Document Title	Requirements on Operating System
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	8

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

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Removed partition restart
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• Narrowed scope to cores running AUTOSAR OS
2021-11-25	R21-11	AUTOSAR Release Management	• Updated traceability
2020-11-30	R20-11	AUTOSAR Release Management	• Added ARTI requirements
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Incorporation of concept "AUTOSAR Run-Time Interface"
2017-12-08	4.3.1	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation

△

2016-11-30	4.3.0	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2014-10-31	4.2.1	AUTOSAR Release Management	• Incorporation of concept "Mechanisms and constraints to protect ASIL BSW against QM BSW"
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Requirements added to support power saving mode • Formal rework of requirement tracing • Updated according to TPS_standardization template as minor changes
2011-12-22	4.0.3	AUTOSAR Administration	• Merging of AUTOSAR_SRS_MultiCoreOS
2010-09-30	3.1.5	AUTOSAR Administration	• Removed requirement for the OS to ensure that interrupt priority registers are consistent with OS configuration • Various minor changes to descriptions to aid understanding • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
2006-11-28	2.1	AUTOSAR Administration	• Legal disclaimer revised
2006-05-16	2.0	AUTOSAR Administration	• Minor formal changes
2005-05-31	1.0	AUTOSAR Administration	• Initial Release

Disclaimer

Contents

1 Scope of this Document

The goal of this document is to define the high-level requirements for the AUTOSAR operating system.

2 How to Read this Document

2.1 Conventions to be used

- The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078] (see Standardization Template [1]).
- In requirements, the following specific semantics shall be used (based on the Internet Engineering Task Force IETF).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as:

- SHALL: This word means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- MUST: This word means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase means that the definition is an absolute prohibition of the specification due to legal constraints.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

2.2 Acronyms and abbreviations

Abbreviation	Description
API	Application Programming Interface
BSW	Basic Software
COM	Communications
ECU	Electronic Control Unit
HW	Hardware
ISR	Interrupt Service Routine
MC	Multi-Core
MCU	Microcontroller Unit
MPU	Memory Protection Unit
NM	Network Management
OIL	OSEK Implementation Language [2]
OS	Operating System
OSEK/VDX	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
SC	Single-Core
SW	Software
SWC	Software Component

3 Requirements Guidelines

Existing specifications shall be referenced (in form of a single requirement). Differences to these specifications are specified as additional requirements.

3.1 Requirements quality

All Requirements shall have the following properties:

- Redundancy

  Requirements shall not be repeated within one requirement or in other requirements

- Clearness

  All requirements shall allow one possibility of interpretation only. Only technical terms of the glossary [3] may be used. Furthermore, it must be clear from the requirement, what object the statement is a requirement on.

  Examples:

  - The <...> module shall/should/may ...
  - The <...> module's environment shall ...
  - The <...> configuration shall...
  - The function <...> shall ...
  - The <...> SWS shall ...
  - The hardware shall ...

- Atomicity

  Each Requirement shall only contain one requirement. A Requirement is atomic if it cannot be split up in further requirements.

- Testability

  Requirements shall be testable by analysis, review or test.

- Traceability

  The source and status of a requirement shall be visible at all times.

- Formulation

  All requirements shall be formulated so that they can be interpreted without the surrounding context (for example: "the function Xyz..." instead of "this function...").

3.2 Requirements identification

Each requirement has its unique identifier starting with BSW as prefix. For any review annotations, remarks and/or questions please refer to this unique ID rather than chapter or page numbers!

3.3 Requirements status

Additionally, each requirement contains a state information. The state can be one of the following.

State:	Description:
Open	The requirement has been created by a member of the WP but not yet discussed in the WP meeting.
Proposed	The requirement has been reviewed during the WP meeting. It is accepted, but there are still open issues pending.
Approved	The requirement has been reviewed and approved by all WP participants.
Conflict	The requirement has been reviewed, but there are conflicts (e.g. contradictions with other requirements) which could not be resolved so far.
Rejected	The requirement has been reviewed and rejected.

Thus, all requirements finalized are in the state "Approved".

4 Requirement Specification

4.1 Traceability

Feature no	Feature name
RS_BRF_01200	AUTOSAR OS shall be backwards compatible to OSEK OS
RS_BRF_01232	AUTOSAR OS shall support isolation and protection of application software
RS_BRF_01096	AUTOSAR shall support start-up and shutdown of ECUs
RS_BRF_01208	AUTOSAR OS shall support to start lists of tasks regularly
RS_BRF_01216	AUTOSAR OS shall support to synchronize ScheduleTables to an outside time source
RS_BRF_01240	AUTOSAR OS shall support communication between OSApplications
RS_BRF_02008	AUTOSAR shall provide mechanisms to protect the system from unauthorized read access
RS_BRF_01224	AUTOSAR OS shall support timing protection
RS_BRF_01248	AUTOSAR OS shall support to terminate and restart OSApplications
RS_BRF_01256	AUTOSAR OS shall offer support to switch off cores
RS_BRF_01264	AUTOSAR OS shall support multi-core deadlock free mutual exclusion
RS_BRF_01184	AUTOSAR shall support different methods of degradation
RS_BRF_00206	AUTOSAR shall support multi-core MCUs

4.2 Real-Time Operating System

4.2.1 Functional description

The real-time operating system in an embedded automotive ECU builds the basis for the dynamic behavior of the software. It manages the scheduling of tasks and events, the data flow between different tasks and provides features for monitoring and error handling.

However, in automotive systems the requirements on an operating system are highly domain specific. For instance, in the body, powertrain and chassis domains, the focus is on efficient scheduling of tasks and alarms, handling of shared resources and deadline monitoring. The used operating system has to be very efficient in runtime and small in memory footprint.

In multimedia and telematics applications, the feature set provided by the operating system and also the available computing resources are significantly different. Here, on top of pure task management, also complex data handling (e.g. streams, flash file systems, etc.), memory management and often even a graphical user interface are contained in the OS.

