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

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-10-31	4.4.0	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Scope of Document

This document refines the requirements specified in RS_ARTI_915. See chapter Scope of RS_ARTI_915. It focuses on special requirements for the Classic Platform of AUTOSAR.

2 Conventions to be used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template [1], chapter Support for Traceability.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template [1], chapter Support for Traceability.

2.1 Requirements Guidelines

Not applicable yet.

2.1.1 Requirements quality

2.1.2 Requirements identification

2.1.3 Requirements status

3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to RS_ARTI that are not included in the AUTOSAR Glossary [2].

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 RS_ARTI_915. See chapter Functional Overview of RS_ARTI_915. It focuses on special requirements for the Classic Platform of AUTOSAR.

4.2 Functional Requirements on ARTI Template

The requirements in this section all concern how the ARTI template shall be defined. This chapter refines the requirements of RS_ARTI_915 specifically for Classic Platform.

[RS_ARTICP_00001] Support for core specific ARTI additions ⌈

Type:	draft
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 EcucCoreDefinition.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific core related information to display and trace the core activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00002] Support for core specific current application parameter ⌈

Type:	draft
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:	Support for core specific ARTI additions
Use Case:	Debuggers and Tracing tools need to know the current application to display and trace the core activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00003] Support for core specific current task parameter ⌈

Type:	draft
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:	Support for core specific ARTI additions
Use Case:	Debuggers and Tracing tools need to know the current task to display and trace the core activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00004] Support for core specific last error parameter ⌈

Type:	draft
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:	Support for core specific ARTI additions
Use Case:	Debuggers and Tracing tools need to know the last error to display and trace the core activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00005] Support for OS specific ARTI additions ⌈

Type:	draft
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 EcucDefs/Os/OsOS.

▽

△

Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific OS related information to display and trace the OS activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00006] Support for OS specific application mode parameter ⌈

Type:	draft
Description:	The ARTI template shall define a parameter that contains the evaluation for the "application mode" of this OS. It shall include a reference to EcucDefs/Os/OsAppMode.
Rationale:	ARTI needs to know which application mode the OS is running at a specific time.
Dependencies:	Support for OS specific ARTI additions
Use Case:	Debuggers and Tracing tools need to know the application mode to display and trace the OS activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00007] Support for task specific ARTI additions ⌈

Type:	draft
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 EcucDefs/Os/OsTask.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific task related information to display and trace the task activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00008] Support for SWC specific ARTI additions ⌈

Type:	draft
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 EcucDefs/Rte/RteSwComponentInstance.
Dependencies:	–
Use Case:	Debuggers and Tracing tools need specific SWC related information to display and trace the SWC activity.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

4.3 Functional Requirements on ARTI Description

The requirements in this section all concern how the ARTI description shall be defined. This chapter refines the requirements of RS_ARTI_915 specifically for Classic Platform.

[RS_ARTICP_00009] The ARTI description shall include a core class definition. ⌈

Type:	draft
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_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00010] The ARTI description for a core class shall include a “current application” reference to the interpret the parameter value ⌈

Type:	draft
Description:	A core class shall include a reference to a parameter definition that defines how a specific value of “current application” parameter should be interpreted.

▽

△

Rationale:	An ARTI consuming tool needs to know how to interpret the values for a “current application”, used by this implementation.
Dependencies:	The ARTI description shall include a core class definition
Use Case:	Defining the display for “current application”.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00011] The ARTI description for a core class shall include a “current task” reference to the interpret the parameter value ⌈

Type:	draft
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:	The ARTI description shall include a core class definition
Use Case:	Defining the display for “current task”.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00012] The ARTI description shall include instance definitions for all cores of the ECU. ⌈

Type:	draft
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:	The ARTI description shall include a core class definition
Use Case:	Evaluating the parameter values of a core for debugging and tracing.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00013] The ARTI description for a core instance shall include a “current application” reference to evaluate the parameter value ⌈

Type:	draft
Description:	A core instance shall include a reference to a parameter value that defines how to evaluate value of the “current application” parameter.
Rationale:	An ARTI consuming tool needs to know how to evaluate the values for a “current application”, used by this implementation.
Dependencies:	The ARTI description shall include a core class definition
Use Case:	Evaluating the parameter value of “current application” of a specific core.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00014] The ARTI description for a core instance shall include a “current task” reference to evaluate the parameter value ⌈

Type:	draft
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:	The ARTI description shall include a core class definition
Use Case:	Evaluating the parameter value of “current task” of a specific core.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00015] The ARTI description for a core instance shall include a reference to an Ecuc core definition ⌈

Type:	draft
Description:	A core instance shall include a reference to the definition of the core in Ecuc.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the core in EcuC.
Dependencies:	The ARTI description shall include a core class definition
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00016] The ARTI description shall include an OS class definition. ⌈

