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	R23-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Initial release

Disclaimer

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.

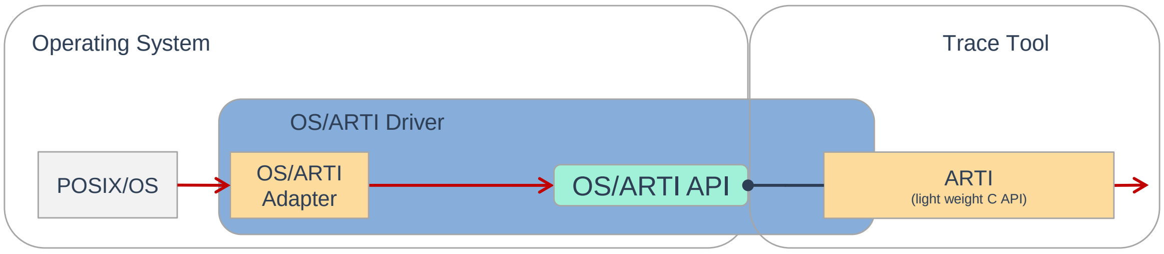

Figure 4.1: Layout of the OS/ARTI Driver

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.

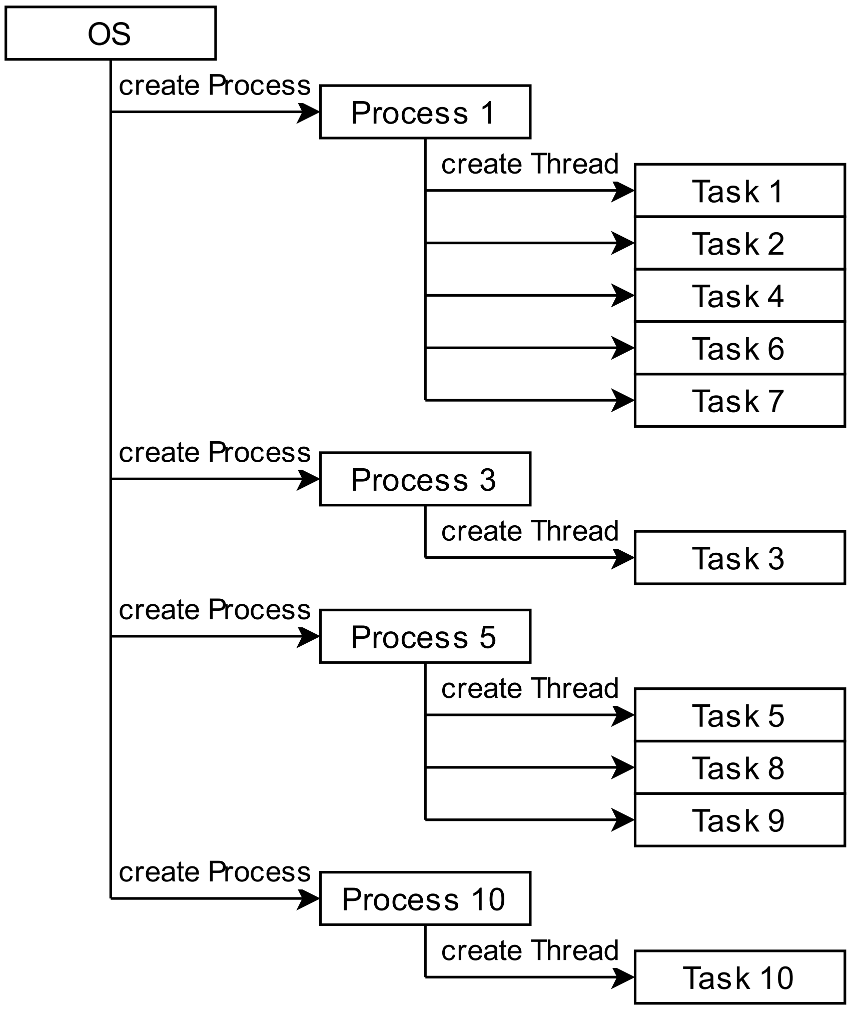

Figure 4.2: Example of Process - Task/Thread Model

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]{DRAFT} **ARTI Version Info** ⌈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.⌋*(RS_OSI_00210)*

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]{DRAFT} **ARTI Initialisation** ⌈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.⌋*(RS_OSI_00210)*

[TR_OSTI_00003]{DRAFT} **ARTI Cleanup** ⌈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.

