Document Title	Specification of AUTOSAR Run-Time Interface
Document Category	Software specification (SWS)
Document UID	923
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Removed GetVersionInfo API • Added more descriptive information • Removed references to requirement specification • Corrected ARXML examples
2025-11-27	R25-11	AUTOSAR Release Management	• Editorial changes
2024-11-27	R24-11	AUTOSAR Release Management	• Added ARTI description for schedule tables and spinlocks • Changed handling of Arti in RTE generation
2023-11-23	R23-11	AUTOSAR Release Management	• Extended stopwatch to allow nesting • Added imported types • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Corrected Error Classification • Introduced ARTI macro with several event parameters • Introduced ARTI class for tracing RTE API and BSW API • Splitted “arti.h” in module depending header files
2021-11-25	R21-11	AUTOSAR Release Management	• Introduced SWS items into specification • Overall review and clarification • ARTI introduced as BSW Module "Arti" • New ARTI API and Errors



△

2020-11-30	R20-11	AUTOSAR Release Management	• Merged EcuC ArtiXxx containers into one Arti container • Added ARTI for RTE • Changed document status from draft to valid
2019-11-28	R19-11	AUTOSAR Release Management	• Added expression syntax • Corrected trace macros and ARTI class names • Added and extended several configuration parameters • Corrected SWS item references • Changed Document Status from Final to published
2018-10-31	R4.4.0	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 7

2 Acronyms and Abbreviations 9

3 Related documentation 11
3.1 Input documents & related standards and norms . . . 11
3.2 Related specification . . . 11

4 Constraints and assumptions 12
4.1 Limitations . . . 12

5 Dependencies to other modules 13

6 Requirements Tracing 14

7 Functional specification 15
7.1 ARTI Module Description . . . 15
7.2 ARTI Hook Implementation . . . 16
7.3 ARTI OS Implementation . . . 18
7.4 ARTI RTE VFB Trace Client . . . 18
7.4.1 RTE VFB Trace Client Configuration . . . 18
7.5 Error Classification . . . 23
7.5.1 Development Errors . . . 23
7.5.2 Runtime Errors . . . 23
7.5.3 Production Errors . . . 23

8 API specification 24
8.1 Imported types . . . 24
8.2 Type definitions . . . 24
8.3 Symbol definitions . . . 24
8.3.1 ARTI_STOPWATCH_FLAT, ARTI_STOPWATCH_NESTED . . . 24
8.4 Function definitions . . . 25
8.4.1 Arti_Init . . . 25
8.5 Scheduled functions . . . 25
8.6 Expected interfaces . . . 25
8.6.1 Common Objectives for ARTI Instrumentation . . . 26
8.6.2 Mandatory interfaces . . . 26
8.6.2.1 ARTI Tracing Macro . . . 26
8.6.2.2 ARTI Tracing Macro with Multiple Parameters . . . 30
8.6.3 Optional interfaces . . . 31
8.6.3.1 ARTI Generic Stopwatch . . . 31
8.6.3.2 ARTI Generic Dataflow Stopwatch . . . 33
8.6.3.3 ARTI Generic Datapoint . . . 35
8.6.3.4 ARTI Category 1 Interrupts . . . 36
8.6.3.5 ARTI RTE VFB Trace Client . . . 39

