Document Title	Specification of Operating System Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	719

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Extended ARTI tracing interface and related Log messages. • Minor changes, document clean up
2023-11-23	R23-11	AUTOSAR Release Management	• Added ARTI tracing interface and related Log messages.
2022-11-24	R22-11	AUTOSAR Release Management	• No content changes
2021-11-25	R21-11	AUTOSAR Release Management	• Uptrace update
2020-11-30	R20-11	AUTOSAR Release Management	• Uptrace update • Clarified Execution Management description • Removed undefined mention of Unrecoverable State

△

2019-11-28	R19-11	AUTOSAR Release Management	• Added description of startup and shutdown of OSI. • Clarified that `Operating System` must allow calling getenv() from C++ constructors. • Document template upgrade. • Changed Document Status from Final to published.
2019-03-29	19-03	AUTOSAR Release Management	• Clarified that PSE51 following POSIX-1003.1-2003 is the currently-targeted version. • Minor changes in tracing, clean up
2018-10-31	18-10	AUTOSAR Release Management	• Add Resource Control • Added Shared object support
2018-03-29	18-03	AUTOSAR Release Management	• Minor changes
2017-10-27	17-10	AUTOSAR Release Management	• Minor changes, document clean up
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

1 Introduction and Functional Overview

This document is the software specification of the Operating System Interface within the AUTOSAR Adaptive Platform.

AUTOSAR Adaptive Platform does not specify a new Operating System for highly performant microcontrollers. Rather, it defines an execution context and programming interface for use by Adaptive Applications.

Note that this Operating System Interface specification contains application interfaces that are part of ARA, the standard application interface of Adaptive Application. The OS itself may very well provide other interfaces, such as creating processes, that are required by Execution Management to start an Adaptive Application. However, the interfaces providing such functionality, among others, are not available as part of ARA and it is defined to be platform implementation dependent.

The Operating System Interface provides both C and C++ interfaces. In case of a C program, the application's main source code business logic include C function calls defined in the POSIX standard, namely PSE51 defined in IEEE1003.13 [1]. During compilation, the compiler determines which C library from the platform's Operating System provides these C functions and the application's Executable must be linked against at runtime. In case of a C++ program, application software component's source code includes function calls defined in the C++ Standard and its Standard C++ Library.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to this document.

Term	Description
Initial Process	A process with management rights, e.g. to determine exit status, for all processes within the AUTOSAR Adaptive Platform.
Operating System	Software responsible for managing Processes on a Machine and for providing an interface to hardware resources.

Table 2.1: Technical Terms

The following technical terms used in this document are defined in the corresponding document mentioned in the table below.

Term	Description
Operating System Interface	see [2] Requirements on Operating System Interface
Execution Management	see [3] Requirements on Execution Management
Adaptive Application	see [4] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [4] AUTOSAR Glossary
Executable	see [4] AUTOSAR Glossary
Foundation	see [4] AUTOSAR Glossary
Functional Cluster	see [4] AUTOSAR Glossary
Machine	see [4] AUTOSAR Glossary
Process	see [4] AUTOSAR Glossary
Task	In case of POSIX a Task is called thread or pthread. see [4] AUTOSAR Glossary
ARTI	see [4] AUTOSAR Glossary

Table 2.2: Reference to Technical Terms

3 Related Documentation

3.1 Input Documents & Related Standards and Norms

[1] IEEE Standard for Information Technology- Standardized Application Environment Profile (AEP)-POSIX Realtime and Embedded Application Support
https://standards.ieee.org/findstds/standard/1003.13-2003.html

[2] Requirements on Operating System Interface
AUTOSAR_AP_RS_OperatingSystemInterface

[3] Requirements on Execution Management
AUTOSAR_AP_RS_ExecutionManagement

[4] Glossary
AUTOSAR_FO_TR_Glossary

[5] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

3.2 Further applicable specification

None.

4 Constraints and assumptions

4.1 Known Limitations

This chapter lists known limitations of this software specification. The intent is to not only provide a specification of the current state of the Operating System Interface but also an indication how the AUTOSAR Adaptive Platform will evolve in future releases.

These Requirements are not yet part of this specification.

- [RS_OSI_00204]
- [RS_OSI_00208]
- [RS_OSI_00209]

5 Dependencies to Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 "Provided Interfaces" lists the interfaces provided by Operating System Interface to other Functional Clusters. Section 5.2 "Required Interfaces" lists the interfaces required by Operating System Interface.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [5].

5.1 Provided Interfaces

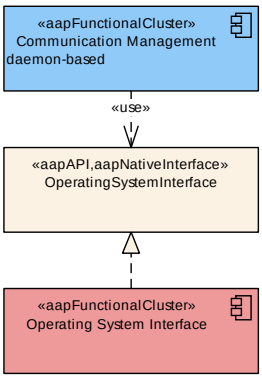

Figure 5.1: Interfaces provided by Operating System Interface to other Functional Clusters

Figure 5.1 shows the interfaces provided by Operating System Interface to other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
OperatingSystem Interface	Communication Management	Communication Management should use this interface to create and control Threads used by the implementation.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

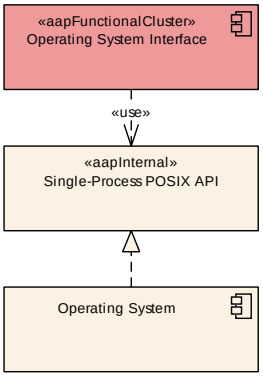

Figure 5.2: Interfaces required by Operating System Interface

Figure 5.2 shows the interfaces required by `Operating System Interface`. Table 5.2 provides a complete list of required interfaces from other `Functional Clusters` within the `AUTOSAR Adaptive Platform`.