The classic domain of an automotive OS covers the core features of scheduling and synchronization only. In the AUTOSAR architecture the additional features discussed above are outside the scope of the OS. Such features are covered by the other AUTOSAR Basic Software Modules. (e.g. COM provides a communication abstraction). Integrating the feature sets of other OSs (e.g. QNX, VxWorks and Windows CE etc.)

into a monolithic OS/communication/drivers structure is not possible under architectural constraints of AUTOSAR. Therefore, the AUTOSAR OS shall consider only the core features.

4.2.2 Core Operating System requirements

[SRS_Os_00097] The OS shall provide an API that is backward compatible to the API of OSEK OS

Upstream requirements: RS_BRF_01200

⌈

Description:	The OS shall provide an API that is backward compatible to the API of OSEK OS [4]. Valid requirements shall be integrated as an extension of the functionality provided by OSEK OS.
Rationale:	Guarantee migration progress
Use Case:	Existing driver software can be reused as its interface to the OS is not changed.
Dependencies:	–
Supporting Material:	[STD_OSEK_OS], see [4]

⌋

[SRS_Os_11001] The OS shall provide partitions which allow for fault isolation capabilities

Upstream requirements: RS_BRF_01232, RS_BRF_01234

⌈

Description:	The OS shall provide partitions (Fault Containment Regions) which allow for fault isolation capabilities.
Rationale:	AUTOSAR permits mutliple logical applications to co-exist on the same processor. The existing specification of OSEK OS [4] is not aware of multiple logical applications residing on a single processor. There is therefore no facility for the containment of faults. A fault in one application could propagate to another application resident on the same processor. For example, an error in one software component or basic software module may result in a fault being detected in another software component and/or basic software module whose only relation to the faulty part is that it is resident on the same processor. OSEK OS has the following rules OS object manipulation: • Tasks and ISRs are the executable objects managed by the OS. • Standard resources can be manipulated by only those task/ISRs that declare this at configuration time.

▽

▽

△

	△ • Events can be set by any task or ISR. Events can only be waited on or cleared by those tasks that declare this at configuration time. • Alarms can be manipulated by any task or ISR. In AUTOSAR: • Extending this general scheme to table-based schedules ([SRS_Os_00098]) means that Schedule Tables can be manipulated by any task or ISR. This loose ownership of OS objects (tasks, ISRs, alarms, events, schedule tables, resources) makes it difficult to contain certain classes of faults at runtime, for example one software component incorrectly cancelling an alarm belonging to another software component. It is therefore necessary to define the relationship between OS objects and the software components or basic software module to which they belong so that fault containment can be achieved at runtime. The OS shall provide a higher-level abstraction to allow the user to group existing OS objects (tasks, ISRs etc.) so that objects in the group can be manipulated only by objects in the same group. Such a group is called an OS-Application. Furthermore, defining an OS-Application allows a memory protection domain to be provided (see [SRS_Os_11005]).
Use Case:	Under a failure condition the fault handling mechanism needs to stop all objects associated with a software component from executing.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_11018] The OS shall provide interrupt mask functions

Upstream requirements: RS_BRF_01096

⌈

Description:	The OS shall provide interrupt mask functions before calling StartOS() and after a ShutdownOS() call. These functions are already defined in OSEK OS [4] and the usage is now extended.
Rationale:	Needed by SPAL.
Use Case:	The SPAL drivers are required to manipulate the interrupt mask before, during and after normal OS operation.
Dependencies:	C initialization has to be performed before these functions can be used.
Supporting Material:	

⌋

[SRS_Os_11019] The AUTOSAR OS generation tool shall create the interrupt vector table ⌈

Description:	The AUTOSAR OS generation tool shall create the interrupt vector table for.
Rationale:	Each ECU will need to have an interrupt vector table. The operating system configuration already contains details about all interrupts used by the system. The AUTOSAR OS generation tool shall be the final tool in the development process that generates the interrupt vector table.
Use Case:	Integration of other modules.
Dependencies:	–
Supporting Material:	–

⌋

4.3 Statically Defined Scheduling

4.3.1 Functional Overview

In many applications it is necessary to statically define the activation of a set of tasks related to each other. This can be for guaranteeing data consistency in data-flow based designs, synchronizing with a time-triggered network, guaranteeing correct run-time phasing, etc.

A time-triggered operating system is often proposed as a solution to this problem. However, time is simply an event so any event triggered OS, including OSEK OS [4], can implement a scheduler for statically scheduled real-time software in automotive electronic control units.

The requirements for schedules tables provide an OSEK OS object that can be manipulated in the same way as an OSEKtime dispatcher table.

4.3.2 Requirements

[SRS_Os_00098] The Operating System shall provide statically configurable schedule tables based on time tables as an optional service

Upstream requirements: RS_BRF_01208

⌈

Description:	The Operating System shall provide statically configurable schedule tables based on time tables as an optional service.
Rationale:	Requirement of Standard Core users. Table based schedules are more efficient and easier to understand than tasks activated by OSEK alarm services. Adding a table-based scheduling mechanism approach as an extension to OSEK OS [4] provides users with the ability to construct an OSEKtime-like dispatcher table without needing to introduce the unnecessary restrictions of the stack-based scheduling policy or an additional OS specification.
Use Case:	Release a number of tasks synchronously with a statically defined inter-arrival time.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_00099] The Operating System shall provide a mechanism which allows switching between different schedule tables

Upstream requirements: RS_BRF_01208

⌈

Description:	The Operating System shall provide a mechanism which allows switching between different schedule tables.
Rationale:	For different application states (e.g. init, start-up, pre-start, normal operation, diagnosis, pre-sleep, shut down) different schedules are necessary.
Use Case:	ECU modes controlled by ECU State Manager
Dependencies:	[SRS_Os_00098]
Supporting Material:	–

⌋

[SRS_Os_11002] The operating system shall provide the ability to synchronize the processing of schedule tables with a global system time base

Upstream requirements: RS_BRF_01216

⌈

Description:	The operating system shall provide the ability to synchronize the processing of schedule tables with a global system time base. It shall support immediate (hard) synchronization and gradually adapting (smooth) synchronization.
Rationale:	It is necessary for some distributed applications to be synchronized to a global (to the relevant applications) timebase. This type of feature is needed for users coming to the AUTOSAR OS from OSEKtime.
Use Case:	Users migrating from OSEKtime dispatcher tables can replicate the similar functionality with schedule tables without needing to introduce an additional OS specification.
Dependencies:	–
Supporting Material:	

⌋

4.4 Monitoring Facilities

4.4.1 Functional Overview

A monitoring function detects an error at an appropriate stage in execution and not the instant that the error occurs. Consequently, any monitoring function is the detection of a failure at runtime rather than the prevention of a fault.