8.6.3.6 ARTI BSW Module Interface . . . 45
8.6.4 Callout interfaces . . . 46
8.6.5 Configurable interfaces . . . 47
8.7 Service Interfaces . . . 47
9 Sequence diagrams 48
10 Configuration specification 49
10.1 How to read this chapter . . . 49
10.2 ARTI Parameters . . . 49
10.3 ARTI Generic Container . . . 50
10.3.1 ArtiGenericComponentClass . . . 52
10.3.2 ArtiGenericComponentInstance . . . 55
10.4 ARTI Hardware Container . . . 59
10.5 ARTI Os Container . . . 60
10.6 ARTI Rte Container . . . 61
10.6.1 ArtiRteRunnableClass . . . 62
10.6.2 ArtiRteRunnableInstance . . . 63
10.6.3 ArtiRteSchedulableClass . . . 64
10.6.4 ArtiRteSchedulableInstance . . . 65
10.6.5 ArtiRteVfbTraceHooks . . . 67
10.7 ARTI Values Container . . . 68
10.7.1 ArtiConstant . . . 70
10.7.2 ArtiExpression . . . 71
10.7.3 ArtiHook . . . 72
10.7.4 ArtiObjectClassParameter . . . 76
10.7.5 ArtiObjectInstanceParameter . . . 77
10.7.6 ArtiParameterTypeMap . . . 78
10.7.7 ArtiStates . . . 83
10.8 Published Information . . . 87
A Not applicable requirements 88
B Example 89
B.1 ARTI Instrumentation . . . 89
B.1.1 ARTI Tool Binding (Arti.h) . . . 89
B.1.2 ARTI OS Instrumentation . . . 92
B.1.3 ARTI User Code . . . 93
B.2 ARXML Representation of Instrumentation . . . 94
C Expression Syntax 99
D Change history of AUTOSAR traceable items 100
D.1 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 100
D.1.1 Added Specification Items in R22-11 . . . 100
D.1.2 Changed Specification Items in R22-11 . . . 100

D.1.3 Deleted Specification Items in R22-11 . . . 100
D.2 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 101
D.2.1 Added Specification Items in R23-11 . . . 101
D.2.2 Changed Specification Items in R23-11 . . . 101
D.2.3 Deleted Specification Items in R23-11 . . . 101
D.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 101
D.3.1 Added Specification Items in R24-11 . . . 101
D.3.2 Changed Specification Items in R24-11 . . . 102
D.3.3 Deleted Specification Items in R24-11 . . . 102
D.4 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 102
D.4.1 Added Specification Items in R25-11 . . . 102
D.4.2 Changed Specification Items in R25-11 . . . 102
D.4.3 Deleted Specification Items in R25-11 . . . 102
D.5 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 102
D.5.1 Added Specification Items in R26-11 . . . 102
D.5.2 Changed Specification Items in R26-11 . . . 102
D.5.3 Deleted Specification Items in R26-11 . . . 103

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Run-Time Interface (“ARTI”) for debugging and tracing AUTOSAR modules.

ARTI defines an interface between build tools and debugging/tracing tools. The debugging/tracing tools shall then forward tracing information to trace/timing analysis tools. The interface shall ease and speed up the debugging, tracing and verification of system behavior as well as round-trip engineering.

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 `Runnable Entities`, 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.

ARTI shall especially provide

- Support of “OS Awareness”, for example examination of OS specific tasks, threads etc.
- Support of distributes systems and multi-core
- Support of other AUTOSAR modules (e.g. RTE in CP or ARA in AP)
- Support of instrumentation-based tracing and measurement solutions
- Support of TIMEX

The data flow of the tools and the interfaces of ARTI are depicted in figure 1.1.

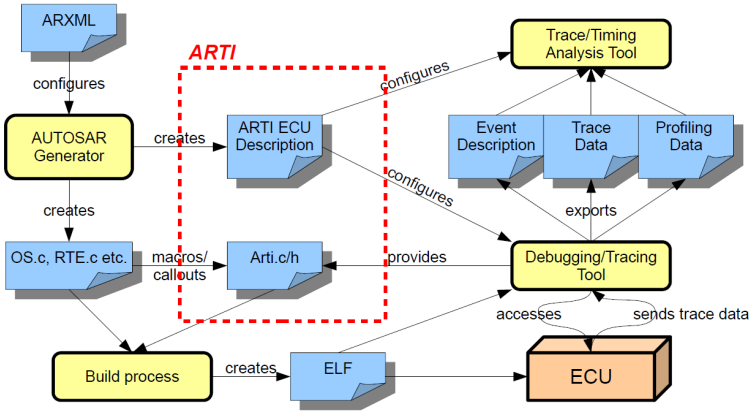

Figure 1.1: ARTI data flow

For some important definitions please read also chapter 1 of RS_FoundationDebugTraceProfile.