Functional Cluster	*Interface*	*Purpose*
No required interfaces		

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracability

The following table references the features specified in [2] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_AP_00111]	Source Code Portability Support	[SWS_OSI_01001] [SWS_OSI_01002]
[RS_AP_00114]	C++ interface shall be compatible with C++14	[SWS_OSI_01002]
[RS_OSI_00100]	POSIX PSE51 Compliance	[SWS_OSI_01001] [SWS_OSI_01002] [SWS_OSI_01003] [SWS_OSI_01006]
[RS_OSI_00103]	The Operating System Interface shall support C++.	[SWS_OSI_01002] [SWS_OSI_01015]
[RS_OSI_00104]	The Operating System Interface shall support the reaction on Process-external stimuli from devices.	[SWS_OSI_01001]
[RS_OSI_00105]	The Operating System Interface shall support the start of Execution Management.	[SWS_OSI_01040]
[RS_OSI_00201]	The Operating System shall provide mechanisms for system memory budgeting.	[SWS_OSI_02000] [SWS_OSI_02001]
[RS_OSI_00202]	The Operating System shall provide mechanisms for CPU time budgeting.	[SWS_OSI_02000] [SWS_OSI_02002]
[RS_OSI_00203]	The Operating System should provide mechanisms for binding Processes to CPU cores.	[SWS_OSI_01006] [SWS_OSI_01012]
[RS_OSI_00206]	The Operating System shall provide multi-Process support for isolation of applications.	[SWS_OSI_01006] [SWS_OSI_01008] [SWS_OSI_01009] [SWS_OSI_01010] [SWS_OSI_01013] [SWS_OSI_01014]
[RS_OSI_00207]	The Operating System shall provide the capability to share code and data in an implicit manner.	[SWS_OSI_01013]
[RS_OSI_00211]	The Operating System shall provide a mechanism to export low-level scheduling and trace information to applications.	[SWS_OSI_02003] [SWS_OSI_02004] [SWS_OSI_02005] [SWS_OSI_02006] [SWS_OSI_02007] [SWS_OSI_02008] [SWS_OSI_02009] [SWS_OSI_02010] [SWS_OSI_02011] [SWS_OSI_02012] [SWS_OSI_02013] [SWS_OSI_02014] [SWS_OSI_02015] [SWS_OSI_10100] [SWS_OSI_10102] [SWS_OSI_10103] [SWS_OSI_10104] [SWS_OSI_10105] [SWS_OSI_10106] [SWS_OSI_10107] [SWS_OSI_10108] [SWS_OSI_10110] [SWS_OSI_10111] [SWS_OSI_10112] [SWS_OSI_10113]

Table 6.1: Requirements Tracing

6.1 Non-applicable requirements

[SWS_OSI_NA]

Upstream requirements: RS_AP_00115, RS_AP_00116, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00122, RS_AP_00124, RS_AP_00125, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00130, RS_AP_00134, RS_AP_00135, RS_AP_00136, RS_AP_00137, RS_AP_00138, RS_AP_00139, RS_AP_00140, RS_AP_00141, RS_AP_00142, RS_AP_00143, RS_AP_00144, RS_AP_00145, RS_AP_00146, RS_AP_00147, RS_AP_00148, RS_AP_00149, RS_AP_00150, RS_AP_00151, RS_AP_00153, RS_AP_00154, RS_AP_00155, RS_AP_00156, RS_OSI_00210, RS_OSI_NA

⌈These requirements are not applicable as they are not within the scope of this release.⌋

7 Functional specification

7.1 Functional Cluster Lifecycle

7.1.1 Operating System Overview

The real-time Operating System in an embedded automotive ECU offers the foundation for dynamic behavior of the software applications. It manages the scheduling of processes and events, the data exchange and synchronization between different processes and provides features for monitoring and error handling. This chapter describes requirements addressed to the Operating System. Applications, in particular Adaptive Applications may not have the system rights to fully use or configure these aspects directly.

7.1.2 Process Handling

[SWS_OSI_01040] Start Execution Management as Initial Process.

Upstream requirements: RS_OSI_00105

⌈The Operating System shall allow starting the Execution Management as the Initial Process of the AUTOSAR Adaptive Platform.⌋

[SWS_OSI_01006] Multi-Threading Support

Upstream requirements: RS_OSI_00100, RS_OSI_00203, RS_OSI_00206

⌈The Operating System shall allow running multiple execution contexts (threads) such that the process can execute multiple code flows.⌋

On multi-core platforms, multiple threads permitted by [SWS_OSI_01006] may execute concurrently on different cores. All the threads belong to some process, so it is possible that multiple threads in the same process may execute on multiple cores concurrently. Additionally, Execution Management requires the ability to bind a specific Process to a core as part of resource management [SWS_EM_02104].

