Document Title	Requirements on Debugging, Tracing and Profiling support of AUTOSAR Components
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	916

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	• Removal of "transient faults"
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• Added requirements for tracing runnables, schedulables, VFB tracing hooks and BSW module entries/exits.
2021-11-25	R21-11	AUTOSAR Release Management	• Added functional requirements on Tracing • Removed some superfluous requirements • Adjusted spec items to module name
2020-11-30	R20-11	AUTOSAR Release Management	• Changed document status from draft to valid
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	• Initial release

Disclaimer

Contents

1 Scope of Document

This document refines the requirements specified in Foundation [1]. It focuses on requirements special to the Classic Platform of AUTOSAR. See also chapter "Scope of Document" of [1].

2 Conventions used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([2]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([2]).

3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to this document that are not included in the AUTOSAR Glossary [3].

Abbreviation / Acronym:	Description:
ARTI	AUTOSAR Run Time Interface
OS	Operating System
SWC	Software Component

Table 3.1: Acronyms and Abbreviations

4 Requirements Specification

This chapter describes all requirements driving the work to define the ARTI extensions in Classic Platform.

4.1 Functional Overview

This document refines the requirements specified in the Foundation [1]. It focuses on requirements special to the Classic Platform of AUTOSAR. See also chapter "Functional Overview" of [1].

4.2 Functional Requirements on Tracing

[RS_Arti_00029] AUTOSAR shall support recording timing events of application states

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of application state changes.
Rationale:	Provide a standardized way to trace timing events on application state changes to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to application state changes.
Supporting Material:	-

⌋

[RS_Arti_00030] AUTOSAR shall support recording timing events of tasks

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of task state changes.
Rationale:	Provide a standardized way to trace timing events on task state changes to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to task state changes.
Supporting Material:	-

⌋

[RS_Arti_00031] AUTOSAR shall support recording timing events of category 2 interrupt states

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of category 2 interrupt state changes.
Rationale:	Provide a standardized way to trace timing events on category 2 interrupt state changes to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to category 2 interrupt state changes.
Supporting Material:	-

⌋

[RS_Arti_00032] AUTOSAR shall support recording timing events of service calls

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of service call entries and exits.
Rationale:	Provide a standardized way to trace timing events on service call entries and exits to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to service calls.
Supporting Material:	-

⌋

[RS_Arti_00033] AUTOSAR shall support recording timing events of spinlock states

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of spinlock state changes.
Rationale:	Provide a standardized way to trace timing events on spinlock state changes to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to spinlock state changes.
Supporting Material:	-

⌋

[RS_Arti_00034] AUTOSAR shall support recording timing events of protection hooks

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of protection hooks.
Rationale:	Provide a standardized way to trace timing events on protection hooks to ensure tool compatibility.
Dependencies:	–
Use Case:	Add a trace instrumentation to protection hooks.
Supporting Material:	-

⌋

[RS_Arti_00035] AUTOSAR shall support tracing of arbitrary intervals between a start and a stop event

Upstream requirements: RS_Main_01026

⌈

Description:	The tracing of arbitrary intervals between a start and a stop event shall be supported.
Rationale:	Provide a standardized way to define this to ensure tool compatibility.
Dependencies:	–
Use Case:	Add user defined instrumentation to software components.
Supporting Material:	-

⌋

[RS_Arti_00036] AUTOSAR shall support tracing of arbitrary intervals between a start and several stop events

Upstream requirements: RS_Main_01026

⌈

Description:	The tracing of arbitrary intervals between a start and several stop events shall be supported, with the aim to provide insides to a dataflow.
Rationale:	Provide a standardized way to define this to ensure tool compatibility.
Dependencies:	–
Use Case:	Add user defined instrumentation to software components.
Supporting Material:	-

⌋

[RS_Arti_00037] AUTOSAR shall support tracing of arbitrary values

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The tracing of arbitrary values shall be supported.
Rationale:	Provide a standardized way to define this to ensure tool compatibility.
Dependencies:	–
Use Case:	Add user defined instrumentation to software components.
Supporting Material:	-

⌋

[RS_Arti_00038] AUTOSAR shall support tracing of category 1 interrupts.

