSTI_00012] ARTI Process Switch Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter should call `ArtiProcessSwitch` whenever an OS process switch happens.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the current process is scheduled on.
- The parameter `nextId` shall be set to the operating system specific process ID of the next process.

⌋

[TR_OSTI_00013] ARTI Process Creation Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter shall call `ArtiProcessCreate` whenever an OS process is created.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the process is scheduled on.
- The parameter `processId` shall be set to the operating system specific process ID of the process that is being created.
- If there is a parent process then the parameter `parentId` shall be set to the operating system specific process ID of the process that is the parent of the process created otherwise it shall be set to the operating system specific process ID that is created.

⌋

[TR_OSTI_00014] ARTI Process Destroy Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter shall call `ArtiProcessDestroy` whenever an OS process ends.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `coreId` shall be set to the coreId the process is scheduled on.
- The parameter `processId` shall be set to the operating system specific process ID of the process.

⌋

[TR_OSTI_00015] ARTI Process Renaming Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter should call `ArtiProcessRename` whenever an OS process is named or renamed.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `processId` shall be set to the operating system specific process ID of the process.
- The parameter `processName` shall be set to the operating system specific process name.

⌋

Additional interfaces to processes:

[TR_OSTI_00016] ARTI Process Information Notification

Upstream requirements: RS_OSI_00210

⌈If `ARTI` is enabled then the OS/ARTI Adapter should call `ArtiProcessInfo` for each existing process directly after calling `ArtiInit` or whenever tracing is started.

- The parameter `callingContext` shall be set to the `CallingContext` which represents the current interrupt handling.
- The parameter `processId` shall be set to the operating system specific process ID of the process.
- The parameter `parentId` shall be set to the operating system specific process ID of the parent process.
- The parameter `processName` shall be set to the operating system specific process name.

This function provides information about process name and parent process. This will build up the initial process list.⌋

5 API Specification

5.1 Type Definitions

5.1.1 ArtiVersionInfoType

[TR_OSTI_00516] Definition of API class ArtiVersionInfoType

Upstream requirements: RS_OSI_00210

⌈

Kind:	struct
Header file:	#include "ara/log/osarti.h"
Forwarding header file:	#include "ara/log/log_fwd.h"
Symbol:	ArtiVersionInfoType
Syntax:	`typedef struct {...} ArtiVersionInfoType;`
Description:	Holds information about the ARTI version supported by the OS and the ARTI-driver.

⌋

[TR_OSTI_00518] Definition of API variable ArtiVersionInfoType::apiVersion

Upstream requirements: RS_OSI_00210

⌈

Kind:	variable
Header file:	#include "ara/log/osarti.h"
Symbol:	apiVersion
Type:	`uint32_t`
Syntax:	`uint32_t apiVersion;`
Description:	The version of the API. As an input parameter, it specifies the requested version of the API. As an output parameter, it indicates the supported version of the ARTI-driver. Set to 0x01 for the current AUTOSAR release.

⌋

[TR_OSTI_00519] Definition of API variable ArtiVersionInfoType::buildVersion

Upstream requirements: RS_OSI_00210

⌈

Kind:	variable
Header file:	#include "ara/log/osarti.h"
Symbol:	buildVersion
Type:	`uint32_t`
Syntax:	`uint32_t buildVersion;`
Description:	The version of the driver. This is an informal parameter. As an input parameter, it is the build version of the OS part of the driver or the OS build version. As an output parameter, it is the build version of the ARTI driver.

⌋

[TR_OSTI_00521] Definition of API variable ArtiVersionInfoType::productName

Upstream requirements: RS_OSI_00210

⌈

Kind:	variable
Header file:	#include "ara/log/osarti.h"
Symbol:	productName
Type:	`const char *`
Syntax:	`const char* productName;`
Description:	The product name of the implementation. This is an informal parameter. As an input parameter, it is the name of the OS. As an output parameter, it is the name of the ARTI driver.

⌋

[TR_OSTI_00520] Definition of API variable ArtiVersionInfoType::vendorName

Upstream requirements: RS_OSI_00210

⌈

Kind:	variable
Header file:	#include "ara/log/osarti.h"
Symbol:	vendorName
Type:	`const char *`
Syntax:	`const char* vendorName;`
Description:	The vendor name. This is an informal parameter. As an input parameter, it is the name of the OS vendor. As an output parameter, it is the name of the ARTI driver vendor.