[SWS_OSI_01012] Specification of Core Affinity

Upstream requirements: RS_OSI_00203

⌈The Operating System shall provide mechanisms for binding processes to CPU cores.⌋

In general, a process provides at least the following:

- A `main()` function as the entry point of the first execution thread of the process.

- A local memory context (address space), providing local, non-shared memory, that includes at least the code, data and heap of the process.
- Some level of memory protection, such that incorrect or invalid memory accesses are detected by the underlying `Operating System`.
- `Operating System` descriptors permitting access to OS managed resources.

[SWS_OSI_01008] Multi-Process Support

Upstream requirements: RS_OSI_00206

⌈The `Operating System` shall support multiple processes.⌋

[SWS_OSI_01009] Multi-Process Isolation

Upstream requirements: RS_OSI_00206

⌈The `Operating System` shall isolate each process from one another such that an incorrect or invalid memory access is detected by the `Operating System`.⌋

[SWS_OSI_01014] Multi-Process Creation Capability Restriction

Upstream requirements: RS_OSI_00206

⌈The `Operating System` shall allow configuring a process to be forbidden from creating other processes.⌋

[SWS_OSI_01010] Virtual Memory

Upstream requirements: RS_OSI_00206

⌈`Operating System` shall execute each process in a dedicated address space.⌋

Each process has its own logical address space where the code and data are located. The address may or may not correspond to their underlying physical address space as the process's address space is virtualized. In particular, multiple instances of the same `Executable` running in different logical address spaces may share the physical address for its code and read-only data, as they are read-only, to save some physical memory. The rewritable data, on the other hand, need to be separate, so they are mapped to different physical addresses.

Shared objects (also sometimes called DLLs) usually consist of code and data usable from multiple processes simultaneously. When multiple processes use a shared object, code and read-only data of the shared object is usually mapped in each process but present only once in system memory, while shared data may be duplicated immediately or when needed (Copy-on-Write, or CoW).

Shared objects can be used in mainly two ways:

- *Implicit loading*: at build time, an `Executable` may be linked against a shared object ; later on, at load time, the `Operating System` and its loading framework

enable the mapping and use of the shared object code and data in the process of the Adaptive Application. This is mainly used for space saving and ease of deploying fixes in shared code, but sometimes also for licensing reasons. The process itself does not require any specific capability or knowledge of this shared library existence to make use of it.

- *Explicit loading*: at run time, the Process requests the Operating System and its loading framework to open and load a shared object on the target, and to let it resolve symbol names and load its code and data. This is usually done for plugin mechanisms where all plugins expose the same shared symbols. The Executable itself has no knowledge of the plugins at link time, and typically uses the `dlopen()`/`dlsym()`/`dlclose()` to enable using the plugin-style loaded shared object.

[SWS_OSI_01013] Implicit shared object support

Upstream requirements: RS_OSI_00207, RS_OSI_00206

⌈The Operating System shall allow the use of Implicit loading of shared objects for Executables.⌋

Note that for safety, security or other reasons, an Executable may be built fully statically-linked, and therefore not use the capability to use shared objects.

7.1.3 Scheduling Policies

The Operating System Scheduler is designed to keep all system resources busy allowing multiple software control flows to share the CPU cores in an effective manner. The main goals of the scheduling mechanisms may be one or more from the following:

- Maximizing throughput in terms of amount of work done per time unit.
- Maximizing responsiveness by minimizing the time between job activation and actual begin of data processing.
- Maximizing fairness in terms of ensuring appropriate CPU time according with priority and workload of each job.
- Assuring a timelined and ordered activation of jobs according to some policy-dependent job execution eligibility (e.g. priority, deadline, tardiness, etc).

In real life these goals are often in conflict, implementing the scheduling mechanisms is therefore always a compromise.

[SWS_OSI_01003] Default Scheduling Policies

Upstream requirements: RS_OSI_00100

⌈The AUTOSAR Adaptive Platform Operating System shall support the following scheduling policies defined in the IEEE1003.1 POSIX standard: SCHED_OTHER, SCHED_FIFO, SCHED_RR.⌋

In order to overcome the above mentioned conflicts and to achieve portability between different platforms, the AUTOSAR Adaptive Platform Operating System provides the following scheduling policies categorized in two groups:

- Fair Scheduling Policies
  - SCHED_OTHER
- Real-time Scheduling Policies
  - SCHED_FIFO
  - SCHED_RR

Since the above mentioned default scheduling policies may not guarantee proper execution for all real-time scenarios, the Adaptive Application vendor may provide additional scheduling policies to fulfill any execution requirement. For example, additional non-POSIX scheduling policies like SCHED_DEADLINE (Earliest Deadline First algorithm) could be introduced to satisfy hard real-time requirements.

7.1.4 Time Triggered Execution

POSIX PSE51 provides a means to do time-based periodic processing, using the timer API (e.g. `timer_settime()`) along with POSIX signals. However, signals are sometimes discouraged for safety-critical applications, because they disrupt the execution flow.

Using C++, `std::future::wait_until()` can be used to realize periodic processing. The TimeSync specification may also be used along with std::future to provide event generation. However, both of these APIs only allow single-shot, relative alarms, and efficient, low-overhead requires recurring and/or absolute alarms.

Therefore, these APIs may be extended in the future.

7.1.5 Device Support

The Operating System Interface shall support device access as defined in POSIX PSE51.

