Document Title	Requirements on Debugging, Tracing and Profiling support of AUTOSAR Components
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	915

Document Status	Final
Part of AUTOSAR Standard	Foundation
Part of Standard Release	1.5.0

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-10-31	1.5.0	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Scope of Document

Debugging and tracing enables efficient development, integration, optimization and verification of ECU software. For analyzing several aspects - especially timing aspects - it becomes essential to link the debugging and tracing data to the scheduling of an ECU. Knowledge about tasks, interrupts and runnables, in other words: awareness of the operating system ("OS awareness"), is required.

A good interaction of the tool chain provides complete round-trip engineering from model down to hardware and back - covering several software levels and several phases of the V-model.

The objective of ARTI ("AUTOSAR Run-Time Interface") is an interface description, that allows debugging and tracing of AUTOSAR systems with a flexible awareness supporting different views, such as OS, BSW, RTE, model, user-data and user-defined.

In particular, the interface shall enable tools to perform:

- Debugging
- Tracing
- Timing Measurement
- Profiling

Whereas

- "Debugging" refers to halting a system, either as a whole or in parts, for the purpose of
  - inspecting the contents of the system in a frozen state
  - single stepping, setting breakpoints, starting and stopping in C or Assembly code
- "Tracing" refers to collecting run-time information over a certain period of time
  - either as a pure software solution, or with hardware assistance
  - may include processor instruction trace, OS scheduling trace, and/or pure data trace
  - including time-stamping for further timing analysis
- "Timing Measurement" refers to capturing of timing information
  - by instrumentation, e.g. via Pre-/PostTaskHooks or other hooks or callouts or
  - by dedicated hardware support, e.g. hardware performance counters
  - does not stop execution
- "Profiling" refers to the process of gaining timing parameters/timing statistics

- of functions, tasks, runnables, modules etc.
- possibly with minimum/maximum/average statistics
- possibly with worst case analysis
- possibly calculated out of trace data, repeated snapshots or Timing Measurement

The main focus of this requirements document is the format of the ARTI hooks and the exported ARTI description.

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

Table 3.1: Acronyms and Abbreviations

4 Requirements Specification

This chapter describes all requirements driving the work to define ARTI.

4.1 Functional Overview

The tools that generate AUTOSAR modules (e.g. OS, RTE, etc.) have to emit internal information about this module as a separate file (“ARTI file”). The information therein shall allow to debug and trace the behavior of this module. Additional tools will collect all ARTI files of an ECU and allow selecting specific items to trace and create tracing hook files for a specific trace channel (e.g. internal buffer, hardware trace buffers, etc.). The build environment creates the final application, which then can be used in the ECU. Debugging and tracing tools can read in the ARTI files and are “AUTOSAR” aware, giving additional debugging and tracing features to the developer.

ARTI is supposed to work with debug information created by the compilers. This means each module that supports ARTI needs to be compiled with debug information, and the ARTI file has to use the symbol names created by the compiler.

ARTI introduces new hooks. In order to use them, they shall be incorporated into the module’s C code. Either they are put therein statically, or they have to be configured.

ARTI consist of these functional elements:

- ARTI module description
  The “ARTI Module Description” is emitted as an ARXML file, standardized by an XML document defined by this specification (MOD_ARTI_920).
- ARTI hook implementations
  The ARTI hooks are implemented within the AUTOSAR module’s source code.

4.2 Functional Requirements on ARTI Template

The requirements in this section all concern how the ARTI template shall be defined.

[RS_ARTIFO_00001] The ARTI Description shall be the root for the whole ARTI information of an ECU ⌈

Type:	draft
Description:	The ARTI Template shall be the backbone for all information regarding ARTI of a particular ECU.
Rationale:	To access Run-Time information of an ECU, a tool needs to have one reference to look for the information. All further detailed information can be found by following the contents of this reference. Note that this requirement does not imply that all relevant information is copied into the template. References to elements in other templates may be included to avoid redundant elements in the model.
Dependencies:	ARTI tools need to be able to read other templates (esp. ECU), too, if these are referenced by the ARTI description.
Use Case:	The ARTI Template includes specifications on how to model generic classes and instances, without changing the template, allowing to specify the access to any arbitrary object of the ECU. The template includes references to other templates and components to be able to use this information, too.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00002] Support generic objects ⌈

Type:	draft
Description:	Generic objects define arbitrary objects.
Rationale:	A generic object is any arbitrary object that is not part of another module. The definition shall be able to define the layout of an object (“class”) as well as the values of a specific instance.
Dependencies:	-
Use Case:	The generic object definition defines how an arbitrary object should show up in a debugger or when tracing.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00003] Support class definition of a generic object ⌈

Type:	draft
Description:	The class definition of a generic object defines the layout of an arbitrary object.
Rationale:	An object that is not part of another AUTOSAR module has to be defined to an ARTI consuming tool. The ARTI Template shall provide a mechanism to define the object layout (“class”) with its name, description, and parameters.
Dependencies:	-
Use Case:	The generic object definition defines how an arbitrary object should show up in a debugger or when tracing.