⌋*(RS_OSI_00210)*

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.

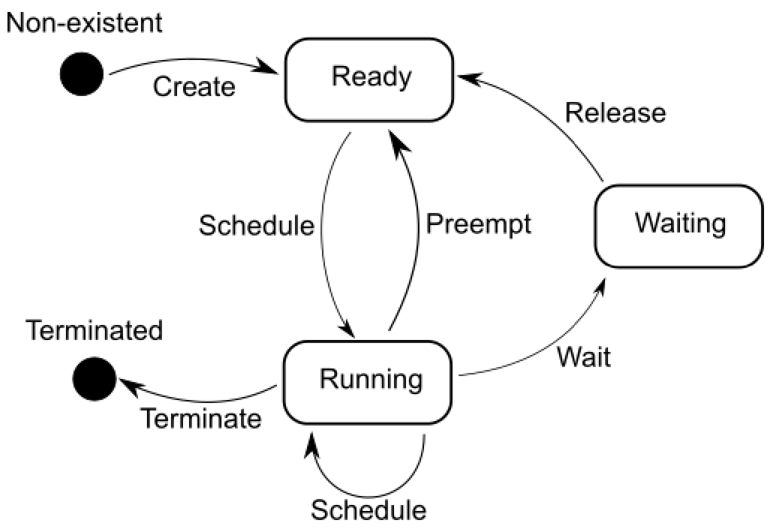

Figure 4.3: Minimal state machine of a task

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]{DRAFT} **ARTI Task Switch Notification** ⌈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.

⌋*(RS_OSI_00210)*

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]{DRAFT} **ARTI Task Wait Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00006]{DRAFT} **ARTI Task Release Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00007]{DRAFT} **ARTI Task Preempt Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00008]{DRAFT} **ARTI Task Exit Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00009]{DRAFT} **ARTI Task Creation Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00010]{DRAFT} **ARTI Task Renaming Notification** ⌈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.

⌋*(RS_OSI_00210)*

Additional interfaces to tasks:

[TR_OSTI_00011]{DRAFT} **ARTI Task Information Notification** ⌈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.⌋*(RS_OSI_00210)*

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]{DRAFT} **ARTI Process Switch Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00013]{DRAFT} **ARTI Process Creation Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00014]{DRAFT} **ARTI Process Destroy Notification** ⌈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.

⌋*(RS_OSI_00210)*

[TR_OSTI_00015]{DRAFT} **ARTI Process Renaming Notification** ⌈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.

⌋*(RS_OSI_00210)*

Additional interfaces to processes:

[TR_OSTI_00016]{DRAFT} **ARTI Process Information Notification** ⌈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.⌋*(RS_OSI_00210)*

5 API Specification

5.1 Type Definitions

5.1.1 ArtiVersionInfoType

[TR_OSTI_00516]{DRAFT} **Definition of API class ArtiVersionInfoType** ⌈

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

⌋*(RS_OSI_00210)*

[TR_OSTI_00518]{DRAFT} **Definition of API variable ArtiVersionInfoType::apiVersion** ⌈

Kind:	variable
Symbol:	apiVersion
Type:	`uint32_t`
Syntax:	`uint32_t apiVersion;`
Header file:	#include "ara/log/osarti.h"
Description:	the version of the API As input parameter it covers the requested version of the API. As output parameter it holds the supported version of the ARTI-driver.

⌋*(RS_OSI_00210)*

[TR_OSTI_00519]{DRAFT} **Definition of API variable ArtiVersionInfoType::buildVersion** ⌈

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

⌋*(RS_OSI_00210)*

[TR_OSTI_00521]{DRAFT} **Definition of API variable ArtiVersionInfoType::productName** ⌈

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

⌋*(RS_OSI_00210)*

[TR_OSTI_00520]{DRAFT} **Definition of API variable ArtiVersionInfoType::vendorName** ⌈

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

⌋*(RS_OSI_00210)*

5.1.2 CallingContext

[TR_OSTI_00515]{DRAFT} **Definition of API enum CallingContext** ⌈

Kind:	enumeration	
Symbol:	CallingContext	
Underlying type:	–	
Syntax:	`enum class CallingContext {...};`	
Values:	kInterruptsDisabled= 0	indicating that the hook gets called in a context where interrupts are disabled
	kInterruptsMayBe Disabled= 1	indicating that the called hook may disable interrupts
	kInterruptsMayNotBe Disabled= 2	indicating the called hook can not disable interrupts
Header file:	#include "ara/log/osarti.h"	
Description:	specifies whether interrupts are disabled or can be disabled	

⌋*(RS_OSI_00210)*

5.2 Callback Notifications

This is a list of functions provided for other modules.