4.4.2 Requirements

[SRS_Os_11003] The operating system shall be able to monitor stack usage and check for a stack overflow on a per executable object basis

Upstream requirements: RS_BRF_01232

⌈

Description:	The operating system shall be able to monitor stack usage and check for a stack overflow on a per executable object basis (task/ISR).
Rationale:	On some hardware it will not be possible to implement any sophisticated memory protection. Stack monitoring provides an alternative (but less secure) solution where some protection is deemed better than none.

▽

△

Use Case:	If a system where an application could overflow its stack is implemented on hardware that cannot support true memory protection, stack monitoring is a useful alternative.
Dependencies:	–
Supporting Material:	–

⌋

4.5 Protection Facilities

4.5.1 Functional Overview

The AUTOSAR concept requires multiply-sourced OS-Applications to co-exist on the same processor. To prevent unexpected interaction between these OS-Applications it is necessary to provide mechanisms that protect them from one another. There are two major use-cases:

1. For a safety-critical system, the development of a safety-case is made much easier if individual OS-Application safety cases can be integrated into an overall safety case. This is only feasible if it can be demonstrated that at least a fault in one OS-Application cannot propagate beyond its own boundary and cause a fault in another, unrelated, OS-Application.
2. Suppliers can only be expected to take responsibility (and some liability) for their software components and/or basic software modules if they can be assured that their software cannot be incorrectly blamed for a processor-wide failure.

Both of these use-cases can be satisfied by the addition of protection mechanisms to OSEK OS [4]. The following sections outline the areas of protection.

4.5.2 Memory Protection requirements

[SRS_Os_11005] The operating system shall prevent an OS-Application from modifying the memory of other OS-Applications

Upstream requirements: RS_BRF_01232, RS_SAF_31201

⌈

Description:	The operating system shall provide the ability of partitioning OS-Applications with respect to memory and prevent an OS-Application from modifying the memory of other OS-Applications.

▽

△

Rationale:	Where multiple OS-Applications (of different software integrity) are resident on the same processor, their memory will be globally writable by any code. This means that the data of one OS-Application could be corrupted by another unrelated OS-Application (i.e. there is fault propagation between OS-Applications). For example a task of an OS-Application may overflow its stack, causing static data of an unrelated OS-Application to be corrupted, causing it to fail. To permit reasoning about adequate independence between the functions of different integrity levels, it is essential that this is prevented at runtime. Note that [SRS_Os_11003] is different: It only detects fault rather than preventing a memory access error from generating a fault.
Use Case:	–
Dependencies:	Note that satisfying this requirement implies the satisfaction of the stack monitoring requirement as a stack overflow cannot occur if the stack is bounded by memory write access control. The write access protection needs appropriate hardware support.
Supporting Material:	

⌋

[SRS_Os_11006] The operating system shall allow tasks and ISRs within an OS-Application to exchange data

Upstream requirements: RS_BRF_01240

⌈

Description:	The operating system shall allow tasks and ISRs within an OS-Application to exchange data using direct access to shared memory.
Rationale:	It is common to exchange data using shared memory for performance reasons at runtime (e.g. using global variables). However, in AUTOSAR multiple OS-Applications will share a processor and therefore any data communication that happens through shared memory breaks the memory protection scheme. Therefore, it is necessary to provide OS-Applications with the ability to share data using memory which is globally accessible to tasks and ISRs within the application but which is not accessible to other OS-Applications i.e. shared memory local to scope of an OS-Application.
Use Case:	An OS-Application implements communication and uses an ISR to handle the reception of CAN frames from the vehicle network but uses a task to process the contents of the CAN frame to reduce ISR level blocking.
Dependencies:	–
Supporting Material:	[DOC_WP112_REQ], see [5]

⌋

[SRS_Os_11007] The operating system shall allow OS-Applications to execute shared code

Upstream requirements: RS_BRF_01240

⌈

Description:	The operating system shall allow OS-Applications to execute shared code.
Rationale:	If code cannot be shared then any piece of software that is common to a number of software components/basic software modules will have to be included multiple times in a software build. This has two implications: 1. a large increase in code space 2. a problem is introduced for software maintenance as a modification to logically shared code will have to be made to every instance of the code in a build of an ECU (a single change has become multiple changes)
Use Case:	Using shared libraries.
Dependencies:	–
Supporting Material:	

⌋

[SRS_Os_11000] The OS may offer support to protect the memory sections of an OS-Application against read accesses by all other OS-Applications

Upstream requirements: RS_BRF_02008

⌈

Description:	The OS may offer support to protect the memory sections of an OS-Application against read accesses by all other OS-Applications.
Rationale:	If a task/ISR can read from any memory then it may operate on incorrect data. This could result in failures at runtime. Preventing read accesses provides a way of trapping such faults as soon as they occur. A secondary issue is security. While it is not anticipated that there are any security implications between OS-Applications on the same processor, read accesses does provide protection if required
Use Case:	Security: protect secret keys; Debugging support
Dependencies:	–
Supporting Material:	–