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support the recording and tracing of category 1 interrupts.
Rationale:	Provide a standardized way to trace timing events on category 1 interrupts to ensure tool compatibility and compatibility to TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to category 1 interrupts.
Supporting Material:	-

⌋

[RS_Arti_00039] AUTOSAR shall support recording timing events of runnable entities.

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support recording and tracing of runnable entities state changes.
Rationale:	Provide standardized way to trace timing events on runnable entities state changes to ensure tool compatibility and compatibility with TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to runnable entities state changes.
Supporting Material:	-

⌋

[RS_Arti_00040] AUTOSAR shall support recording timing events of schedulable entities.

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support recording and tracing of schedulable entities state changes.
Rationale:	Provide standardized way to trace timing events on schedulable entities state changes to ensure tool compatibility and compatibility with TIMEX.
Dependencies:	–
Use Case:	Add a trace instrumentation to schedulable entities state changes.
Supporting Material:	-

⌋

[RS_Arti_00041] AUTOSAR shall support recording events from the standardized VFB tracing interface.

Upstream requirements: RS_Main_00340, RS_Main_01026

⌈

Description:	AUTOSAR shall support recording events from the standardized VFB tracing interface.
Rationale:	Provide a standarized way to support the VFB tracing interface with ARTI.
Dependencies:	[SRS_Rte_00045]
Use Case:	Add ARTI trace instrumentation to VFB tracing events.
Supporting Material:	-

⌋

[RS_Arti_00042] AUTOSAR shall support tracing of entries and exits of BSW modules.

Upstream requirements: RS_Main_01026

⌈

Description:	AUTOSAR shall support recording and tracing of entry and exit events of BSW modules.
Rationale:	Provide a standardized way to trace timing events on BSW entries and exits to ensure tool compatibility.
Dependencies:	[SRS_Rte_00045]
Use Case:	Add a trace instrumentation to BSW module API functions.

▽

△

Supporting Material:	-

⌋

4.3 Functional Requirements on ARTI Template

This chapter defines the requirements on the ARTI template.

[RS_Arti_00001] The ARTI template shall support core specific ARTI additions

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a mechanism to allow core specific ARTI parameters.
Rationale:	ARTI needs core specific evaluations. The Template shall define a “class” to define additional parameters to cores as well as an “instance” to define values of specific cores. The core instance shall include a reference to the core definition in the EcuC description.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific core related information to display and trace the core activity.
Supporting Material:	-

⌋

[RS_Arti_00002] The ARTI template shall support a parameter for the current application

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a parameter that contains the evaluation for the “current application” that is running on a specific core.
Rationale:	ARTI needs to know which application is running at a core at a specific time.
Dependencies:	[RS_Arti_00001]
Use Case:	Debuggers and Tracing tools need to know the current application to display and trace the core activity.

▽

△

Supporting Material:	-

⌋

[RS_Arti_00003] The ARTI template shall support a parameter for the current task

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a parameter that contains the evaluation for the "current task" that is running on a specific core.
Rationale:	ARTI needs to know which task is running at a core at a specific time.
Dependencies:	[RS_Arti_00001]
Use Case:	Debuggers and Tracing tools need to know the current task to display and trace the core activity.
Supporting Material:	-

⌋

[RS_Arti_00004] The ARTI template shall support a parameter for the last error

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a parameter that contains the evaluation for the "last error" that happened on a specific core.
Rationale:	ARTI needs to know which error happened at a core at a specific time.
Dependencies:	[RS_Arti_00001]
Use Case:	Debuggers and Tracing tools need to know the last error to display and trace the core activity.
Supporting Material:	-

⌋

[RS_Arti_00005] The ARTI template shall support OS specific ARTI additions

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a mechanism to allow OS specific ARTI parameters.
Rationale:	ARTI needs OS specific evaluations. The Template shall define a “class” to define additional parameters to an OS as well as an “instance” to define values of the OS. The OS instance shall include a reference to the OS definition in the EcuC description.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific OS related information to display and trace the OS activity.
Supporting Material:	-