▽

△

Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00004] Support instance definition of a generic object ⌈

Type:	draft
Description:	The instance definition of a generic object defines the values of an arbitrary object instance.
Rationale:	The ARTI Template shall define, how an ARTI consuming tool can evaluate the parameters of an object instance.
Dependencies:	-
Use Case:	-
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00005] Support expressions to evaluate a parameter value ⌈

Type:	draft
Description:	Expressions define how a specific value of a parameter can be accessed on the target.
Rationale:	Expressions are typically represented as C expression.
Dependencies:	-
Use Case:	Expressions and values are used by a debugger to display a current state of the application.
Supporting Material:	-

⌋(*RS_Main_01025*)

[RS_ARTIFO_00006] Support constants used by object parameters ⌈

Type:	draft
Description:	Constants define the value of an object parameter.
Rationale:	Object parameters may be constant. This definition shall supply the container for the constant.
Dependencies:	-
Use Case:	Object parameters may refer to expressions or constants. This is the definition for constant parameters.

▽

△

Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00007] Support parameter maps ⌈

Type:	draft
Description:	Parameter maps translate the value of a parameter to a string to display.
Rationale:	A parameter value may refer to a specific meaning (e.g. “state”). The parameter map allows to translate the value to a (human readable) string.
Dependencies:	-
Use Case:	Parameter maps are used by an ARTI consuming tool to display a parameter value in a meaningful way.
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.

[RS_ARTIFO_00008] The ARTI description shall follow the ARTI Template. ⌈

Type:	draft
Description:	All information concerning ARTI shall be collected in the ARTI description, following the ARTI Template.
Rationale:	To access Run-Time information of an ECU, a tool needs to know how to read the information. By strictly following the template, the tool is able to extract all necessary information without further user interaction.
Dependencies:	ARTI tools need to be able to read and understand the ARTI Template and possibly other templates (e.g. ECU).
Use Case:	Using the ARTI description, an ARTI tool is able to detect, visualize and measure arbitrary objects of an ECU.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00009] The ARTI description shall include all class definitions of generic objects. ⌈

Type:	draft
Description:	All generic (arbitrary) objects that should be processed by an ARTI consuming tool shall be defined in a class definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know the layout of all generic objects used by the module.
Dependencies:	-
Use Case:	Evaluating the form of display for generic objects.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00010] The ARTI description shall include all instance definitions of generic objects. ⌈

Type:	draft
Description:	All generic (arbitrary) instances of objects that should be processed by an ARTI consuming tool shall be defined in an instance definition, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to get the values of parameters of generic objects.
Dependencies:	-
Use Case:	Evaluating the values to display for generic objects.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00011] The ARTI description shall include all expression definitions to evaluate parameter values. ⌈

Type:	draft
Description:	All expressions used to evaluate parameter values shall be defined, following the ARTI Template.
Rationale:	An ARTI consuming tool needs to know how to read the values of parameters from the target. Expressions are typically represented as C expression.
Dependencies:	-
Use Case:	Evaluating the values to display for object parameters.
Supporting Material:	-

⌋(*RS_Main_01025*)

[RS_ARTIFO_00012] The ARTI description shall include all constant definitions used by parameters. ⌈

Type:	draft
Description:	All constants used by parameters shall be defined, following the ARTI Template.
Rationale:	Object parameters may be constant. An ARTI consuming tool needs to know which parameter is constant and which value to show.
Dependencies:	-
Use Case:	Evaluating the values to display for object parameters.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

[RS_ARTIFO_00013] The ARTI description shall include all parameter maps used by parameters. ⌈

Type:	draft
Description:	All parameter maps used by parameters shall be defined, following the ARTI Template.
Rationale:	A parameter value may refer to a specific meaning (e.g. “state”). The parameter map translates the value of a specific parameter to a (human readable) string.
Dependencies:	-
Use Case:	Parameter maps are used by an ARTI consuming tool to display a parameter value in a meaningful way.
Supporting Material:	-

⌋(*RS_Main_01025*, *RS_Main_01026*)

4.4 Functional Requirements on ARTI Hooks

The requirements in this section all concern how ARTI hooks shall be used and defined.