⌋

4.5.3 Timing Protection requirements

[SRS_Os_11008] Timing Fault Detection and Prevention of Propagation

Upstream requirements: RS_BRF_02056, RS_SAF_31202

⌈

Description:	The OS shall provide a mechanism to detect timing faults in OS-Applications and prevent propagation to a different OS-Application resident on the same processor to ensure freedom from interference. Additional information: A timing fault is defined as: • exceeding a specified execution time • exceeding a specified arrival rate
Rationale:	When these parameters are specified for every task/Category 2 ISR in the system it is possible to determine whether or not each task/ Category 2 ISR always meets its deadline. Timing correctness on an ECU running any fixed-priority pre-emptive OS, including OSEK OS [4], can only be guaranteed using schedulability analysis. This uses information about the tasks and interrupts (how often they run, how long they run for, which resources they access, how long they hold them for) and then calculates that the system will meet its real-time performance deadlines. The scope of timing protection is to ensure that an AUTOSAR system that has been shown to meet its deadlines does not violate the model used for analysis at runtime due to failures in the functional behavior of applications (or their constituent parts). Strict enforcement of the assumptions of the real-time performance analysis means two important things: 1. A timing fault is detected early, and hence can be picked up earlier in the software life cycle. For example, a faulty software component from a supplier can be rejected prior to full integration test. The costs of remedying a fault are therefore reduced. 2. A timing fault is not propagated. By detecting the fault as it occurs the effects of the fault are confined to the OS-Application where the fault occurred. Thus the problems of real-time failures induced in the wrong sub-system (or even the wrong ECU in a network) are eliminated.
Use Case:	An object in one OS-Application executing for too long, causes an object in another OS-Application to miss its deadline as a result.
Dependencies:	–
Supporting Material:	–

⌋

4.5.4 Service Protection requirements

The OS must preserve both its own integrity and the integrity of the OS-Applications that it schedules at runtime.

[SRS_Os_11009] The operating system shall prevent the corruption of the OS by any call of a system service

Upstream requirements: RS_BRF_01232

⌈

Description:	The operating system shall prevent the corruption of the OS by any call of a system service.
Rationale:	If it was possible to place the OS into an unknown state, or corrupt OS data structures at runtime then this would damage every OS-Application resident on the same processor. This means that either: • every OS service call must have defined behavior in all cases; or • the OS must not allow service calls to be made from contexts that would potentially result in the OS being placed into an undefined state. This increases the integrity of the OS itself.
Use Case:	Avoid undefined behavior from e.g. calling services from wrong context.
Dependencies:	In case the current specification of OSEK OS allows configurations which do not protect the OS the AUTOSAR configuration has to make sure that these configurations can not be selected.
Supporting Material:	–

⌋

[SRS_Os_11010] The operating system shall prevent an OS-Application modifying OS objects that are not owned by that OS-Application

Upstream requirements: RS_BRF_01232

⌈

Description:	The operating system shall prevent an OS-Application modifying OS objects that are not owned by that OS-Application.
Rationale:	An OS-Application could manipulate objects in another OS-Application that cause it to behave outside the scope of its design at runtime. Protecting the integrity of OS-Applications means that one OS-Application cannot manipulate an object owned by another OS-Application, for example through OS service calls, causing potential failure in another OS-Application, unless access to the object expressly granted at configuration time. This increases the ability to trace faults arising from OS-Application coupling by restricting the possible sources of the fault.
Use Case:	Canceling an alarm that activates a task in another OS-Application

▽

△

Dependencies:	In the case where the current specification of OSEK OS allows configurations which do not protect OS-Applications, the AUTOSAR configuration has to make sure that these configurations can not be selected.
Supporting Material:	–

⌋

[SRS_Os_11011] The OS shall protect itself against OS-Applications attempting to modify control registers directly which are managed by the OS

Upstream requirements: RS_BRF_01232

⌈

Description:	The OS shall protect itself against OS-Applications attempting to modify control registers directly which are managed by the OS.
Rationale:	The OS must be protected against OS-Applications attempting (directly or indirectly) to circumvent the protection mechanisms. Typically this means that OS-Applications should be prevented from accessing the MCU status registers and memory protection registers that might be in use.
Use Case:	OS uses the processor status word for managing interrupts and the register is written by a rogue OS-Application at runtime, corrupting the internal data structures of the OS.
Dependencies:	The target hardware must support privileged/non-privileged modes and a MPU for this protection to be possible. This feature will therefore not be available on those targets that do not provide sufficient hardware support.
Supporting Material:	–

⌋

[SRS_Os_11012] The OS shall provide scalability for its protection features

Upstream requirements: RS_BRF_01232

⌈

Description:	The OS shall provide scalability for its protection features.
Rationale:	Take full advantage of the processor's hardware features: The key protection features may not be available on all hardware (e.g. some types of memory protection are not possible when the processor has no MPU), but this should not prevent users for using the other protection features that can be supported. Customize to specific user's needs: Protection may only be necessary around some applications (ones where we cannot be sure of their run-time behavior) and protection can be applied selectively based on assessment of the risk of failure.

▽

△

Use Case:	Implementing an AUTOSAR compliant OS on a microcontroller without hardware memory protection. Where an ECU is engineered using a process that can statically guarantee that no protection violations will occur at runtime it does not need to dedicate resources to check for violations. For example, if worst-case execution times are statically analyzed then timing protection is not needed at runtime. Furthermore, because the analysis shows that the conditions that trigger execution of the code will never occur, the code is "dead code" and should be removed because of the potential safety risk it brings.
Dependencies:	–
Supporting Material:	–

⌋

4.5.5 Protection Errors

The OS must be able to identify when an error that violates the protection schemes has occurred and must also provide facilities through which action can be taken to correct the fault. However, it is not the task of the OS to define the error handling scheme.

[SRS_Os_11013] The OS shall be capable of notifying the occurrence of a protection error at runtime

Upstream requirements: RS_BRF_01232

⌈

Description:	The OS shall be capable of notifying the occurrence of a protection error at runtime. A protection error is any memory access violation, timing fault, unauthorized call to OS service or software trap (for example division by zero, illegal instruction).
Rationale:	If protection errors are notified at runtime this provides scope to potentially correct or handle the error according to a predefined fault handling strategy.
Use Case:	The application needs to provide some kind of runtime fault tolerance that needs to take action on the type and/or number of errors that occur to improve availability at runtime.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_11014] In case of a protection error, the OS shall provide an error reaction on OS-, OS-Application and task/ISR-level

Upstream requirements: RS_BRF_01248

⌈

Description:	In case of a protection error, the OS shall provide an error reaction on OS-, OS-Application and task/ISR-level. The user shall be able to select the action.
Rationale:	The action taken on the occurrence of an error is a function of the failure modes of the system as a whole. For example, in some cases it will be appropriate to simply terminate the faulty task, in others this may pose more of a risk to safety than allowing it to continue to execute. Therefore, the decision which action is appropriate is up to the application.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

4.6 Timer Services

4.6.1 Functional Overview

Timer Services provide software timers for use in application and basic software.

