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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History			
Date	**Release**	**Changed by**	**Description**
2017-10-27	17-10	AUTOSAR Release Management	• Minor changes, document clean up
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

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

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 (OSI) 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 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 OSI 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 the Operating System Interface that are not included in the [2, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
OSI	Operating System Interface

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] Glossary
AUTOSAR_TR_Glossary

[3] Requirements on Operating System Interface
AUTOSAR_RS_OperatingSystemInterface

4 Constraints and assumptions

4.1 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 `Adaptive Platform` will evolve in future releases.

The following functionality is mentioned within this document but is not fully specified in this release:

- The currently known limitations are the requirements in [3] which are listed within Appendix A.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other modules

There are no dependencies to other `Adaptive Platform` standard modules. Any underlying modules required by OS such as bootloader, BSP (Board Support Package), HAL (Hardware Abstraction Layer), BIOS (Basic Input/Output System), and etc., are specific to the OS hence not standardized.

6 Requirements Tracing

The following table references the features specified in [3] and links to the fulfillments of these.

Feature	Description	Satisfied by
[RS_OSI_00100]	Operating System Interface	[SWS_OSI_01001] [SWS_OSI_01002]
[RS_OSI_00101]	Multi-threaded Execution	[SWS_OSI_01003]
[RS_OSI_00102]	Time-Triggered Execution	[SWS_OSI_01007]
[RS_OSI_00104]	Reaction on Application-external Stimuli from devices	[SWS_OSI_01011]
[RS_OSI_00105]	Start of Execution Management	[SWS_OSI_01040]
[RS_OSI_00200]	The Operating System shall support common best-effort real-time scheduling strategies on thread level within a process.	[SWS_OSI_01001] [SWS_OSI_01002]
[RS_OSI_00201]	The Operating System shall provide mechanisms for system memory budgeting.	[SWS_OSI_NA]
[RS_OSI_00202]	The Operating System shall provide mechanisms for CPU time budgeting.	[SWS_OSI_NA]
[RS_OSI_00203]	The Operating System should provide mechanisms for binding processes to CPU cores.	[SWS_OSI_NA]
[RS_OSI_00204]	The Operating System shall support authorized operating system object access for the software entities which are allowed to do so.	[SWS_OSI_NA]
[RS_OSI_00205]	The Operating Syste shall provide optimized mechanisms for running periodic, time-based loops.	[SWS_OSI_01001] [SWS_OSI_01002]
[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]

7 Functional specification

7.1 Operating System Specification

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.

7.1.2 Process handling

[SWS_OSI_01040] Start Execution Management as *init* process. ⌈ Based on the Operating System used, the mechanisms allowing the Execution Management to be executed as the *init* process, shall be used. ⌋*(RS_OSI_00105)*

[SWS_OSI_01006] Multi-Threading Support ⌈ The Operating System shall allow running multiple execution contexts (threads) such that the application can execute multiple code flows. ⌋*(RS_OSI_00206)*

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.

In general, a process provides at least the following:

- A `main()` function as the entry point of the first execution thread of the application
- A local memory context (address space), providing local, non-shared memory, that includes at least the code, data and heap of the application.
- 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 ⌈ The `Adaptive Platform Operating System` shall support multiple processes at any one time on the system. ⌋*(RS_OSI_00206)*

[SWS_OSI_01009] Multi-Process Isolation ⌈ The `Adaptive Platform Operating System` shall isolate each process from one another such that an incorrect or invalid access is detected by the `Adaptive Platform Operating System`. ⌋*(RS_OSI_00206)*

[SWS_OSI_01010] Virtual Memory ⌈ Each process running on the `Adaptive Platform Operating System` shall be executed in a dedicated address space. ⌋ *(RS_OSI_00206)*

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.

7.1.3 Scheduling Policies

The Operating System Scheduler is designed to keep all system resources busy allowing multiple software entities 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.

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

[SWS_OSI_01003] Default Scheduling Policies ⌈ The `Adaptive Platform Operating System` shall support the following scheduling policies defined in the IEEE1003.1 POSIX standard: SCHED_OTHER, SCHED_FIFO, SCHED_RR. ⌋ *(RS_OSI_00101)*

In order to overcome the above mentioned conflicts and to achieve portability between different platforms, the `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, addi-

tional 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

[SWS_OSI_01007] Time triggered execution ⌈ The OSI shall support time triggered execution of `Application`s using POSIX timers, signals and other synchronization primitives as defined in POSIX PSE51. ⌋*(RS_OSI_00102)*

7.1.5 Device support

[SWS_OSI_01011] Device support ⌈ The OSI shall support device access as defined in POSIX PSE51. ⌋*(RS_OSI_00104)*

7.2 API specification

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.

7.2.1 Application Interface C (POSIX PSE51)

[SWS_OSI_01001] Use of C Language ⌈ The OSI shall provide OS functionality with POSIX PSE51 interface for Applications written in C. ⌋*(RS_OSI_00100, RS_OSI_00200, RS_OSI_00205)*

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 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 Application may use PSE51 as OS interface whereas platform Application 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 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.

7.2.2 Application Interface C++11

[SWS_OSI_01002] Use of C++ Language ⌈ The OSI shall provide OS functionality with C++11 Standard Library for `Application`s written in C++. ⌋*(RS_OSI_00100, RS_OSI_00200, RS_OSI_00205)*

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 C++ applications 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.

A Not applicable requirements

[SWS_OSI_NA] ⌈ These requirements are not applicable as they are not within the scope of this release. ⌋*(RS_OSI_00201, RS_OSI_00202, RS_OSI_00203, RS_OSI_00204)*