Type:	draft
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_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00017] The ARTI description for an OS class shall include an application mode reference to the interpret the parameter value ⌈

Type:	draft
Description:	An OS class shall include a reference to a parameter definition that defines how a specific value of the application mode parameter should be interpreted.
Rationale:	An ARTI consuming tool needs to know how to interpret the values for the application mode, used by this implementation.
Dependencies:	The ARTI description shall include an OS class definition
Use Case:	Defining the display for “application mode”.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00018] The ARTI description shall include an instance definitions for the OS of the ECU. ⌈

Type:	draft
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:	The ARTI description shall include an OS class definition
Use Case:	Evaluating the parameter values of an OS for debugging and tracing.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00019] The ARTI description for an OS instance shall include an application mode reference to evaluate the parameter value ⌈

Type:	draft
Description:	An OS instance shall include a reference to a parameter value that defines how to evaluate value of the application mode parameter.
Rationale:	An ARTI consuming tool needs to know how to evaluate the values for the application mode, used by this implementation.
Dependencies:	The ARTI description shall include an OS class definition
Use Case:	Evaluating the parameter value of the application mode of a specific OS.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00020] The ARTI description for an OS instance shall include a reference to an Ecuc AppMode definition ⌈

Type:	draft
Description:	An OS instance shall include a reference to the definition of the AppMode in Ecuc.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the AppMode in EcuC.
Dependencies:	The ARTI description shall include an OS class definition
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00021] The ARTI description for an OS instance shall include a reference to an Ecuc OS definition ⌈

Type:	draft
Description:	An OS instance shall include a reference to the definition of the OS in Ecuc.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the OS in EcuC.
Dependencies:	The ARTI description shall include an OS class definition
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00022] The ARTI description shall include a task class definition. ⌈

Type:	draft
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_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00023] The ARTI description shall include instance definitions for all tasks of the ECU. ⌈

Type:	draft
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:	The ARTI description shall include a task class definition
Use Case:	Evaluating the parameter values of a task for debugging and tracing.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00024] The ARTI description for a task instance shall include a reference to an Ecuc task definition ⌈

Type:	draft
Description:	A task instance shall include a reference to the definition of the task in Ecuc.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the task in EcuC.
Dependencies:	The ARTI description shall include a task class definition
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00025] The ARTI description shall include instance definitions for all SWCs of the ECU. ⌈

Type:	draft
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:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00026] The ARTI description for an SWC instance shall include a reference to an Ecuc core definition ⌈

Type:	draft
Description:	An SWC instance shall include a reference to the definition of the core in Ecuc that runs this SWC.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the core in EcuC.
Dependencies:	-
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTICP_00027] The ARTI description for an SWC instance shall include a reference to an Ecuc SwcInstance definition ⌈

Type:	draft
Description:	An SWC instance shall include a reference to the definition of the SwcInstance in Ecuc.
Rationale:	An ARTI consuming tool may need to evaluate the configuration of the SwcInstance in EcuC.
Dependencies:	-
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

4.4 Functional Requirements regarding locating

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

[RS_ARTICP_00028] The locating process shall allow grouping of “traceables” into separate memory regions. ⌈

Type:	draft
Description:	Many processors support instruction tracing (“flow-tracing”) which consumes high band-width and also special versions of the processor (cf. Infineon AURIX emulation device). When looking at e.g. the level of tasks or runnables, instruction traces provide far more details than necessary. So for tracing tasks, runnables, etc. instruction tracing is not the right method. More and more processors allow efficient tracing (without any software instrumentation) 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 typically also tasks are implemented as functions so when AUTOSAR offered a way to group all tasks and runnables into memory regions of their own, such efficient non-intrusive tracing would be supported.
Rationale:	Processors which support non-intrusive high-level tracing need objects to trace (“traceables”) such as tasks, runnables, etc. to be located “together” in a memory-region.
Dependencies:	–
Use Case:	Highly efficient tracing of tasks, runnables, functions belonging to dedicated SW-Cs etc.
Supporting Material:	-

⌋*(RS_Main_01026)*

4.5 Default Error Tracer (DET)

[SRS_ARTICP_04090] A configurable list of error report receivers shall be provided ⌈

Type:	draft
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_Main_00011)*

[SRS_ARTICP_04086] Report errors shall contain a dedicated set of information ⌈

Type:	draft
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_Main_00011*)

[SRS_ARTICP_04087] The Default Error Tracer shall provide a development error report reception service ⌈

Type:	draft
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_Main_00011*)

[SRS_ARTICP_04089] The DET module shall support fan-out of received error reports ⌈

Type:	draft
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_Main_00011*)

[SRS_ARTICP_04085] The Default Error Tracer shall provide an interface to receive error reports ⌈