⌋

5.1.2 CallingContext

[TR_OSTI_00515] Definition of API enum CallingContext

Upstream requirements: RS_OSI_00210

⌈

Kind:	enumeration	
Header file:	#include "ara/log/osarti.h"	
Forwarding header file:	#include "ara/log/log_fwd.h"	
Symbol:	CallingContext	
Underlying type:	--	
Syntax:	`typedef enum {...} CallingContext;`	
Values:	kInterruptsDisabled= 0	Indicates that the hook is called in a context where interrupts are disabled.
	kInterruptsMayBe Disabled= 1	Indicates that the called hook may disable interrupts.

▽

△

	kInterruptsMayNotBe Disabled= 2	Indicates that the called hook cannot disable interrupts.
Description:	Specifies whether interrupts are disabled or can be disabled.	

⌋

5.2 Callback Notifications

This is a list of functions provided for other modules.

5.2.1 ArtiTaskSwitch

[TR_OSTI_00502] Definition of API function ArtiTaskSwitch

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskSwitch (CallingContext callingContext, uint32_t coreId, uint32_t nextId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core that switches the task.
	nextId	The ID of the task that enters the running state.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer about a task switch. The OS/ARTI Adapter should call this hook when a task enters the running state. This implies that the previous task on this core, which was in the running state, enters the ready state (preemption).	

⌋

5.2.2 ArtiTaskWait

[TR_OSTI_00503] Definition of API function ArtiTaskWait

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskWait (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.

▽

△

	coreId	The ID of the core on which the task is running.
	taskId	The ID of the task that is entering the wait state.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer that a task is entering the wait state. The OS/ARTI Adapter should call this hook when a task is entering the wait state.	

⌋

5.2.3 ArtiTaskRelease

[TR_OSTI_00504] Definition of API function ArtiTaskRelease

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskRelease (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core on which the task is running.
	taskId	The ID of the task that is leaving the wait state.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer that a task is leaving the wait state and entering the ready state. The OS/ARTI Adapter should call this hook when a task is leaving the wait state and entering the ready state.	

⌋

5.2.4 ArtiTaskPreempt

[TR_OSTI_00505] Definition of API function ArtiTaskPreempt

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskPreempt (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core on which the task is running.
	taskId	The ID of the task that is leaving the running state.
Return value:	None	

▽

△

Thread Safety:	reentrant
Description:	Notifies the tracer that a task is leaving the running state and entering the ready state. The OS/ARTI Adapter should call this hook when a task is leaving the running state and entering the ready state.

⌋

5.2.5 ArtiTaskExit

[TR_OSTI_00506] Definition of API function ArtiTaskExit

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskExit (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core where the task is exiting.
	taskId	The ID of the task that is exiting.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer about a task exit. The OS/ARTI Adapter should call this hook when a task is terminated.	

⌋

5.2.6 ArtiTaskCreate

[TR_OSTI_00507] Definition of API function ArtiTaskCreate

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskCreate (CallingContext callingContext, uint32_t coreId, uint32_t processId, uint32_t taskId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core that creates the task.
	processId	The ID of the process creating the new task.
	taskId	The ID of the task that is being created.
Return value:	None	
Thread Safety:	reentrant	

▽

△

Description:	Notifies the tracer about the creation of a task. The OS/ARTI Adapter should call this at the time when the OS creates a new task.

⌋

5.2.7 ArtiTaskRename

[TR_OSTI_00508] Definition of API function ArtiTaskRename

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskRename (CallingContext callingContext, uint32_t taskId, const char *taskName);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	taskId	The ID of the task that is being renamed.
	taskName	The name that has to be assigned to the task. The string has to be null-terminated.
Return value:	None	
Thread Safety:	reentrant	
Description:	Provides a name for a task. This function allows users to identify a specific task by its name. The OS/ARTI Adapter should call this function to provide a task name for a taskId.	