To implement the features, ARTI uses a similar approach that the former OSEK-ORTI had, but extends this to current requirements. The tools that generate AUTOSAR modules (e.g. OS, RTE, etc.) have to extend the ECU configuration with internal information about this module and emit the extended configuration 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. These tools can export a trace file, which in turn can be used in trace analysis tools for extended timing analysis, time measurements and optimization runs.

Using the standardized work flow allows interchanging the tools as necessary, and use the tool that fits best for each solution without the need of adapting the work flow.

The work flow of the ARTI file generation and usage is depicted in figure 1.2. ARTI shall only define interfaces within the build process of an AUTOSAR application (i.e. the export of the generators, and the hooks within the AUTOSAR modules). The interfaces for tool communication are post-build and not subject to this specification.

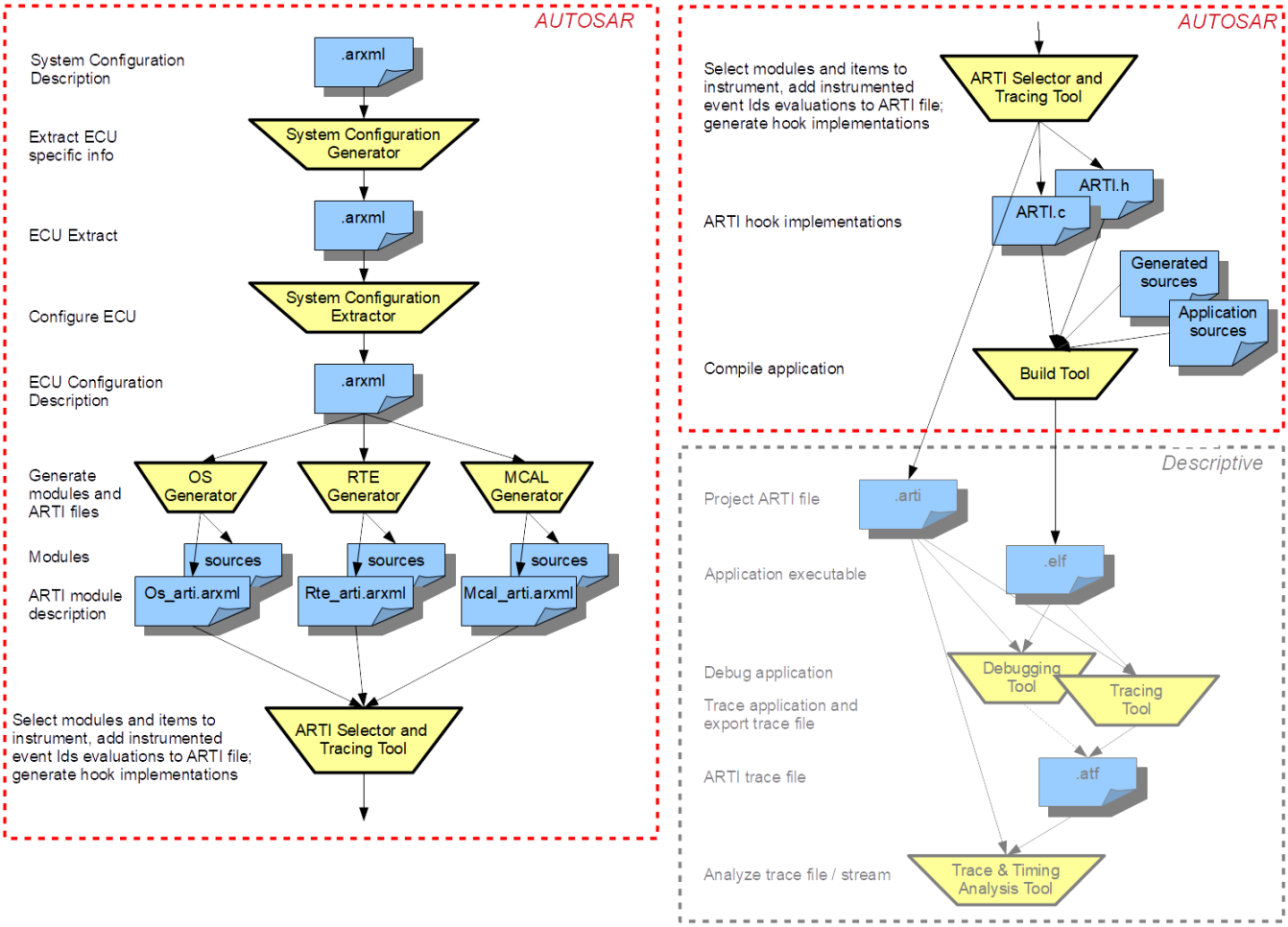

Figure 1.2: ARTI work flow

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the ARTI module that are not included in the [1, AUTOSAR glossary].

Abbreviation/Acronym	Description
ORTI	"OSEK Run Time Interface", an OSEK specification (in its version 2.2) that defines how debuggers can access OSEK OS internal information.

Table 2.1: Abbreviations and Acronyms

Term	Description
Interrupt	See AUTOSAR glossary [1].
Runnable Entity	See AUTOSAR glossary [1].
Task	See AUTOSAR glossary [1].
Debugging	"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	"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	"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	"Profiling" refers to the process of gaining timing parameters/timing statistics • of functions, tasks, Runnable Entities, modules etc. • possibly with minimum/maximum/average statistics • possibly with worst case analysis • possibly calculated out of trace data, repeated snapshots or Timing Measurement

Table 2.2: Terms