⌋

[RS_Arti_00007] The ARTI template shall support task specific ARTI additions

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a mechanism to allow task specific ARTI parameters.
Rationale:	ARTI needs task specific evaluations. The Template shall define a “class” to define additional parameters to a task as well as an “instance” to define values of the task. The task instance shall include a reference to the task definition in the EcuC description.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific task related information to display and trace the task activity.
Supporting Material:	-

⌋

[RS_Arti_00008] The ARTI template shall support SWC specific ARTI additions

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	The ARTI template shall define a mechanism to allow SWC specific ARTI parameters.
Rationale:	ARTI needs SWC specific evaluations. The Template shall define an “instance” to define values of an SWC. The SWC instance shall include a reference to the SWC definition in the EcuC description.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific SWC related information to display and trace the SWC activity.
Supporting Material:	-

⌋

4.4 Functional Requirements on ARTI Description

This chapter defines the requirements on the ARTI description.

[RS_Arti_00009] The ARTI description shall include a core class definition.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameters to a core are collected in a class definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know the layout of a core class used by this implementation.
Dependencies:	–
Use Case:	Evaluating the form of display for cores.
Supporting Material:	-

⌋

[RS_Arti_00011] The ARTI description for a core class shall include a “current task” reference to the interpret the parameter value

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	A core class shall include a reference to a parameter definition that defines how a specific value of the “current task” parameter should be interpreted.
Rationale:	An ARTI consuming tool needs to know how to interpret the values for a “current task”, used by this implementation.
Dependencies:	[RS_Arti_00009]
Use Case:	Defining the display for “current task”.
Supporting Material:	-

⌋

[RS_Arti_00012] The ARTI description shall include instance definitions for all cores of the ECU.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameter values to a core are collected in an instance definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to evaluate the parameter values of a specific core used by this implementation.
Dependencies:	[RS_Arti_00009]
Use Case:	Evaluating the parameter values of a core for debugging and tracing.
Supporting Material:	-

⌋

[RS_Arti_00014] The ARTI description for a core instance shall include a “current task” reference to evaluate the parameter value

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	A core instance shall include a reference to a parameter value that defines how to evaluate value of the “current task” parameter.
Rationale:	An ARTI consuming tool needs to know how to evaluate the values for a “current task”, used by this implementation.
Dependencies:	[RS_Arti_00011], [RS_Arti_00012]
Use Case:	Evaluating the parameter value of “current task” of a specific core.

▽

△

Supporting Material:	-

⌋

[RS_Arti_00016] The ARTI description shall include an OS class definition.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameters to an OS are collected in a class definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know the layout of an OS class used by this implementation.
Dependencies:	–
Use Case:	Evaluating the form of display for cores.
Supporting Material:	-

⌋

[RS_Arti_00018] The ARTI description shall include an instance definition for the OS of the ECU.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameter values to an OS are collected in an instance definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to evaluate the parameter values of a specific OS used by this implementation.
Dependencies:	[RS_Arti_00016]
Use Case:	Evaluating the parameter values of an OS for debugging and tracing.
Supporting Material:	-

⌋

[RS_Arti_00022] The ARTI description shall include a task class definition.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameters to a task are collected in a class definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know the layout of a task class used by this implementation.
Dependencies:	–
Use Case:	Evaluating the form of display for task.
Supporting Material:	-

⌋

[RS_Arti_00023] The ARTI description shall include instance definitions for all tasks of the ECU.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameter values to a task are collected in an instance definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to evaluate the parameter values of a specific task used by this implementation.
Dependencies:	[RS_Arti_00022]
Use Case:	Evaluating the parameter values of a task for debugging and tracing.
Supporting Material:	-

⌋

[RS_Arti_00025] The ARTI description shall include instance definitions for all SWCs of the ECU.

Upstream requirements: RS_Main_01025, RS_Main_01026

⌈

Description:	Additional parameter values to an SWC are collected in an instance definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to evaluate the parameter values of a specific SWC used by this implementation.
Dependencies:	-
Use Case:	Evaluating the parameter values of an SWC for debugging and tracing.

▽

△

Supporting Material:	-

⌋

4.5 Functional Requirements regarding locating

The requirements in this section are related to how code is located in the ECU memory.