⌋

5.2.8 ArtiTaskInfo

[TR_OSTI_00509] Definition of API function ArtiTaskInfo

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiTaskInfo (CallingContext callingContext, uint32_t taskId, uint32_t processId, const char *taskName);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	taskId	The ID of the task for which information is provided.
	processId	The ID of the process that owns this task.
	taskName	The name of the task. The string has to be null-terminated.
Return value:	None	
Thread Safety:	reentrant	

▽

△

Description:	Provides information about an existing task. This function provides information about the task name and parent process. The OS/ARTI Adapter should call this function for each existing task directly after calling ArtiInit(), or whenever tracing is started. This will build up the initial task list.

⌋

5.2.9 ArtiProcessSwitch

[TR_OSTI_00510] Definition of API function ArtiProcessSwitch

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiProcessSwitch (CallingContext callingContext, uint32_t coreId, uint32_t nextId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core that switches the process.
	nextId	The ID of the process that gets the CPU resources.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer about a process switch. This hook is called when the CPU resources are switched to another process. Usually, this information can be derived from a task switch. The OS/ARTI Adapter should call this hook when a process is switched.	

⌋

5.2.10 ArtiProcessCreate

[TR_OSTI_00511] Definition of API function ArtiProcessCreate

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiProcessCreate (CallingContext callingContext, uint32_t coreId, uint32_t processId, uint32_t parentId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core that creates the process.
	processId	The ID of the process that is being created.
	parentId	Optional ID of the parent process. If parentId == processId, then parentId is not used.
Return value:	None	

▽

△

Thread Safety:	reentrant
Description:	Notifies the tracer about the creation of a process. The OS/ARTI Adapter should call this at the time when the OS creates a new process.

⌋

5.2.11 ArtiProcessDestroy

[TR_OSTI_00512] Definition of API function ArtiProcessDestroy

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiProcessDestroy (CallingContext callingContext, uint32_t coreId, uint32_t processId);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	coreId	The ID of the core that destroys the memory context.
	processId	The ID of the process that is to be destroyed.
Return value:	None	
Thread Safety:	reentrant	
Description:	Notifies the tracer about the destruction of a process. The OS/ARTI Adapter should call this hook when the process is destroyed.	

⌋

5.2.12 ArtiProcessRename

[TR_OSTI_00513] Definition of API function ArtiProcessRename

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiProcessRename (CallingContext callingContext, uint32_t processId, const char *processName);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	processId	The ID of the process that is being renamed.
	processName	The name that has to be assigned to the process. The string has to be null-terminated.
Return value:	None	
Thread Safety:	reentrant	

▽

△

Description:	Provides a name for a process. This name is needed to identify a certain process by the user. The OS/ARTI Adapter should call this function to provide a process name.

⌋

5.2.13 ArtiProcessInfo

[TR_OSTI_00514] Definition of API function ArtiProcessInfo

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiProcessInfo (CallingContext callingContext, uint32_t processId, uint32_t parentId, const char *processName);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	processId	The ID of the process for which information is provided.
	parentId	The ID of the parent process.
	processName	The name of the process. The string has to be null-terminated.
Return value:	None	
Thread Safety:	reentrant	
Description:	Provides information about an existing process. This function provides information about the process name and parent process. The OS/ARTI Adapter should call this function for each existing process directly after calling ArtiInit(), or whenever tracing is started. This will build up the initial process list.	

⌋

5.2.14 ArtiVersionInfo

[TR_OSTI_00517] Definition of API function ArtiVersionInfo

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`ArtiVersionInfoType const *const ArtiVersionInfo (CallingContext callingContext, ArtiVersionInfoType const *const versionInfoPtr);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
	versionInfoPtr	Constant pointer to a constant `ArtiVersionInfoType` holding the values of the operating system.
Return value:	ArtiVersionInfoType const *const	Constant pointer to a constant `ArtiVersionInfoType` holding the the values of the ARTI-driver.
Thread Safety:	reentrant	