The core of a timing mechanism is already provided by the counters and alarms in OSEK OS [4]. Introducing an almost identical mechanism, in the form of Timer Services, is therefore unnecessary.

However, to provide general purpose software timing a few supplementary features need to be added to AUTOSAR OS. These are described in [SRS_Os_11020] and [SRS_Os_11021].

4.6.2 Functional Requirements

[SRS_Os_11020] The OS shall provide a standard interface to tick a software counter ⌈

Description:	The OS shall provide a standard interface to tick a software counter.
Rationale:	OSEK OS [4] does not define the interface between counters and alarms. This creates a problem when porting applications between different vendors' implementations. Defining this interface in AUTOSAR OS removes this portability problem.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_11021] The OS shall provide a mechanism to cascade multiple software counters from a single hardware counter. ⌈

Description:	The OS shall provide a mechanism to cascade multiple software counters from a single hardware counter.
Rationale:	If counters with different resolutions are required it may not be possible (e.g. because of limited hardware timers) or desirable (e.g. because of interrupt interference) to use multiple hardware timer sources. In many cases a lower resolution software counter can be driven from a higher resolution counter by ticking the lower resolution counter from the higher resolution counter.
Use Case:	Drive a 1ms software counter by a 1ms timer interrupt and a 100ms counter from the 1ms counter.
Dependencies:	This requirement implies that an implementation must support more than one counter (otherwise cascading would not be possible). Specification of the lower limit on the number of counters that must be supported by an implementation is provided in the AUTOSAR OS SWS.
Supporting Material:	–

⌋

4.7 Scalability

4.7.1 Functional Overview

For a particular application, the operating system can be configured such that it only comprises the services required for this application. Thus the resource requirements of the operating system are as small as possible.

Scalability for the core OS is provided by OSEK OS's [4] conformance classes. Scalability with regard to AUTOSAR OS features is defined here.

4.7.2 Functional Requirements

[SRS_Os_11016] The OS implementation shall offer scalability which is configurable by a generation tool

Upstream requirements: RS_BRF_01232

⌈

Description:	The OS shall provide the following configurations with at least the specified features (additional features may be included): Class1 : OSEK OS [4] + Planned Schedules Class2 : Class1 + Timing Protection Class3 : Class1 + Memory Protection Class4 : Class1+ Class2 + Class3
Rationale:	Hardware support is required for Classes 3 and 4. Mandating this functionality would prevent AUTOSAR OS from being implemented on many commonly used microcontrollers. Implementations may choose different strategies for implementation, with a corresponding increase in performance, if some features are not required.
Use Case:	–
Dependencies:	[SRS_Os_11012]
Supporting Material:	–

⌋

4.8 Application Error Handling

4.8.1 Functional Overview

Some errors affecting an application may not lead to protection violations that are detectable by the OS but nevertheless result in an application entering a state where it cannot itself recover and continue executing. The detection of such errors is the responsibility of the application itself, but recovery from such errors requires interaction with the threads of control managed by the OS. The OS therefore needs to provide a mechanism upon which applications can build internal recovery mechanisms.

The core OSEK OS [4] provides some support: tasks can detect internal errors and terminate themselves; the system can use the alarm mechanism to program timeouts, implement threshold-based error detection; internal errors can be detected and the

system can be shut down etc. However, the core OS does not provide the means to implement error recovery for logical applications that are defined by OS-Applications (see [SRS_Os_11001]).

This section introduces requirements to provide a framework of application-level and system-level control of OS-Applications.

4.8.2 Functional Requirements

[SRS_Os_11022] The OS shall provide a mechanism to terminate OS-Application

Upstream requirements: RS_BRF_01248

⌈

Description:	The OS shall provide a mechanism by which an OS-Application can be terminated as a single unit and all resources held by the OS-Application and managed by the OS are released. Termination shall prevent any task owned by the OS-Application from running and any interrupt handled by an ISR owned by the OS-Application from occuring.
Rationale:	Error handling for AUTOSAR software components requires that the OS can terminate an OS-Application. When an application comprises multiple tasks/ISRs it is not possible in the core OS to stop the application atomically unless the OS itself is shut down. Shutdown is not practical when there are other OS-Applications for which termination is not required. Therefore, the OS must provide a mechanism to terminate an OS-Application that does not affect other OS-Applications.
Use Case:	Error reaction in an OS-Application in response to the detection of an internal error.
Dependencies:	–
Supporting Material:	–

⌋

4.9 General Multi-Core issues

4.9.1 Overview

The requirements in this section are very general. They define the spirit of the Multi-Core (MC) capability that will be supported by the AUTOSAR environment. From an architectural point of view, Multi-Core hardware can be managed in various different ways. On the one extreme, cores might be understood as almost separate ECU, on

the other extreme, they can be presented to the user almost like a Single-Core (SC) system with the capability of true parallelism.

In automotive systems the requirements on the MC support is very domain specific. Efficient scheduling, low resource consumption and short response time are essential.

The requirements are designed in a way that the introduction of Multi-Core does not change the overall AUTOSAR philosophy.

The MC concept allows to handle several cores almost like a SC system, but gives freedom to use cores outside this concept, e.g. as a dedicated I/O controller.