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of Operating System
AUTOSAR_CP_SWS_OS

[3] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[4] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

3.2 Related specification

Not applicable yet.

4 Constraints and assumptions

The ARTI concept expects to get an own ARTI module description from each module to be debugged, traced or profiled, e.g. OS and RTE. This allows mixing modules with ARTI support with those without ARTI support. However, as ARTI contains internal information, the implementers of the modules have to provide the ARTI file.

4.1 Limitations

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 are put therein dynamically by a generator as configured.

Tracing internal events is very time critical. ARTI focuses on the solutions with the least impact on timing (in some cases with no timing overhead at all), but this depends on the hardware capabilities of the ECU and the tools. ARTI provides examples that describe the possibilities for tracing, depending on the available hardware and software capabilities (see Appendix B).

5 Dependencies to other modules

ARTI depends on OS and RTE module, which refine the ARTI description and hooks for their purposes.

6 Requirements Tracing

The following table references the requirements specified in RS_ClassicPlatformDebugTraceProfile and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Arti_91004]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Arti_91006]
[SRS_BSW_00337]	Classification of development errors	[SWS_Arti_00009] [SWS_Arti_91002]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_Arti_00009]

Table 6.1: Requirements Tracing

7 Functional specification

ARTI consists of these functional elements:

- ARTI module description
- ARTI hook implementations

The “ARTI Module Description” is intended to be emitted as an ARXML file. Additional files, such as the “project ARTI file” or “ARTI trace file” may be stored in another file format, whereas this format is beyond AUTOSAR and defined elsewhere.