▽

△

Description:	Assures compatibility between the OS and the ARTI-driver. The OS/ARTI Adapter should call this function just before `ArtiInit` is called. It is used to assure the compatibility of the OS and the ARTI-driver. The apiVersion of the OS and the returned apiVersion of the ARTI-driver must be equal for further using these hooks. When this function is called, versionInfoPtr is filled with the OS-related values. The versionInfo Ptr->apiVersion is filled by the OS with the highest supported version of the OS. The driver returns a pointer to a filled `ArtiVersionInfoType` with the values of the ARTI-driver. The returned apiVersion should be adapted to the version of the OS if possible. If this is not possible, then the highest supported version of the driver is filled. When the apiVersion of the OS and the ARTI-driver are: • Identical, then tracing is possible and can start with `ArtiInit`. • OS apiVersion is higher than the ARTI-driver apiVersion, then the OS checks whether this is also supported. In this case, it calls ArtiVersionInfo again with an adapted major version. If it is not supported, then there is a mismatch and tracing cannot happen. • OS apiVersion is lower than the ARTI-driver apiVersion, then tracing is not possible. ArtiVersionInfo is called once or twice. The ARTI-driver knows whether tracing is possible when the ARTI-driver returns the same apiVersion that it got from the OS.

⌋

5.2.15 ArtiInit

[TR_OSTI_00500] Definition of API function ArtiInit

Upstream requirements: RS_OSI_00210

⌈

Kind:	function	
Header file:	#include "ara/log/osarti.h"	
Syntax:	`void ArtiInit (CallingContext callingContext);`	
Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
Return value:	None	
Thread Safety:	reentrant	
Description:	Initializes the OS/ARTI Adapter. The OS/ARTI Adapter should call this function when it is started in the system. It may be used to initialize the trace driver implementing the adapter.	

⌋

5.2.16 ArtiCleanup

[TR_OSTI_00501] Definition of API function ArtiCleanup

Upstream requirements: RS_OSI_00210

⌈

Kind:	function
Header file:	#include "ara/log/osarti.h"
Syntax:	`void ArtiCleanup (CallingContext callingContext);`

▽

△

Parameters (in):	callingContext	Specifies whether interrupts are disabled or can be disabled.
Return value:	None	
Thread Safety:	reentrant	
Description:	Cleans up the OS/ARTI Adapter. The OS/ARTI Adapter should call this function when it is stopped. It may be used to free local memory or flush pending messages.	

⌋

A Change History

A.1 Change History of this document according to AUTOSAR Release R25-11

A.1.1 Added Specification Items in R25-11

none

A.1.2 Changed Specification Items in R25-11

none

A.1.3 Deleted Specification Items in R25-11

none

A.2 Change History of this document according to AUTOSAR Release R24-11

A.2.1 Added Specification Items in R24-11

none

A.2.2 Changed Specification Items in R24-11

Number	Heading
[TR_OSTI_00001]	ARTI Version Info
[TR_OSTI_00002]	ARTI Initialisation
[TR_OSTI_00003]	ARTI Cleanup
[TR_OSTI_00004]	ARTI Task Switch Notification
[TR_OSTI_00005]	ARTI Task Wait Notification
[TR_OSTI_00006]	ARTI Task Release Notification
[TR_OSTI_00007]	ARTI Task Preempt Notification
[TR_OSTI_00008]	ARTI Task Exit Notification
[TR_OSTI_00009]	ARTI Task Creation Notification
[TR_OSTI_00010]	ARTI Task Renaming Notification

△

Number	Heading
[TR_OSTI_00011]	ARTI Task Information Notification
[TR_OSTI_00012]	ARTI Process Switch Notification
[TR_OSTI_00013]	ARTI Process Creation Notification
[TR_OSTI_00014]	ARTI Process Destroy Notification
[TR_OSTI_00015]	ARTI Process Renaming Notification
[TR_OSTI_00016]	ARTI Process Information Notification
[TR_OSTI_00500]	Definition of API function ArtiInit
[TR_OSTI_00501]	Definition of API function ArtiCleanup
[TR_OSTI_00502]	Definition of API function ArtiTaskSwitch
[TR_OSTI_00503]	Definition of API function ArtiTaskWait
[TR_OSTI_00504]	Definition of API function ArtiTaskRelease
[TR_OSTI_00505]	Definition of API function ArtiTaskPreempt
[TR_OSTI_00506]	Definition of API function ArtiTaskExit
[TR_OSTI_00507]	Definition of API function ArtiTaskCreate
[TR_OSTI_00508]	Definition of API function ArtiTaskRename
[TR_OSTI_00509]	Definition of API function ArtiTaskInfo
[TR_OSTI_00510]	Definition of API function ArtiProcessSwitch
[TR_OSTI_00511]	Definition of API function ArtiProcessCreate
[TR_OSTI_00512]	Definition of API function ArtiProcessDestroy
[TR_OSTI_00513]	Definition of API function ArtiProcessRename
[TR_OSTI_00514]	Definition of API function ArtiProcessInfo
[TR_OSTI_00515]	Definition of API enum CallingContext
[TR_OSTI_00516]	Definition of API class ArtiVersionInfoType
[TR_OSTI_00517]	Definition of API function ArtiVersionInfo
[TR_OSTI_00518]	Definition of API variable ArtiVersionInfoType::apiVersion
[TR_OSTI_00519]	Definition of API variable ArtiVersionInfoType::buildVersion
[TR_OSTI_00520]	Definition of API variable ArtiVersionInfoType::vendorName
[TR_OSTI_00521]	Definition of API variable ArtiVersionInfoType::productName