7.1.6 Resource control

While correct behavior is expected from each application, intentional or unintentional misbehavior must be contained for system stability. Simultaneously, some level of dynamic behavior must be allowed. From a feature perspective, applications can be assembled in groups such that they can follow a similar usage pattern, sharing memory, CPU time, and in general resources.

[SWS_OSI_02000] ResourceGroup minimum requirement

Upstream requirements: RS_OSI_00201, RS_OSI_00202

⌈The `Operating System` shall support the configuration of at least 8 groups of processes in the system.⌋

Depending on the `Operating System`, the number of usable `ResourceGroup`s may vary. Furthermore, when OS-level-virtualized containers are used, some `Operating Systems` may additionally constrain the number of usable `ResourceGroup`s, with an extreme of just 1 available `ResourceGroup`.

[SWS_OSI_02001] Memory ResourceGroups

Upstream requirements: RS_OSI_00201

⌈The `Operating System` shall support a mechanism to define groups of processes that may dynamically allocate memory from a configuration-defined limit.⌋

The memory taken in consideration for the limit covers:

- Code and read-only Data from the `Executable`
- Modifiable Data from the `Executable`
- Memory used for thread stack for each thread of the process
- Heap
- System memory that is used by the `Operating System` for holding the kernel resources allocated to the process (e.g. thread control block, semaphore, page table entries for MMU mapping, etc)
- Shared memory between processes of the same group
- Implicitly loaded shared objects between processes of the same group

Because memory accounting may differ between `Operating Systems`, some elements can be considered inside or outside the memory usage limit of the process group, in an implementation-specific manner:

- Shared memory between processes of different groups
- Memory-mapped files

- Implicitly loaded shared objects between processes of different groups

[SWS_OSI_02002] CPU ResourceGroups

Upstream requirements: RS_OSI_00202

⌈The Operating System shall support a mechanism to define groups of processes that may use a maximum configured amount of CPU time over a defined period of time.⌋

Because scheduling is done in very different ways depending on the Operating System, the specific algorithm for scheduling as well as limiting the CPU usage is not described here.

Example valid group scheduling schemes include (but not limited to):

- Fixed-periodic enablement of processes over a fixed range of time, in a manner similar to what the ARINC 653 standard defines.
- Processes use time from a quota of time allocated to the group. If no time remains, no thread from the processes in the expired group can be scheduled. Each period, the quota is replenished to allow more time to be used and corresponding threads to be scheduled again.
- Processes accumulate time usage. Each period or each context switch, time usage accumulated over a certain count of past periods is calculated. Processes of each group that used time over a threshold are disabled, and processes of each group that used time under a threshold are enabled.

Most notably, on some Operating Systems, idle time, which by definition is not requested to be used by any process group, may be distributed to any process, including those belonging to a group that is considered to be using time over the defined limit. This is a worthy optimization, but is currently not considered in the specification as a requirement.

7.2 Startup

The startup steps of an Operating System have to be executed in an implementation-specific way. These steps include starting any Operating System-related middleware, including device-drivers and services handling low-level middleware, as well as starting Execution Management.

As an important remark, it is expected that Execution Management will be started early on during the system boot, ideally as the first process, in order to allow booting all the required Processes. However, depending on the Operating System, other system services and supporting middleware may be started before or in parallel. An example may be a filesystem service, if the Operating System has one that is not part of its kernel.

7.3 Shutdown

Similarly, shutdown steps for an Operating System are implementation-specific. They may include flushing some middleware buffers, shutting down some peripherals, and optionally turn off the entire system, depending on the system configuration.

7.4 ARTI Tracing Interface

The ARTI Tracing Interface is used to understand, verify and visualize the timing behavior of the OS. It is used to collect information about tasks and processes of the OS.

The ARTI interface follows the two-level approach of AUTOSAR, where a “task” is a schedulable unit (in POSIX and C++ called “thread”), and a “process” is a mandatory environment holding several tasks. A system may look like this:

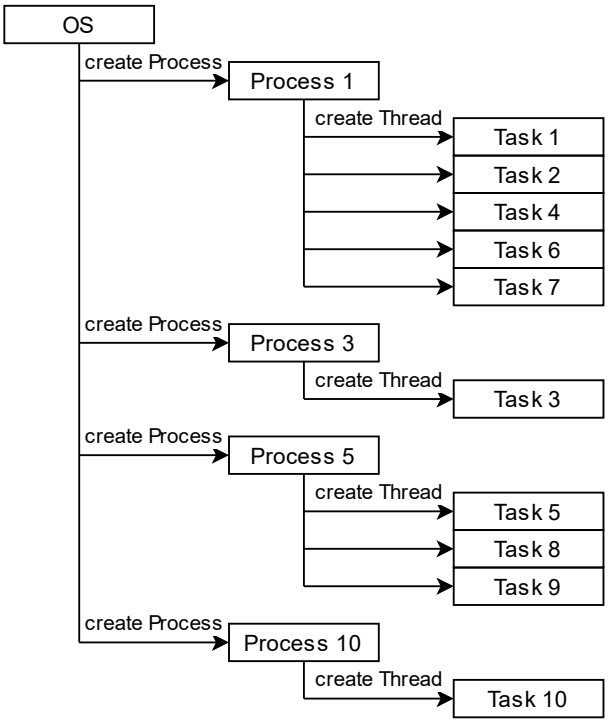

Figure 7.1: Process - Task/Thread Model

The OS provides the information of processes and tasks in different ways. It can implement trace buffers that contain kernel internal information, it could provide propritary hooks within the kernel with internal information or it could provide modeled messages out of the box.