4.9.2 Functional Requirements

[SRS_Os_80001] The OS shall be able to manage multiple closely coupled CPU Cores

Upstream requirements: RS_BRF_00206

⌈

Description:	The OS shall be able to manage multiple closely coupled CPU Cores. That does not imply that all cores on a μC are controlled by the OS.
Rationale:	Reasons to provide a solution with one OS that controls multiple cores are: • Enables efficient parallelization of functions. • Upward and downward scalability in number of cores. • Allows the restriction of the AUTOSAR Multi-Core extensions to a subset of available cores to run other OS instances on uncontrolled cores.
Use Case:	• Applications (e.g. signal processing applications) with the need to achieve high performance computing via algorithm parallelization. • Multi core systems with common BSW. • Applications that grow beyond the boundary of the given number of cores (e.g. one) can easily utilize a higher number of cores (upward scalability). • Applications designed for multiple cores can be stripped down (e.g. for low cost systems) to fewer (e.g. one) cores (downward scalability). • Migration of engine control systems to Multi-Core. • Integration of formerly separated applications into one Multi-Core ECU.
Dependencies:	[SRS_Os_80008]
Supporting Material:	

⌋

[SRS_Os_80003] The multi core extension shall provide the same degree of predictability as the single core

Upstream requirements: RS_BRF_00206

⌈

Description:	The multi core extension shall provide the same degree of predictability as the single core. This covers deadlock free execution and freedom from unbounded blocking.
Rationale:	Real-time capability is a key requirement of the automotive domain. The existing SC solution is designed in a way that their usage cannot cause unbounded blocking and guarantees deadlock free execution. The MC solution shall behave in a similar manner.
Use Case:	–
Dependencies:	[SRS_Os_80005]
Supporting Material:	–

⌋

4.9.3 Additional description of the term "One AUTOSAR system controlling multiple cores"

When talking about an AUTOSAR system that is controlling multiple cores as in [SRS_Os_80001] several implications are made:

- The system shall be aware of the existence of multiple cores.
- The system shall be responsible for the scheduling of tasks on multiple cores.
- Parts of the system code shall be able to run concurrently (e.g. by using reentrant code).
- All BSW IDs (e.g. IDs of tasks, events, alarms, ...) shall be unique across cores.
- It shall be allowed to access shared objects (e.g. data, peripheral units, ...) from any core, unless restricted by protection mechanisms.

4.9.4 Additional description of the term "unbounded blocking"

Blocking is a situation where a high priority runtime object can not execute because a low priority runtime object prevents this (e.g. by occupying a needed resource).

Unintentional blocking might be caused by priority inversion.

The term unbounded blocking means that the potential blocking duration is not limited and therefore the required real-time behaviour cannot be guaranteed.

4.10 Assignment of runtime objects to cores

4.10.1 Overview

One major question when defining a Multi-Core system is whether runtime objects (tasks and ISR) can change between cores dynamically or not.

The ability to dynamically assign runtime objects to different cores would have a massive impact on all efficiency aspects (code size / data size / speed / response time / real time capability) of a system.

It is conceivable to bind OsApplications to cores or to define the core binding on the level of TASKS and ISRs. To minimize the impact and complexity of the Multi-Core support within AUTOSAR it was decided to define the core binding the level of Os Applications.

This section defines requirements, that state core binding as the way to handle runtime objects within a MC AUTOSAR environment.

4.10.2 Requirements

[SRS_Os_80005] OsApplications and as a result TASKS and OsISRs shall be assigned statically to cores

Upstream requirements: RS_BRF_00206

⌈

Description:	OsApplications and as a result TASKS and OsISRs shall be assigned statically to cores.
Rationale:	• If TASKS or OsISRs can change the core during runtime, the real-time capability might be violated. • If tasks of a single OsApplication can be bound to different cores the shutdown of an OsApplication becomes hard. Valid mechanisms would be required. • To fulfill requirement [BSW00009] and [BSW00010] it shall be possible to access EVENTS and TASKS of different OsApplications. • In case of Multi-Core OSApplications shall be used irrespective of the scalability class. (see AUTOSAR_SWS_OS).
Use Case:	–
Dependencies:	OS specification; [SRS_Os_80003], [SRS_Os_80015], [SRS_Os_80016]
Supporting Material:	AUTOSAR_SWS_OS

⌋

4.11 Startup of Multi-Core systems

4.11.1 Overview

This section contains some high-level requirements on the start-up.

Depending on the used microcontroller, the start-up or reset behavior of the microcontrollers can differ. The most common behavior after a reset is as follows:

- Only a so-called master core starts to execute while all other cores (slave cores) remain in *halt* state. The slave cores need to be started by the master core.
- A different approach is conceivable with all cores starting to execute concurrently after a reset. A master core does not exist in this case.

The wakeup mechanism and bootstrap requirements vary across different microcontrollers and microcontroller derivates.

The progress in the start-up code on the different cores is not reproducible; this is because the length of time for load and store operations depends on bus arbitration and other very timing sensitive effects of the HW. Therefore the boot code must be designed in a way that it does not rely on the knowledge of the boot progress on other core(s). It is required to synchronize the different cores during start-up.

4.11.2 Requirements

[SRS_Os_80006] Initialization/Start-up of the system shall be synchronized

Upstream requirements: RS_BRF_00206

⌈

Description:	Initialization/Start-up of the system shall be synchronized.
Rationale:	To support a wide spectrum of hardware it is necessary to synchronize the software of the different cores at certain points. Otherwise one cannot rely on the state of the other core. (While one core already executes a task, the other is still in the initialization phase.)
Use Case:	Boot strapping of a MC system.
Dependencies:	• OS specification • ECU State Manager • HW • Applies only to cores managed by AUTOSAR Os.
Supporting Material:	–

⌋

4.12 Shutdown of Multi-Core systems

4.12.1 Overview

Similar to the startup, the shutdown behavior of a Multi-Core system is different from the behavior of a Single-Core system.

If a run time object with the proper rights calls "`ShutdownOS`", the whole system (all cores controlled by the MC-OS) have to shut down. Once the shutdown procedure has been started, validtasks cannot be activated. It is in the responsibility of the developer/ system integrator to make sure that any preparations for shutdown on application and basic software level are completed before calling "`ShutdownOS`".

4.12.2 Requirements

[SRS_Os_80007] Shutdown procedure shall be triggered by any core

Upstream requirements: RS_BRF_00206

⌈

Description:	The shutdown procedure can be triggered from any core.
Rationale:	In case of an error, the related handler may require system shutdown. This must be possible by any core.
Use Case:	Protection hook returns PRO_SHUTDOWN
Dependencies:	OS specification
Supporting Material:	–

⌋

4.13 Configuration of Multi-Core systems

4.13.1 Overview

This section contains high-level requirements on the system configuration with respect to Multi-Core.