ARTI is not a traditional software module that creates code and changes the system behavior. Instead ARTI is explicitly designed to *not* affect the overall system behavior. Especially the generation and export of the ARTI module description is intended to not influence the module that generates the ARTI export; ARTI should export information that is already internally available. The exported information will then be post-processed and used by further debugging and tracing tools. However, it might be necessary to introduce some special variables or functions to be able to generate requested information. While this causes some slight impact to the code, it is again the intention not to change the overall behavior of the module using ARTI. The same applies to the hooks: while the hooks itself may have some slight impact on the code base and while the hook implementation (done by the tools consuming ARTI) may have some impact on the timing and on the program flow, it is the intention of ARTI to change the module behavior as little as possible – ideally not at all. Depending on the hook implementation, the behavior may differ. It is the responsibility of the tool vendor to minimize the impact to the behavior of the system.

ARTI shall be defined in a way that it is applicable on the road. If ARTI hooks are implemented, this obviously comes with high safety requirements regarding the implementation of the hooks since e.g. some of the ARTI hooks will be executed in the context of the OS. Special care has to be taken in a multi-core context.

If the implementation of the hooks cannot guarantee safe execution, the ECU must not be used “on the road”. “On the road” here refers to situations where the operation or malfunction might cause danger to persons or property.

7.1 ARTI Module Description

An "ARTI Module Description" is an ARXML file that contains detailed information about a specific module (e.g. OS, RTE, etc.). In particular, this is:

- Constants
  A Constant defines a constant value that is specific to this application or environment. E.g. the number of CPUs used in an ECU could be defined as a constant. Constants are used by a debugger to know about the configuration, or to display the value in a convenient way.
  Constants are referred to by an object information (see "Object Information" be-

low). and are only meaningful in the context of an object.
A Constant is represented by the container ArtiConstant (see chapter 10.7.1).

- Expressions
  An Expression defines how a specific value can be accessed on the target by a debugger to display the current state of the application. Expressions are like C expressions but limited so that they can be evaluated statically. Hence, Expressions allow only accesses to global variables, and only unary, binary and ternary operators are allowed. Especially accesses to local variables and calls to functions are not allowed. See Appendix C for a full syntax specification of Expressions.
  Expressions are referred to by an object information (see "Object Information" below) and are used to define the evaluation of parameter values therein.
  An Expression is represented by the container ArtiExpression (see chapter 10.7.2).
- Hook definitions
  Hook definitions contain information about which hooks are present in the module and how they look like. These hook definitions are used to create the hook implementation and to trace the information defined by the hook.
  A Hook definition is represented by the container ArtiHook (see chapter 10.7.3).
- Object information
  Objects within a module (e.g. an “OsTask”) get an own representation in the ARTI module description. The object information contains references to the original object as well as references to the expressions and hooks used for this object. All objects of a specific kind are collected in a container. The detailed layout of an object within a specific module is defined in the according SWS.
- Generic components
  ARTI is able to define objects that should show up in a debugger or when tracing, even if those are not standard AUTOSAR objects (e.g. user defined, or additional OS features like semaphores). See chapter 10.3.

7.2 ARTI Hook Implementation

The ARTI hook implementations are generated by a tool that consumes the ARTI description files. They are mainly represented by these files:

- Arti.h
  This file contains the `ARTI_TRACE` macro implementation. The `ARTI_TRACE` macro implementation may expand to a module dependent macro.
- <Mip>_Arti.h This file contains all module dependent macros that are used in the modules supporting ARTI to instrument certain events. It may also contain the implementation of the macro, or may refer to an implementation in Arti.c. <Mip>_Arti.h shall include Arti.h in front of any module specific macro definitions.

- Arti.c
  This file contains the implementation of the API (see chapter 8) and the actual implementation of each macro, if it is not empty or not implemented in the <Mip>_Arti.h file.

All events that are not active will be mapped to an empty macro definition. All events that are active will be expanded to the implementation of the instrumentation. The actual implementation depends on the hardware and software capabilities of the tracing tool. Thus, it depends on the used tracing tool, how the macros are implemented.