These different variants of the OS specific information need to be sent using modeled messages to the ara::log API. This can be implemented in different ways.

For example, for Operating Systems that are using trace buffers with internal information, there could be a separate application. Here such application or daemon is called “OS/ara::log Adapter”. The Figure 7.2 “Example layout of the OS/ara::log Adapter” shows how it integrates in the AUTOSAR framework. The OS/ara::log Adapter reads OS specific trace buffers, translates the information to modeled messages and sends them to the ara::log API.

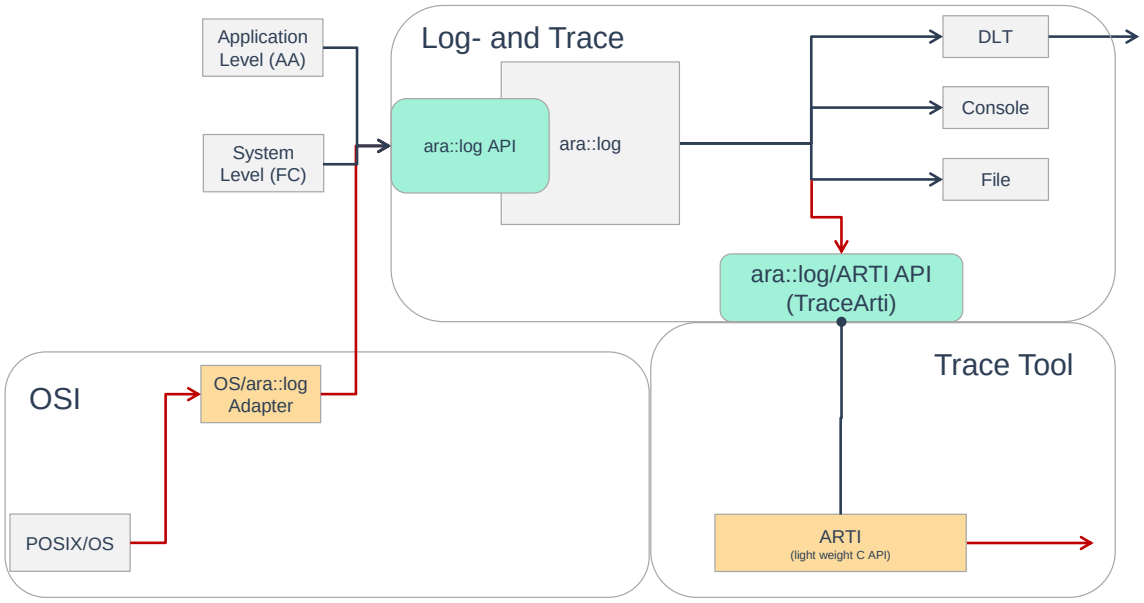

Figure 7.2: Example layout of the OS/ara::log Adapter

Sending the modeled messages can also be delegated to a different Functional Cluster. This can be helpfull when sending these messages would be the only active part of the Operating System Interface.

The initial state of existing processes and tasks when the tracing is started is logged using the OsProcessInfo and OsTaskInfo message. This initial state assures that task ids and process ids can be correctly interpreted and can be assigned to Executables.

[SWS_OSI_10100] Log OS tracing started

Upstream requirements: RS_OSI_00211

⌈Whenever the Operating System Interface starts the tracing of processes and tasks, it shall

- log a modeled message of type OsProcessInfo for each process currently available.
- log a modeled message of type OsTaskInfo for each task currently available.

⌋

While the concepts of process and task are very common in most memory-partitioning OSes, there are still very significant variations on how these concepts are concretely implemented. As a consequence, the order of declaration of creation, destruction and modification of each resource logged in the output is voluntarily weakly defined. The only requirement is that the log describes a coherent view of the system. Tooling must be tolerant to the variety of ordering while decoding the information.

7.4.1 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 7.3.

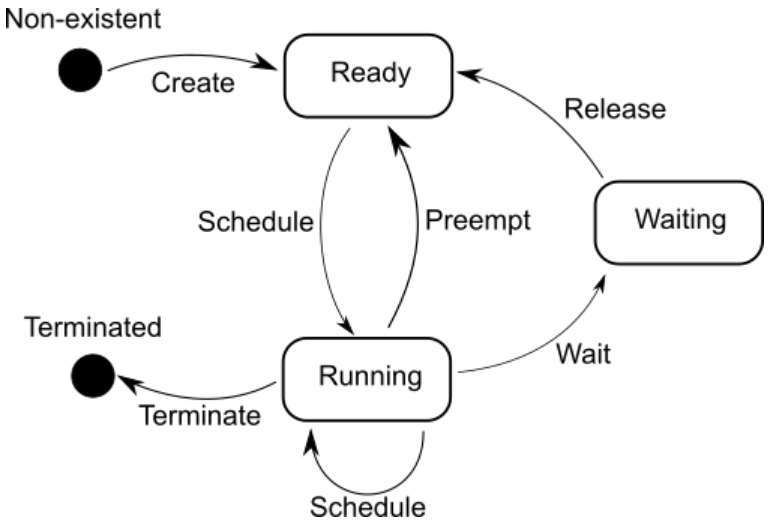

Figure 7.3: minimal state machine of a task

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 events for tracing switches between Ready and Running shall be mandatory, and `ARTI` trace events for switching to Waiting state are optional.