[RS_Arti_00028] Grouping of Traceables

Upstream requirements: RS_Main_01026

⌈

Description:	The locating process shall allow grouping of traceable functions to be located in the same memory region.
Rationale:	Many processors support instruction tracing (“program flow trace”) without any software instrumentation. Tracing everything consumes high band-width and trace memory, whereas the user usually only is interested in a subset of functions. Most processors allow efficient tracing on a function level as long as the functions to trace are located in the same memory region (in which no other function must be located). Runnables and task bodies are implemented as functions, so by offering a way to group all functions of interest into memory regions of their own, AUTOSAR supports such efficient non-intrusive tracing.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋

4.6 Default Error Tracer (DET)

[RS_Arti_04090] A configurable list of error report receivers shall be provided

Upstream requirements: RS_Main_00011

⌈

Description:	The Default Error Tracer shall support a configurable list of functions for fan-out of received error reports. This list can be empty.
Rationale:	This implements the debugging concept in R4.0 (DocumentId 298).
Use Case:	Even development errors shall be captured by the Log and Trace functionality. Error Handling shall be enabled to react on development errors
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04086] Report errors shall contain a dedicated set of information

Upstream requirements: RS_Main_00011

⌈

Description:	Error reports, which the Default Error Tracer receives, shall consist of the ID of the reporting module, the ID of reporting instance, the ID of the API service in which the error has been detected and the error ID itself.
Rationale:	For optimal support of the error tracing some tracing information is necessary.
Use Case:	During software development phase a BSW module has been called using wrong parameters. Due to communication of some tracing information the location of the error source will be supported.
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04087] The Default Error Tracer shall provide a development error report reception service

Upstream requirements: RS_Main_00011

⌈

Description:	The Default Error Tracer shall be accessible by applications to report development error.
Rationale:	It shall be possible to perform error tracing during development of applications.
Use Case:	During software development phase a applictaion has received an unexpected response by a BSW module. By generating a development error and reporting it to the DET, configuration errors can be detected.
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04089] The DET module shall support fan-out of received error reports

Upstream requirements: RS_Main_00011

⌈

Description:	The Default Error Tracer shall forward each received error report by calling each element of a configurable list of functions.
Rationale:	This implements the debugging concept in R4.0 (DocumentId 298)
Use Case:	Even development errors shall be captured by the Log and Trace functionality. Error Handling shall be enabled to react on development errors
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04085] The Default Error Tracer shall provide an interface to receive error reports

Upstream requirements: RS_Main_00011, RS_Main_00100

⌈

Description:	The Default Error Tracer shall provide an interface to get a development error report.
Rationale:	An interface will be needed to enable handling of development errors

▽

△

Use Case:	During software development phase a BSW module has been called using wrong parameters. By generating a development error and reporting it to the DET, configuration errors can be detected.
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04101] The DET module shall forward its trace events to the DLT

Upstream requirements: RS_Main_00011

⌈

Description:	The DET receives trace events from errors from the BSW and application during debugging time. If a DLT module exists, these events should be forwarded to the DLT to collect logs and traces only in one instance.
Rationale:	To have an overview of all log, trace and error messages and to set all of them in the correct context, it is important to have all these messages and events in one list (context). Also it is not practicable to use more than one mechanism to report errors, logs and traces to a debugging interface. So all these sources should be routed to the DLT.
Use Case:	• A debugging scenario, an application or BSW Module uses the DET interface to trace an error • This error is forwarded by the DET to the DLT • The DLT turns these events in the DLT format and sends it over the debugging interface, together with all the other logs and traces
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04143] The Default Error Tracer shall provide an interface to receive runtime error reports

Upstream requirements: RS_Main_00011, RS_Main_00100

⌈

Description:	The Default Error Tracer shall provide an interface to get a runtime error report, issued by BSW modules. The Default Error Tracer returns to the caller in order to allow continuation of intended program flow.