Table A.1: Changed Specification Items in R24-11

A.2.3 Deleted Specification Items in R24-11

none

A.3 Change History of this document according to AUTOSAR Release R23-11

A.3.1 Added Specification Items in R23-11

Number	Heading
[TR_OSTI_00001]	ARTI Version Info
[TR_OSTI_00002]	ARTI Initialisation
[TR_OSTI_00003]	ARTI Cleanup
[TR_OSTI_00004]	ARTI Task Switch Notification
[TR_OSTI_00005]	ARTI Task Wait Notification
[TR_OSTI_00006]	ARTI Task Release Notification
[TR_OSTI_00007]	ARTI Task Preempt Notification
[TR_OSTI_00008]	ARTI Task Exit Notification
[TR_OSTI_00009]	ARTI Task Creation Notification
[TR_OSTI_00010]	ARTI Task Renaming Notification
[TR_OSTI_00011]	ARTI Task Information Notification
[TR_OSTI_00012]	ARTI Process Switch Notification
[TR_OSTI_00013]	ARTI Process Creation Notification
[TR_OSTI_00014]	ARTI Process Destroy Notification
[TR_OSTI_00015]	ARTI Process Renaming Notification
[TR_OSTI_00016]	ARTI Process Information Notification
[TR_OSTI_00500]	Definition of API function ArtiInit
[TR_OSTI_00501]	Definition of API function ArtiCleanup
[TR_OSTI_00502]	Definition of API function ArtiTaskSwitch
[TR_OSTI_00503]	Definition of API function ArtiTaskWait
[TR_OSTI_00504]	Definition of API function ArtiTaskRelease
[TR_OSTI_00505]	Definition of API function ArtiTaskPreempt
[TR_OSTI_00506]	Definition of API function ArtiTaskExit
[TR_OSTI_00507]	Definition of API function ArtiTaskCreate
[TR_OSTI_00508]	Definition of API function ArtiTaskRename
[TR_OSTI_00509]	Definition of API function ArtiTaskInfo
[TR_OSTI_00510]	Definition of API function ArtiProcessSwitch
[TR_OSTI_00511]	Definition of API function ArtiProcessCreate
[TR_OSTI_00512]	Definition of API function ArtiProcessDestroy
[TR_OSTI_00513]	Definition of API function ArtiProcessRename
[TR_OSTI_00514]	Definition of API function ArtiProcessInfo
[TR_OSTI_00515]	Definition of API enum CallingContext
[TR_OSTI_00516]	Definition of API class ArtiVersionInfoType

▽

△

Number	Heading
[TR_OSTI_00517]	Definition of API function ArtiVersionInfo
[TR_OSTI_00518]	Definition of API variable ArtiVersionInfoType::apiVersion
[TR_OSTI_00519]	Definition of API variable ArtiVersionInfoType::buildVersion
[TR_OSTI_00520]	Definition of API variable ArtiVersionInfoType::vendorName
[TR_OSTI_00521]	Definition of API variable ArtiVersionInfoType::productName

Table A.2: Added Specification Items in R23-11

A.3.2 Changed Specification Items in R23-11

none

A.3.3 Deleted Specification Items in R23-11

none