4.13.2 Requirements

[SRS_Os_80008] It shall be a common OS configuration across multiple cores

Upstream requirements: RS_BRF_00206

⌈

Description:	Disjunctive object IDs have to be generated, if objects are to be addressed across cores.
Rationale:	If, e.g. tasks are activated across cores, IDs have to be unique across cores. This results in a common configuration and affects flashing/programming strategies.
Use Case:	Activating tasks or setting events across cores
Dependencies:	OS specification [SRS_Os_80001] [SRS_Os_80015] [SRS_Os_80016]
Supporting Material:	Multi-Core Concept document

⌋

[SRS_Os_80011] The number of cores that the operating system manages shall be configurable offline

Upstream requirements: RS_BRF_00206

⌈

Description:	The number of cores that the operating system manages shall be configurable offline.
Rationale:	The operating system specification shall not be limited to a certain number of cores.
Use Case:	Use of the operating system in projects with different numbers of cores.
Dependencies:	• Configuration specification (e.g. System template) • Boot procedure (e.g. ECU State manager)
Supporting Material:	–

⌋

4.14 Services in Multi-Core systems

4.14.1 Overview

The following chapter defines a set of mechanisms / services that allow optimal usage of a Multi-Core environment. The services might be provided by different AUTOSAR BSW modules. The AUTOSAR_SWS_Multi-Core defines from which module which service can be accessed.

4.14.2 Requirements

[SRS_Os_80013] The behaviour of services shall be identical to single core systems

Upstream requirements: RS_BRF_00206

⌈

Description:	The behaviour of services (e.g. task activation) should be identical to single core systems when the originating and the manipulated object (e.g. a task) reside on the same core.
Rationale:	Known services for SC systems should behave identically on a MC system when used locally, i.e. without crossing core boundaries.
Use Case:	–
Dependencies:	OS Specification, BSW Specifications
Supporting Material:	

⌋

[SRS_Os_80015] The MC extensions shall provide a mechanism to activate tasks on different cores

Upstream requirements: RS_BRF_00206

⌈

Description:	The MC extensions shall provide a mechanism to activate tasks on different cores, in different OsApplications.
Rationale:	The offline relocation of tasks between cores in different projects (e.g. low cost and high-end vehicles) shall be possible without reprogramming all task activations. Moreover, it shall be possible for the system integrator to assign sub-functionality to a core with free processing power.
Use Case:	Usage of third-party SW delivered as object code.
Dependencies:	OS Specification

▽

△

Supporting Material:	–

⌋

[SRS_Os_80016] Event mechanism shall work across cores

Upstream requirements: RS_BRF_00206

⌈

Description:	The MC extensions shall provide a mechanism to send an event to a task on a different core, in different OsApplications
Rationale:	If events are used and a task is moved to a different core (offline), it shall still be possible to use events.
Use Case:	Monitoring/safety concept.
Dependencies:	OS Specification
Supporting Material:	

⌋

[SRS_Os_80018] A method to synchronize tasks on more than one core shall be provided

Upstream requirements: RS_BRF_00206

⌈

Description:	A method to synchronize tasks on more than one core shall be provided.
Rationale:	Necessity to synchronize tasks across cores in time. This can be done by several means, e.g. alarms activating tasks across cores, by synchronizing counters or by using shared hardware timers.
Use Case:	Synchronized applications
Dependencies:	OS Specification
Supporting Material:	–

⌋

[SRS_Os_80020] A data exchange mechanism shall be provided

Upstream requirements: RS_BRF_01240

⌈

Description:	A data exchange mechanism shall be provided that guaranties data consistency independent from the HW
Rationale:	To minimize the HW dependency of the RTE a exchange mechanism is required that can be used by the RTE.
Use Case:	Data exchange in a MC system.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_80021] The MC extension of the AUTOSAR environment shall support a mutual exclusion mechanism between cores that shall not cause deadlocks

Upstream requirements: RS_BRF_01264

⌈

Description:	The MC extension of the AUTOSAR environment shall support a mutual exclusion mechanism between cores that shall not cause deadlocks, if configured and used properly. The mechanism shall be usable from task and ISR level.
Rationale:	In a MC system, a mutual exclusion mechanism is needed, to synchronize different cores. This mutual exclusion mechanism shall support the user to prevent from building deadlocks.
Use Case:	Concurrent access to shared resources
Dependencies:	• OS specification • HW support required
Supporting Material:	–

⌋

[SRS_Os_80022] In case no task is going to be scheduled on a specific core, the OS shall execute a user selectable operation

Upstream requirements: RS_BRF_01184

⌈

Description:	In case no task is going to be scheduled on a specific core, the OS shall execute a user selectable operation.
Rationale:	In order to set a core in a low power mode independently from the others, an indirect approach is used. Instead of explicitly requesting a core to HALT, a mechanism similar to the rubber band principle implemented in some modules such as ECUM is considered: the core remains in normal mode as long as its activity is required by some task allocated on it and is halted as soon as no task is in RUNNING or READY state. The core can be woken up by a SW interrupt (managed by OS) or by a HW interrupt.
Use Case:	Reduction of energy consumption by setting unused cores temporarily in a power saving mode
Dependencies:	• OS specification • HW support required
Supporting Material:	–

⌋

[SRS_Os_80023] The OS shall execute an operation which can be selected at runtime, in case no task is going to be scheduled on a specific core

Upstream requirements: RS_BRF_01184

⌈

Description:	The OS shall execute an operation which can be selected at runtime, in case no task is going to be scheduled on a specific core
Rationale:	OS shall offer different options as for the actions to be taken when the conditions at [SRS_Os_80022] are met. It shall be possible to define different actions, ranging from the predefined NO_HALT mode (no action taken, the core is left to run) to a number of OS and HW specific options, defined by the OS-vendor, which set the core in a HALT state.
Use Case:	Reduction of energy consumption by setting unused cores temporarily in a power saving mode
Dependencies:	• OS specification • HW support required
Supporting Material:	–