[RS_ARTIFO_00014] ARTI Hooks shall be implemented as macros ⌈

Type:	draft
Description:	ARTI Hooks shall be implemented in source code as macros.
Rationale:	ARTI Hooks will be extended by an ARTI consuming tool. Macros are used for full flexibility within the source code.
Dependencies:	–

▽

△

Use Case:	Hooks may be implemented as “C” macros. This allows easily to expand the macros to “nothing”, causing no intrusion or side effects at all. An ARTI consuming tool may choose to extend the macro to whatever it works best for this tool.
Supporting Material:	-

⌋(*RS_Main_01026*)

[RS_ARTIFO_00015] ARTI Hooks shall follow a fixed format ⌈

Type:	draft
Description:	ARTI Hooks shall follow a fixed format.
Rationale:	To allow an ARTI consuming tool to expand the ARTI Hooks it is beneficially if these follow a common format.
Dependencies:	RS_ARTIFO_00015
Use Case:	Hooks may be implemented as “C” macros. This allows easily to expand the macros to “nothing”, causing no intrusion or side effects at all. An ARTI consuming tool may choose to extend the macro to whatever it works best for this tool.
Supporting Material:	-

⌋(*RS_Main_01026*)

[RS_ARTIFO_00016] ARTI Hooks shall provide information about the calling context ⌈

Type:	draft
Description:	ARTI Hooks shall provide information about the calling context.
Rationale:	To ensure that the ARTI Hook implementation creates as little over head as possible ARTI consuming tool needs to know the calling context.
Dependencies:	RS_ARTIFO_00016
Use Case:	An ARTI consuming tool may choose to extend the macro to whatever it works best for this tool and in the current context. This mainly concerns the privilege level and whether interrupts are looked or not.
Supporting Material:	-

⌋(*RS_Main_01026*)

[RS_ARTIFO_00017] ARTI Hooks shall provide the name of the class they belong to ⌈

Type:	draft
Description:	ARTI Hooks shall provide the name of the class they belong to.
Rationale:	Hooks shall be grouped into classes belonging to a similar functionality (events)
Dependencies:	RS_ARTIFO_00016
Use Case:	These classes can be: AR_CP_OS_TASKSCHEDULER, AR_CP_OS_CAT2DISPATCHER, ...
Supporting Material:	-

⌋(*RS_Main_01026*)

[RS_ARTIFO_00018] ARTI Hooks shall provide the name of the event ⌈

Type:	draft
Description:	ARTI Hooks shall provide the name of the event. These events should be grouped into classes.
Rationale:	
Dependencies:	RS_ARTIFO_00016
Use Case:	Event names can be: OsTask_Activation, OsTask_Start, OsTask_Stop, ...
Supporting Material:	-

⌋(*RS_Main_01026*)

[RS_ARTIFO_00019] ARTI Hooks shall provide a parameter for the event ⌈

Type:	draft
Description:	ARTI Hooks shall provide a parameter for the event.
Rationale:	
Dependencies:	RS_ARTIFO_00018
Use Case:	e.g. Task Index for OsTask_Activation, OsTask_Start and OsTask_Stop
Supporting Material:	-

⌋(*RS_Main_01026*)

4.5 Non-Functional Requirements (Qualities)

5 Requirements Tracing

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

Feature	Description	Satisfied by

[RS_Main_01025]	AUTOSAR shall support debugging of software on the target and onboard	[RS_ARTIFO_00001] [RS_ARTIFO_00002] [RS_ARTIFO_00003] [RS_ARTIFO_00004] [RS_ARTIFO_00005] [RS_ARTIFO_00006] [RS_ARTIFO_00007] [RS_ARTIFO_00008] [RS_ARTIFO_00009] [RS_ARTIFO_00010] [RS_ARTIFO_00011] [RS_ARTIFO_00012] [RS_ARTIFO_00013]
[RS_Main_01026]	AUTOSAR shall support tracing and profiling on the target and onboard	[RS_ARTIFO_00001] [RS_ARTIFO_00002] [RS_ARTIFO_00003] [RS_ARTIFO_00004] [RS_ARTIFO_00006] [RS_ARTIFO_00007] [RS_ARTIFO_00008] [RS_ARTIFO_00009] [RS_ARTIFO_00010] [RS_ARTIFO_00012] [RS_ARTIFO_00013] [RS_ARTIFO_00014] [RS_ARTIFO_00015] [RS_ARTIFO_00016] [RS_ARTIFO_00017] [RS_ARTIFO_00018] [RS_ARTIFO_00019]

6 References

[1] System Template
AUTOSAR_TPS_SystemTemplate

[2] Glossary
AUTOSAR_TR_Glossary

[3] Main Requirements
AUTOSAR_RS_Main