▽

△

Rationale:	An interface will be needed to enable handling of runtime errors, caused by seldom occurring systematic faults. The caller will handle the error and continue appropriate in a deterministic manner.
Use Case:	CANNM_E_NET_START_IND: Reception of NM PDUs in Bus-Sleep Mode
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

[RS_Arti_04145] The Default Error Tracer shall forward received runtime error reports to configured integrator code

Upstream requirements: RS_Main_00011, RS_Main_00100

⌈

Description:	The Default Error Tracer shall propagate all received runtime error reports using configurable callout. The received callout return value shall be returned to the reporter of the runtime error. If no callout has been configured, a default return value shall be provided. The Default Error Tracer returns to the caller in order to allow continuation of intended program flow.
Rationale:	Integrator shall be able to recognize runtime errors and to handle in an appropriate manner.
Use Case:	CANNM_E_NET_START_IND: Reception of NM PDUs in Bus-Sleep Mode
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋

5 Requirements Tracing

The following table references the features specified in [4] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_Main_00011]	Mechanisms for Reliable Systems	[RS_Arti_04085] [RS_Arti_04086] [RS_Arti_04087] [RS_Arti_04089] [RS_Arti_04090] [RS_Arti_04101] [RS_Arti_04143] [RS_Arti_04145]
[RS_Main_00100]	AUTOSAR shall provide standardized Basic Software	[RS_Arti_04085] [RS_Arti_04143] [RS_Arti_04145]
[RS_Main_00340]	AUTOSAR shall support the continuous timing requirement analysis	[RS_Arti_00041]
[RS_Main_01025]	AUTOSAR shall support debugging of software on the target and onboard	[RS_Arti_00001] [RS_Arti_00002] [RS_Arti_00003] [RS_Arti_00004] [RS_Arti_00005] [RS_Arti_00007] [RS_Arti_00008] [RS_Arti_00009] [RS_Arti_00011] [RS_Arti_00012] [RS_Arti_00014] [RS_Arti_00016] [RS_Arti_00018] [RS_Arti_00022] [RS_Arti_00023] [RS_Arti_00025] [RS_Arti_00037]
[RS_Main_01026]	AUTOSAR shall support tracing and profiling on the target and onboard	[RS_Arti_00001] [RS_Arti_00002] [RS_Arti_00003] [RS_Arti_00004] [RS_Arti_00005] [RS_Arti_00007] [RS_Arti_00008] [RS_Arti_00009] [RS_Arti_00011] [RS_Arti_00012] [RS_Arti_00014] [RS_Arti_00016] [RS_Arti_00018] [RS_Arti_00022] [RS_Arti_00023] [RS_Arti_00025] [RS_Arti_00028] [RS_Arti_00029] [RS_Arti_00030] [RS_Arti_00031] [RS_Arti_00032] [RS_Arti_00033] [RS_Arti_00034] [RS_Arti_00035] [RS_Arti_00036] [RS_Arti_00037] [RS_Arti_00038] [RS_Arti_00039] [RS_Arti_00040] [RS_Arti_00041] [RS_Arti_00042]

Table 5.1: Requirements Tracing

6 References

[1] Requirements on Debugging, Tracing and Profiling support of AUTOSAR Components
AUTOSAR_FO_RS_DebugTraceProfile

[2] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] Main Requirements
AUTOSAR_FO_RS_Main

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R22-11

A.1.1 Added Requirements in R22-11

Number	Heading
[RS_Arti_00039]	AUTOSAR shall support recording timing events of runnable entities.
[RS_Arti_00040]	AUTOSAR shall support recording timing events of schedulable entities.
[RS_Arti_00041]	AUTOSAR shall support recording events from the standardized VFB tracing interface.
[RS_Arti_00042]	AUTOSAR shall support tracing of entries and exits of BSW modules.

Table A.1: Added Requirements in R22-11

A.1.2 Changed Requirements in R22-11

none

A.1.3 Deleted Requirements in R22-11

none

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 R24-11

A.3.1 Added Requirements in R24-11

none

A.3.2 Changed Requirements in R24-11

Number	Heading
[RS_Arti_00028]	Grouping of Traceables

Table A.2: Changed Requirements in R24-11

A.3.3 Deleted Requirements in R24-11

Number	Heading
[RS_Arti_04144]	The Default Error Tracer shall provide an interface to receive transient fault reports
[RS_Arti_04146]	The Default Error Tracer shall forward received transient fault reports to configured integrator code

Table A.3: Deleted Requirements in R24-11