⌋

4.15 Debugging and Tracing

4.15.1 ARTI Support

[SRS_Os_12001] The OS shall create an ARTI module description file ⌈

Description:	If the OS configuration is set to use ARTI, the OS generator shall create an ARTI module description file
Rationale:	Debugging tools need internal information to visualize the state of the software. Components and modules implementing this requirement shall provide the necessary state information that can be used by internal and external tools.
Use Case:	Debugging the software.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Os_12003] ARTI module description file shall support all ORTI containers

Upstream requirements: RS_Main_01025

⌈

Description:	All information specified in the OSEK ORTI standard shall be provided in the ARTI module description file.
Rationale:	To maintain backward compatibility of OSEK-Realtime-Interface
Use Case:	Displaying information of the state of an application during debugging.
Dependencies:	[RS_Main_01025] AUTOSAR shall support debugging of software on the target and onboard
Supporting Material:	–

⌋

4.15.2 Tracing Support

[SRS_Os_12002] The OS code shall incorporate ARTI hooks ⌈

Description:	If the OS configuration is set to use ARTI, its code shall incorporate ARTI hooks. The OS generator shall expose the hooks and the traceable variables in an ARTI module description.

▽

△

Rationale:	Tracing and timing analysis tools need internal information to visualize and inspect the run-time behavior of the software. Components and modules implementing this requirement shall provide the necessary details and hooks that can be used by tools.
Use Case:	Run-time tracing the software, profiling, timing measurement.
Dependencies:	This requirement depends on the requirement "The generator of a module shall create an ARTI module description file".
Supporting Material:	–

⌋

5 Requirements Tracing

Requirement	Description	Satisfied by
[RS_BRF_00206]	AUTOSAR shall support multi-core MCUs	[SRS_Os_80001] [SRS_Os_80003] [SRS_Os_80005] [SRS_Os_80006] [SRS_Os_80007] [SRS_Os_80008] [SRS_Os_80011] [SRS_Os_80013] [SRS_Os_80015] [SRS_Os_80016] [SRS_Os_80018]
[RS_BRF_01096]	AUTOSAR shall support start-up and shutdown of ECUs	[SRS_Os_11018]
[RS_BRF_01184]	AUTOSAR shall support different methods of degradation	[SRS_Os_80022] [SRS_Os_80023]
[RS_BRF_01200]	AUTOSAR OS shall be backwards compatible to OSEK OS	[SRS_Os_00097]
[RS_BRF_01208]	AUTOSAR OS shall support to start lists of tasks regularly	[SRS_Os_00098] [SRS_Os_00099]
[RS_BRF_01216]	AUTOSAR OS shall support to synchronize ScheduleTables to an outside time source	[SRS_Os_11002]
[RS_BRF_01232]	AUTOSAR OS shall support isolation and protection of application software and BSW	[SRS_Os_11001] [SRS_Os_11003] [SRS_Os_11005] [SRS_Os_11009] [SRS_Os_11010] [SRS_Os_11011] [SRS_Os_11012] [SRS_Os_11013] [SRS_Os_11016]
[RS_BRF_01234]	AUTOSAR OS shall support isolation and protection between BSW modules	[SRS_Os_11001]
[RS_BRF_01240]	AUTOSAR OS shall support communication between OSApplications	[SRS_Os_11006] [SRS_Os_11007] [SRS_Os_80020]
[RS_BRF_01248]	AUTOSAR OS shall support to terminate and restart OSApplications	[SRS_Os_11014] [SRS_Os_11022]
[RS_BRF_01264]	AUTOSAR OS shall support multi-core deadlock free mutual exclusion	[SRS_Os_80021]
[RS_BRF_02008]	AUTOSAR shall provide mechanisms to protect the system from unauthorized read access	[SRS_Os_11000]
[RS_BRF_02056]	AUTOSAR OS shall support timing protection	[SRS_Os_11008]
[RS_Main_01025]	AUTOSAR shall support debugging of software on the target and onboard	[SRS_Os_12003]
[RS_SAF_31201]	Memory Protection of Applications	[SRS_Os_11005]
[RS_SAF_31202]	Timing Protection of Applications	[SRS_Os_11008]

Table 5.1: Requirements Tracing

6 References

[1] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[2] ISO 17356-6: Road vehicles – Open interface for embedded automotive applications – Part 6: OSEK/VDX Implementation Language (OIL)

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] ISO 17356-3: Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS)

[5] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

A Change history of AUTOSAR traceable items

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

A.1.1 Added Requirements in R24-11

none

A.1.2 Changed Requirements in R24-11

Number	Heading
[SRS_Os_11001]	The OS shall provide partitions which allow for fault isolation capabilities
[SRS_Os_11008]	Timing Fault Detection and Prevention of Propagation
[SRS_Os_11014]	In case of a protection error, the OS shall provide an error reaction on OS-, OS-Application and task/ISR-level
[SRS_Os_11022]	The OS shall provide a mechanism to terminate OS-Application

Table A.1: Changed Requirements in R24-11

A.1.3 Deleted Requirements in R24-11

Number	Heading
[SRS_Os_11023]	The OS shall provide a mechanism by which a terminated OS-Application can be restarted

Table A.2: Deleted Requirements in R24-11

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

A.2.1 Added Requirements in R23-11

none

A.2.2 Changed Requirements in R23-11

none

A.2.3 Deleted Requirements in R23-11

none

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

A.3.1 Added Requirements in R22-11

none

A.3.2 Changed Requirements in R22-11

Number	Heading
[SRS_Os_12001]	The OS shall create an ARTI module description file
[SRS_Os_12002]	The OS code shall incorporate ARTI hooks
[SRS_Os_80006]	Initialization/Start-up of the system shall be synchronized

Table A.3: Changed Requirements in R22-11

A.3.3 Deleted Requirements in R22-11

Number	Heading
[SRS_Os_80026]	It shall be possible to start any of the cores in a multi core system
[SRS_Os_80027]	It shall be possible to initialize any of the cores in a multi core system

Table A.4: Deleted Requirements in R22-11