Type:	draft
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_Main_00011*, *RS_Main_00100*)

[SRS_ARTICP_04101] The DET module shall forward its trace events to the DLT ⌈

Type:	draft
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_Main_00011*)

[SRS_ARTICP_04143] The Default Error Tracer shall provide an interface to receive runtime error reports ⌈

Type:	draft
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_Main_00011, RS_Main_00100)*

[SRS_ARTICP_04144] The Default Error Tracer shall provide an interface to receive transient fault reports ⌈

Type:	draft
Description:	The Default Error Tracer shall provide an interface to get a transient fault 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 transient faults, caused by seldom occurring transient hardware faults.
Use Case:	• CAN controller goes offline due to bit-flip in its control register • Peripheral action lasts accidentally longer than expected (and specified)
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋*(RS_Main_00011, RS_Main_00100)*

[SRS_ARTICP_04145] The Default Error Tracer shall forward received runtime error reports to configured integrator code ⌈

Type:	draft

▽

△

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:	–

⌋(*RS_Main_00011*, *RS_Main_00100*)

[SRS_ARTICP_04146] The Default Error Tracer shall forward received transient fault reports to configured integrator code ⌈

Type:	draft
Description:	The Default Error Tracer shall propagate all received transient fault report using configurable callout. The received callout return value shall be returned to the reporter of the transient fault. 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 transient faults and to handle in an appropriate manner and to advice the reporter.
Use Case:	• CAN controller goes offline due to bit-flip in its control register. Integrator decides that reporting CAN driver shall re-initialize the CAN controller. • CAN controller goes offline due to bit-flip in its control register. Integrator decides that reporting CAN driver shall treat offline state of CAN controller as intended.
AppliesTo:	CP
Dependencies:	–
Supporting Material:	–

⌋(*RS_Main_00011*, *RS_Main_00100*)

5 Requirements Tracing

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

Feature	Description	Satisfied by

[RS_Main_00011]	AUTOSAR shall support the development of reliable systems	[SRS_ARTICP_04085] [SRS_ARTICP_04086] [SRS_ARTICP_04087] [SRS_ARTICP_04089] [SRS_ARTICP_04090] [SRS_ARTICP_04101] [SRS_ARTICP_04143] [SRS_ARTICP_04144] [SRS_ARTICP_04145] [SRS_ARTICP_04146]
[RS_Main_00100]	AUTOSAR shall provide standardized Basic Software	[SRS_ARTICP_04085] [SRS_ARTICP_04143] [SRS_ARTICP_04144] [SRS_ARTICP_04145] [SRS_ARTICP_04146]
[RS_Main_01025]	AUTOSAR shall support debugging of software on the target and onboard	[RS_ARTICP_00001] [RS_ARTICP_00002] [RS_ARTICP_00003] [RS_ARTICP_00004] [RS_ARTICP_00005] [RS_ARTICP_00006] [RS_ARTICP_00007] [RS_ARTICP_00008] [RS_ARTICP_00009] [RS_ARTICP_00010] [RS_ARTICP_00011] [RS_ARTICP_00012] [RS_ARTICP_00013] [RS_ARTICP_00014] [RS_ARTICP_00015] [RS_ARTICP_00016] [RS_ARTICP_00017] [RS_ARTICP_00018] [RS_ARTICP_00019] [RS_ARTICP_00020] [RS_ARTICP_00021] [RS_ARTICP_00022] [RS_ARTICP_00023] [RS_ARTICP_00024] [RS_ARTICP_00025] [RS_ARTICP_00026] [RS_ARTICP_00027]

[RS_Main_01026]	AUTOSAR shall support tracing and profiling on the target and onboard	[RS_ARTICP_00001] [RS_ARTICP_00002] [RS_ARTICP_00003] [RS_ARTICP_00004] [RS_ARTICP_00005] [RS_ARTICP_00006] [RS_ARTICP_00007] [RS_ARTICP_00008] [RS_ARTICP_00009] [RS_ARTICP_00010] [RS_ARTICP_00011] [RS_ARTICP_00012] [RS_ARTICP_00013] [RS_ARTICP_00014] [RS_ARTICP_00015] [RS_ARTICP_00016] [RS_ARTICP_00017] [RS_ARTICP_00018] [RS_ARTICP_00019] [RS_ARTICP_00020] [RS_ARTICP_00021] [RS_ARTICP_00022] [RS_ARTICP_00023] [RS_ARTICP_00024] [RS_ARTICP_00025] [RS_ARTICP_00026] [RS_ARTICP_00027] [RS_ARTICP_00028]

6 References

[1] System Template
AUTOSAR_TPS_SystemTemplate

[2] Glossary
AUTOSAR_TR_Glossary

[3] Main Requirements
AUTOSAR_RS_Main