5.2.1 ArtiTaskSwitch

[TR_OSTI_00502]{DRAFT} **Definition of API function ArtiTaskSwitch** ⌈

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

⌋*(RS_OSI_00210)*

5.2.2 ArtiTaskWait

[TR_OSTI_00503]{DRAFT} **Definition of API function ArtiTaskWait** ⌈

Kind:	function	
Symbol:	ArtiTaskWait(CallingContext callingContext, uint32_t coreId, uint32_t taskId)	
Syntax:	`void ArtiTaskWait (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	taskId	id of the task that is entering wait state.
	coreId	coreId of the core that puts the task into wait state.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify 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.	

⌋*(RS_OSI_00210)*

5.2.3 ArtiTaskRelease

[TR_OSTI_00504]{DRAFT} **Definition of API function ArtiTaskRelease** ⌈

Kind:	function	
Symbol:	ArtiTaskRelease(CallingContext callingContext, uint32_t coreId, uint32_t taskId)	
Syntax:	`void ArtiTaskRelease (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	taskId	id of the task that is leaving the wait state.
	coreId	coreId of the core that releases the task.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify 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.	

⌋*(RS_OSI_00210)*

5.2.4 ArtiTaskPreempt

[TR_OSTI_00505]{DRAFT} **Definition of API function ArtiTaskPreempt** ⌈

Kind:	function	
Symbol:	ArtiTaskPreempt(CallingContext callingContext, uint32_t coreId, uint32_t taskId)	
Syntax:	`void ArtiTaskPreempt (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	taskId	id of the task that is leaving the running state.
	coreId	id of the core that preempts the task
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify the tracer that a task is leaving running state and entering ready state. The OS/ARTI Adapter should call this hook when a task is leaving the running state and entering the ready state.	

⌋*(RS_OSI_00210)*

5.2.5 ArtiTaskExit

[TR_OSTI_00506]{DRAFT} **Definition of API function ArtiTaskExit** ⌈

Kind:	function	
Symbol:	ArtiTaskExit(CallingContext callingContext, uint32_t coreId, uint32_t taskId)	
Syntax:	`void ArtiTaskExit (CallingContext callingContext, uint32_t coreId, uint32_t taskId);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	coreId	id of the core that exits the task
	taskId	id of the task that exits
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify the tracer about an exit of a task. The OS/ARTI Adapter shall call this hook when a task is terminated.	

⌋*(RS_OSI_00210)*

5.2.6 ArtiTaskCreate

[TR_OSTI_00507]{DRAFT} **Definition of API function ArtiTaskCreate** ⌈

Kind:	function	
Symbol:	ArtiTaskCreate(CallingContext callingContext, uint32_t coreId, uint32_t processId, uint32_t taskId)	
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	id of the core that creates the task
	processId	id of the process creating the new task
	taskId	id of the task that is beeing created
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify the tracer about the creation of a task. The OS/ARTI Adapter shall call this at the time when the OS creates a new task.	

⌋*(RS_OSI_00210)*

5.2.7 ArtiTaskRename

[TR_OSTI_00508]{DRAFT} **Definition of API function ArtiTaskRename** ⌈

Kind:	function	
Symbol:	ArtiTaskRename(CallingContext callingContext, uint32_t taskId, const char *taskName)	
Syntax:	`void ArtiTaskRename (CallingContext callingContext, uint32_t taskId, const char *taskName);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	taskId	id of the task that is beeing renamed
	taskName	is the name that has to be assigned to the task The size should not exceed 8 bytes.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Provide a name for a task. This name is needed to identify a certain task by the user. The OS/ARTI Adapter should call this function to provide a task name for a taskId.	

⌋*(RS_OSI_00210)*

5.2.8 ArtiTaskInfo

[TR_OSTI_00509]{DRAFT} **Definition of API function ArtiTaskInfo** ⌈

Kind:	function	
Symbol:	ArtiTaskInfo(CallingContext callingContext, uint32_t taskId, uint32_t processId, const char *task Name)	
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	id of the task for which information is provided
	processId	id of the process that owns this task
	taskName	is the task name. The size should not exceed 8 bytes.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Provide information of an existing task. This function provides information about 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.	