The trace points that are related to tasks are:

[SWS_OSI_10102] Log Task Schedule Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task is scheduled and is entering the running state, the `Operating System Interface` shall log a modeled message of type `OsTaskSchedule`.⌋

[SWS_OSI_10103] Log Task Wait Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task enters the wait state, the `Operating System Interface` shall log a modeled message of type `OsTaskWait`.⌋

[SWS_OSI_10104] Log Task Release Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task is released, the `Operating System Interface` shall log a modeled message of type `OsTaskRelease`.⌋

[SWS_OSI_10105] Log Task Preempt Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task is preempted, the `Operating System Interface` shall log a modeled message of type `OsTaskPreempt`.⌋

[SWS_OSI_10106] Log Task Exit Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task exits, the `Operating System Interface` shall log a modeled message of type `OsTaskTerminate`.⌋

[SWS_OSI_10107] Log Task Creation Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task is created, the `Operating System Interface` shall log a modeled message of type `OsTaskCreate`.⌋

[SWS_OSI_10108] Log Task Renaming Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS task is renamed, the `Operating System Interface` shall log a modeled message of type `OsTaskRename`.⌋

The timestamp parameter shall cover the time when the event occured. This assures the most accurate time that is possible. The format of the timestamp is the natural format of the OS.

7.4.2 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.”

The trace points that are related to processes are:

[SWS_OSI_10110] Log Process Switch Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS process is switched, the Operating System Interface shall log a modeled message of type OsProcessSwitch.⌋

[SWS_OSI_10111] Log Process Creation Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS process is created, the Operating System Interface shall log a modeled message of type OsProcessCreate.⌋

[SWS_OSI_10112] Log Process Ending Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS process ends, the Operating System Interface shall log a modeled message of type OsProcessDestroy.⌋

[SWS_OSI_10113] Log Process Renaming Notification

Upstream requirements: RS_OSI_00211

⌈If tracing is desired then whenever an OS process is renamed, the Operating System Interface shall log a modeled message of type OsProcessRename.⌋

The timestamp parameter shall cover the time when the event occured. This assures the most accurate time that is possible. The format of the timestamp is the natural format of the OS.

8 API Specification

The AUTOSAR Adaptive Platform does not specify a new Operating System for highly performant microcontrollers. Rather, it defines an execution context and programming interface for use by Adaptive Applications.

8.1 C++ language binding Operating System

8.1.1 Application Interface C (POSIX PSE51)

[SWS_OSI_01001] POSIX PSE51 Interface

Upstream requirements: RS_OSI_00100, RS_OSI_00104, RS_AP_00111

⌈The Operating System Interface shall provide OS functionality with POSIX PSE51 interface, according to the 1003.13-2003 specification.⌋

Note that PSE51 requires C99 as specified in the standard.

There are several Operating Systems on the market, e.g. Linux, that provide POSIX compliant interfaces. However Adaptive Applications are required to use a more restricted API to the Operating Systems as compared to the platform services and Foundation. In particular, the starting assumption is that an Adaptive Application may use PSE51 as OS interface whereas platform-specific applications may use full POSIX.

The implementation of platform Foundation and platform services functionality may use non-PSE51 APIs, even OS specific ones. The use of specific APIs will be left open to the implementer of the AUTOSAR Adaptive Platform and is not standardized.

In case of a C program, the applications main source code business logic includes C function calls defined in the POSIX standard. During compilation, the compiler determines which C library from the platforms Operating System provides these C functions and the applications Executable must be linked against at runtime. This Operating System provided C library can implement the POSIX-compliant C function in two ways:

- The provided C library implements the behavior as part of the library. Then, the execution of this C function causes no further invocation of the Operating System with a system call.
- The provided C library implements the behavior through a suitable system call of the Operating System kernel. In many cases, the function name and behavior of the Operating System kernel system call match very closely to the Operating System provided C library and to the POSIX-specified function definitions. For example, in the case of typical Linux distributions, these functions

are provided by glibc library, and by default, the gcc compiler links the glibc library dynamically.

[SWS_OSI_01015] Availability of environment variables

Upstream requirements: RS_OSI_00103

⌈The POSIX function getenv() should return valid values as soon as non-Operating System-provided functionality can be called, and specifically from within C++ static initializer code.⌋

8.1.2 Application Interface C++11

[SWS_OSI_01002] Use of C++ Language

Upstream requirements: RS_OSI_00100, RS_OSI_00103, RS_AP_00111, RS_AP_00114

⌈The Operating System Interface shall provide OS functionality with C++11 Standard Library for Adaptive Applications written in C++.⌋

In case of a C++ program, application software components source code can include function calls defined in the C++11 Standard and its Standard C++ Library. The C++ Standards defines C++ Standard Library (http://en.cppreference.com/w/cpp), and it includes Thread support library, Input/output library and others that provide most of PSE51 functionalites through these C++ interfaces. Some PSE51 functions, such as setting thread scheduling policies, are not available yet through these C++ Standard Library and Adaptive Applications (implemented in C++) need to use PSE51 C interface in conjunction with these C++ libraries.