⌋*(RS_OSI_00210)*

5.2.9 ArtiProcessSwitch

[TR_OSTI_00510]{DRAFT} **Definition of API function ArtiProcessSwitch** ⌈

Kind:	function	
Symbol:	ArtiProcessSwitch(CallingContext callingContext, uint32_t coreId, uint32_t nextId)	
Syntax:	`void ArtiProcessSwitch (CallingContext callingContext, uint32_t coreId, uint32_t nextId);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	coreId	id of the core that switches the process
	nextId	id of the process that gets the CPU resources
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify the tracer about a switch of a process. In particular, 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.	

⌋*(RS_OSI_00210)*

5.2.10 ArtiProcessCreate

[TR_OSTI_00511]{DRAFT} **Definition of API function ArtiProcessCreate** ⌈

Kind:	function	
Symbol:	ArtiProcessCreate(CallingContext callingContext, uint32_t coreId, uint32_t processId, uint32_t parentId)	
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	id of the core that creates the process
	processId	id of the process that is being created
	parentId	optional id of the parent process, when parentId == processId then parentId is not used.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Notify the tracer about the creation of a process. The OS/ARTI Adapter shall call this at the time when the OS creates a new process.	

⌋*(RS_OSI_00210)*

5.2.11 ArtiProcessDestroy

[TR_OSTI_00512]{DRAFT} **Definition of API function ArtiProcessDestroy** ⌈

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

⌋*(RS_OSI_00210)*

5.2.12 ArtiProcessRename

[TR_OSTI_00513]{DRAFT} **Definition of API function ArtiProcessRename** ⌈

Kind:	function	
Symbol:	ArtiProcessRename(CallingContext callingContext, uint32_t processId, const char *processName)	
Syntax:	`void ArtiProcessRename (CallingContext callingContext, uint32_t processId, const char *processName);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
	processId	id of the process that is beeing renamed
	processName	is the name that has to be assigned to the process The size should not exceed 8 bytes.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Provide 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.	

⌋*(RS_OSI_00210)*

5.2.13 ArtiProcessInfo

[TR_OSTI_00514]{DRAFT} **Definition of API function ArtiProcessInfo** ⌈

Kind:	function	
Symbol:	ArtiProcessInfo(CallingContext callingContext, uint32_t processId, uint32_t parentId, const char *processName)	
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	id of the process for which information is provided
	parentId	id of the parent process
	processName	is the process name.The size should not exceed 8 bytes.
Return value:	None	
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	
Description:	Provide information of an existing process. This function provides information about 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.	

⌋*(RS_OSI_00210)*

5.2.14 ArtiVersionInfo

[TR_OSTI_00517]{DRAFT} **Definition of API function ArtiVersionInfo** ⌈

Kind:	function	
Symbol:	ArtiVersionInfo(CallingContext callingContext, ArtiVersionInfoType const *const versionInfoPtr)	
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, hold the values of the operating system.
Return value:	ArtiVersionInfoType const *const	constant pointer to a constant ArtiVersionInfoType that holds the values of the ARTI-driver.
Thread Safety:	re-entrant	
Header file:	#include "ara/log/osarti.h"	

▽

△

Description:	Assure compatibility of 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 futher using these hooks. When this function is called versionInfoPtr is filled with the OS related values. The versionInfoPtr->api Version 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.

⌋*(RS_OSI_00210)*

5.2.15 ArtiInit

[TR_OSTI_00500]{DRAFT} **Definition of API function ArtiInit** ⌈

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

⌋*(RS_OSI_00210)*

5.2.16 ArtiCleanup

[TR_OSTI_00501]{DRAFT} **Definition of API function ArtiCleanup** ⌈

Kind:	function	
Symbol:	ArtiCleanup(CallingContext callingContext)	
Syntax:	`void ArtiCleanup (CallingContext callingContext);`	
Parameters (in):	callingContext	specifies whether interrupts are disabled or can be disabled.
Return value:	None	
Thread Safety:	re-entrant	

▽

△

Header file:	#include "ara/log/osarti.h"
Description:	Cleanup the OS/ARTI Adapter. The OS/ARTI Adapter shall call this function when it is stopped. It may be used to free local memory or flush pending messages.

⌋*(RS_OSI_00210)*

A Change History

A.1 Change History of this document according to AUTOSAR Release R23-11

A.1.1 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

▽

△

Number	Heading
[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: Added Specification Items in R23-11

A.1.2 Changed Specification Items in R23-11

none

A.1.3 Deleted Specification Items in R23-11

none