In case of Linux and the gcc C++ compiler (g++), the compiler links the libstdc++ library, which provides the defined Standard C++ library functions. The libstdc++ library itself depends on the glibc library, i.e., the libstdc++ implementation includes function calls to the glibc library.

8.2 API Reference

8.3 Log and Trace Messages

[SWS_OSI_02003] LogMessage OsProcessCreate

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsProcessCreate		
Description	Notify the tracer about the creation of a process.		
MessageId	0x8000e000		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
ProcessId	Id of the process that is being created.	uint32 [1]	NoUnit
ParentId	Id of the parent process that is the parent of the process created. If there is no parent process then it is identical to the process beeing created.	uint32 [1]	NoUnit

⌋

[SWS_OSI_02004] LogMessage OsProcessDestroy

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsProcessDestroy		
Description	Notify the tracer about the destruction of a process.		
MessageId	0x8000e001		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
ProcessId	Id of the process that is being destroyed.	uint32 [1]	NoUnit

⌋

[SWS_OSI_02005] LogMessage OsProcessInfo

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsProcessInfo		
Description	Provide information of an existing process		
MessageId	0x8000e002		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
ProcessId	Id of the process	uint32 [1]	NoUnit
ParentId	Id of the parent process	uint32 [1]	NoUnit
ProcessName	Name of the process that is specified by ProcessId.	uint8 [32, encoding UTF-8]	NoUnit

⌋

[SWS_OSI_02006] LogMessage OsProcessRename

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsProcessRename		
Description	Provide a name for a process.		
MessageId	0x8000e003		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
ProcessId	Id of the process	uint32 [1]	NoUnit
ProcessName	New name of the process.	uint8 [32, encoding UTF-8]	NoUnit

⌋

[SWS_OSI_02007] LogMessage OsProcessSwitch

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsProcessSwitch
Description	Notify the tracer about the switch of a process.
MessageId	0x8000e004

▽

△

MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
NextProcessId	Id of the process	uint32 [1]	NoUnit

⌋

[SWS_OSI_02008] LogMessage OsTaskCreate

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskCreate		
Description	Notify the tracer about the creation of a task.		
MessageId	0x8000e005		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
ProcessId	Id of process the task is in	uint32 [1]	NoUnit
TaskId	Id of the task that is created	uint32 [1]	NoUnit

⌋

[SWS_OSI_02009] LogMessage OsTaskTerminate

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskTerminate		
Description	Notify the tracer about the exit of a task.		
MessageId	0x8000e006		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
TaskId	Id of the task that exits	uint32 [1]	NoUnit

⌋

[SWS_OSI_02010] LogMessage OsTaskInfo

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskInfo		
Description	Provide information of an existing task		
MessageId	0x8000e007		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TaskId	Id of the task	uint32 [1]	NoUnit
ProcessId	Id of the parent process	uint32 [1]	NoUnit
TaskName	New name of the task	uint8 [32, encoding UTF-8]	NoUnit

⌋

[SWS_OSI_02011] LogMessage OsTaskPreempt

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskPreempt		
Description	Notify the tracer that a task is leaving running state		
MessageId	0x8000e008		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
TaskId	Id of the task that is leaving the running state and enters the ready state.	uint32 [1]	NoUnit

⌋

[SWS_OSI_02012] LogMessage OsTaskRelease

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskRelease
Description	Notify the tracer that a task is leaving the wait state
MessageId	0x8000e009

▽

△

MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
TaskId	Id of the task that is leaving the wait state.	uint32 [1]	NoUnit

⌋

[SWS_OSI_02013] LogMessage OsTaskRename

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskRename		
Description	Provide a name for a task		
MessageId	0x8000e00a		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
TaskId	Id of the task	uint32 [1]	NoUnit
TaskName	New name of the task	uint8 [32, encoding UTF-8]	NoUnit

⌋

[SWS_OSI_02014] LogMessage OsTaskSchedule

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskSchedule		
Description	Notify the tracer about that the task is scheduled and is entering the running state.		
MessageId	0x8000e00b		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64 [1]	NoUnit
CoreId	Id of the core	uint32 [1]	NoUnit
NextTaskId	Id of the task that starts running.	uint32 [1]	NoUnit

⌋

[SWS_OSI_02015] LogMessage OsTaskWait

Status: DRAFT
Upstream requirements: RS_OSI_00211

⌈

Dlt-Message	OsTaskWait		
Description	Notify the tracer that a task is entering the wait state.		
MessageId	0x8000e00c		
MessageType Info	DLT_TRACE_STATE		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TimeStamp	Time when the event occurred.	uint64	NoUnit
CoreId	Id of the core	uint32	NoUnit
TaskId	Id of the task that is entering the wait state.	uint32	NoUnit

⌋

A Appendix

A.1 Mentioned Manifest Elements

This section contains the Manifest Elements mentioned in this documentation. It also contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**ResourceGroup**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::AdaptiveModuleImplementation			
Note	This meta-class represents a resource group that limits the resource usage of a collection of processes.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	OsModuleInstantiation.resourceGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
cpuUsage	PositiveInteger	0..1	attr	CPU resource limit in percentage of the total CPU capacity on the machine.
memUsage	PositiveInteger	0..1	attr	Memory limit in bytes.

Table A.1: ResourceGroup

A.2 Interfaces to other Functional Clusters (informative)

Other `Functional Clusters` use the standard C/C++ as well as POSIX APIs to provide their public APIs. They may additionally use `Operating System`-specific APIs to implement their functionality.

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R24-11

B.1.1 Added Specification Items in R24-11

none

B.1.2 Changed Specification Items in R24-11

Number	Heading
[SWS_OSI_01001]	POSIX PSE51 Interface
[SWS_OSI_01002]	Use of C++ Language
[SWS_OSI_02003]	LogMessage OsProcessCreate
[SWS_OSI_02004]	LogMessage OsProcessDestroy
[SWS_OSI_02005]	LogMessage OsProcessInfo
[SWS_OSI_02006]	LogMessage OsProcessRename
[SWS_OSI_02007]	LogMessage OsProcessSwitch
[SWS_OSI_02008]	LogMessage OsTaskCreate
[SWS_OSI_02009]	LogMessage OsTaskTerminate
[SWS_OSI_02010]	LogMessage OsTaskInfo
[SWS_OSI_02011]	LogMessage OsTaskPreempt
[SWS_OSI_02012]	LogMessage OsTaskRelease
[SWS_OSI_02013]	LogMessage OsTaskRename
[SWS_OSI_02014]	LogMessage OsTaskSchedule
[SWS_OSI_02015]	LogMessage OsTaskWait
[SWS_OSI_10100]	Log OS tracing started
[SWS_OSI_10102]	Log Task Schedule Notification
[SWS_OSI_10103]	Log Task Wait Notification
[SWS_OSI_10104]	Log Task Release Notification
[SWS_OSI_10105]	Log Task Preempt Notification
[SWS_OSI_10106]	Log Task Exit Notification
[SWS_OSI_10107]	Log Task Creation Notification

△

Number	Heading
[SWS_OSI_10108]	Log Task Renaming Notification
[SWS_OSI_10110]	Log Process Switch Notification
[SWS_OSI_10111]	Log Process Creation Notification
[SWS_OSI_10112]	Log Process Ending Notification
[SWS_OSI_10113]	Log Process Renaming Notification

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

B.1.3 Deleted Specification Items in R24-11

none

B.2 Traceable item history of this document according to AUTOSAR Release R23-11

B.2.1 Added Specification Items in R23-11

Number	Heading
[SWS_OSI_02003]	LogMessage OsProcessCreate
[SWS_OSI_02004]	LogMessage OsProcessDestroy
[SWS_OSI_02005]	LogMessage OsProcessInfo
[SWS_OSI_02006]	LogMessage OsProcessRename
[SWS_OSI_02007]	LogMessage OsProcessSwitch
[SWS_OSI_02008]	LogMessage OsTaskCreate
[SWS_OSI_02009]	LogMessage OsTaskTerminate
[SWS_OSI_02010]	LogMessage OsTaskInfo
[SWS_OSI_02011]	LogMessage OsTaskPreempt
[SWS_OSI_02012]	LogMessage OsTaskRelease
[SWS_OSI_02013]	LogMessage OsTaskRename
[SWS_OSI_02014]	LogMessage OsTaskSchedule
[SWS_OSI_02015]	LogMessage OsTaskWait
[SWS_OSI_10100]	Log OS tracing started
[SWS_OSI_10102]	Log Task Schedule Notification
[SWS_OSI_10103]	Log Task Wait Notification
[SWS_OSI_10104]	Log Task Release Notification
[SWS_OSI_10105]	Log Task Preempt Notification

▽

△

Number	Heading
[SWS_OSI_10106]	Log Task Exit Notification
[SWS_OSI_10107]	Log Task Creation Notification
[SWS_OSI_10108]	Log Task Renaming Notification
[SWS_OSI_10110]	Log Process Switch Notification
[SWS_OSI_10111]	Log Process Creation Notification
[SWS_OSI_10112]	Log Process Ending Notification
[SWS_OSI_10113]	Log Process Renaming Notification

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

B.2.2 Changed Specification Items in R23-11

none

B.2.3 Deleted Specification Items in R23-11

none

B.3 Traceable item history of this document according to AUTOSAR Release R22-11

B.3.1 Added Advisories in R22-11

none

B.3.2 Changed Advisories in R22-11

none

B.3.3 Deleted Advisories in R22-11

none

B.3.4 Added Specification Items in R22-11

none

B.3.5 Changed Specification Items in R22-11

none

B.3.6 Deleted Specification Items in R22-11

none

B.3.7 Added Constraints in R22-11

none

B.3.8 Changed Constraints in R22-11

none

B.3.9 Deleted Constraints in R22-11

none

B.4 Traceable item history of this document according to AUTOSAR Release R19-11

B.4.1 Added Specification Items in R19-11

none

B.4.2 Changed Specification Items in R19-11

none

B.4.3 Deleted Specification Items in R19-11

none

B.4.4 Added Constraints in R19-11

none

B.4.5 Changed Constraints in R19-11

none

B.4.6 Deleted Constraints in R19-11

none