Example for the OS module:

The OS source code will include the Os_Arti.h file and use the `ARTI_TRACE` macro.

```
#include "Os_Arti.h"
ARTI_TRACE(NOSUSP, AR_CP_OS, Os, GetCoreId(), OsTask_Activate,
    GetTaskId(MyTask));
```

Listing 7.1: Example: Os_Source.c

The Os_Arti.h file (created by the tool vendor) will include Arti.h and implement the module specific macro.

```
#include "Arti.h"
extern volatile uint32_t arti_os_trace;
#define ARTI_TRACE_AR_CP_OS_TASK_OsTask_Activate(CoreId, TaskId) \
    arti_os_trace = (TaskId<<16) | (ARTI_VALID_OS_SIGNALING<<8) | \
        (ARTI_OSTASK_ACTIVATE<<8) | CoreId ;
```

Listing 7.2: Example: Os_Arti.h

The Arti.h file (created by the tool vendor) will implement the `ARTI_TRACE` macro and expand it to the module specific macro.

```
#define ARTI_TRACE(_contextName, _className, _instanceName, \
        instanceParameter, _eventName, eventParameter) \
    ARTI_TRACE ## _ ## _className ## _ ## \
        _eventName((instanceParameter), (eventParameter))
```

Listing 7.3: Example: Arti.h

The Arti.c file will implement the Arti API (`Arti_Init()`) and any further code necessary to implement the tracing.

```
volatile uint32_t arti_os_trace;
void Arti_Init (void) {
    arti_os_trace = 0;
}
```

Listing 7.4: Example: Arti.c

7.3 ARTI OS Implementation

ARTI support for OS is specified in [2] Chapter 7.16 *"ARTI Debug Information"* and [2] Chapter 7.17 *"ARTI Hook Macros"*. It is related to the application note described in [2] Chapter 12.7 *"Debug support"*.

7.4 ARTI RTE VFB Trace Client

The ARTI RTE VFB trace client is designed to adapt the VFB tracing mechanism to the ARTI trace. The VFB tracing mechanism provides hooks including parameters for tracing while ARTI trace focuses on minimal intrusive trace using the `ARTI_TRACE` macro.

The ARTI basic software module implements a trace client of the VFB tracing (see [3] Chapter 5.11 *"VFB Tracing Reference"*).

It configures the RTE to generate the hooks for the trace client. These hooks will be mapped to the `ARTI_TRACE` macro with dedicated ARTI trace classes and events.

This mapping is defined in 8.6.3.5.

Restriction: ARTI supports only a subset of the RTE VFB trace client hooks. ARTI only supports:

- `Rte_Arti_Runnable_<cts>_<reName>_Start`
- `Rte_Arti_Runnable_<cts>_<reName>_Return`
- `Rte_Arti_<api>Hook_<cts>_<ap>_Start`
- `Rte_Arti_<api>Hook_<cts>_<ap>_Return`
- `SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>_Start`
- `SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>_Return`

7.4.1 RTE VFB Trace Client Configuration

The RTE VFB trace client configuration is done in several steps where RTE generator and ARTI module are interacting. Configuration parameters are exchanged in the EcuC.

1. RTE configuration provides /AUTOSAR/EcucDefs/Rte/RteSwComponentInstance after RTE configuration
2. ARTI creates an own VFB trace client called "Arti" and provides the configuration for the trace client using its own /AUTOSAR/EcucDefs/Rte/RteGeneration/RteVfbTraceClient. Within this container all the /AUTOSAR/EcucDefs/Rte/RteGeneration/RteVfbTraceClient/RteVfbTraceFunction (see AUTOSAR_SWS_RTE RteVf-

bTraceFunction) are listed for which the ARTI module requests the hooks to be generated. Here ARTI fills out the value and thus generates a 'wishlist' of tracing a certain hook function. Examples:

- Enable trace of all Schedulable Entity hooks: Rte_Arti_SchM
- Enable trace of all Runnable Entity hooks: Rte_Arti_Runnable
- Enable trace of all Runnable Entity hooks of a certain component: Rte_Arti_Runnable_MyComponentType where MyComponentType is taken from /AUTOSAR/EcucDefs/Rte/RteSwComponentType
- Enable trace of Runnable Entity hooks of a certain Runnable Entity within a certain component: Rte_Arti_Runnable_MyComponentType_MyRunnable where MyRunnable is taken from /AUTOSAR/SoftwareTypes/ComponentTypes/<ApplicationSwComponentType>/<SwcInternalBehavior>/<RunnableEntity>

Within the RteVfbTraceClient container, add an RteVfbTraceHooksRef with an URI pointing to the `ArtiRteVfbTraceHooks` container of the ARTI trace client.

3. Based on this configuration the RTE generator creates the source files containing the trace hooks. The generated hooks are BSW-MODULE-ENTRY where the FUNCTION-PROTOTYPE-EMITTER is "Arti".
4. ARTI generator creates the final trace client based on the BSW-MODULE-ENTRY's for the ARTI trace client.
   - It generates the header file for the mapping of the VFB trace hooks to `ARTI_TRACE` macro. All unused generated hooks are mapped to (void). As part of the mapping, the Arti module needs to provide a Runnable Entity to Runnable ID mapping (see 8.6.3.5, idOf(<reName>)).
   - It updates the the Arti's BSWMD with the missing information:
     - It extends the BswInternalBehavior of the RTE with each ARTI hook as function marked with SW-ADDR-METHOD-REF CODE.
     - It extends the BSW-MODULE-ENTRY of each hook with the correct SW-SERVICE-IMPL-POLICY (MACRO, INLINE or STANDARD).
     - It adds the REQUIRED-ARTIFACTS that implement the hooks to the BSW-IMPLEMENTATION.
     - It specifies the RESOURCE-CONSUMPTION by adding ARTI MEMORY-SECTION that holds the EXECUTABLE-ENTITY-REFS of all hooks and add the SECTION-NAME-PREFIX for the required artifacts.
5. Compile RTE

Example 7.1

1. RTE provides /AUTOSAR/EcucDefs/Rte/RteSwComponentInstance

   ...

```
<ECUC-CONTAINER-VALUE UUID="cd307f8d-8496-421b-a9e8-571463b08250
    ">
   <SHORT-NAME>ConsumerComponent</SHORT-NAME>
   <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">
       /AUTOSAR/EcucDefs/Rte/RteSwComponentInstance
   </DEFINITION-REF>
   ...
</ECUC-CONTAINER-VALUE>
```

2. ARTI creates VFB trace client

```
...
<ECUC-CONTAINER-VALUE UUID="6de0bb4e-c1ff-4c6c-ae19-3a0f536e7e9e
    ">
   <SHORT-NAME>Arti</SHORT-NAME>
   <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">
       /AUTOSAR/EcucDefs/Rte/RteGeneration/RteVfbTraceClient
   </DEFINITION-REF>
   <PARAMETER-VALUES>
       <ECUC-TEXTUAL-PARAM-VALUE>
           <DEFINITION-REF DEST="ECUC-FUNCTION-NAME-DEF">
               /AUTOSAR/EcucDefs/Rte/RteGeneration/
               RteVfbTraceClient/RteVfbTraceFunction
           </DEFINITION-REF>
           <VALUE>Rte_Arti_Runnable_ConsumerComponent</VALUE>
       </ECUC-TEXTUAL-PARAM-VALUE>
       ...
   </PARAMETER-VALUES>
</ECUC-CONTAINER-VALUE>
...
```

3. Based on this configuration the RTE generator creates the source

```
...
<BSW-MODULE-ENTRY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Return
    </SHORT-NAME>
    <FUNCTION-PROTOTYPE-EMITTER>Arti</FUNCTION-PROTOTYPE-EMITTER
        >
    <CALL-TYPE>CALLBACK</CALL-TYPE>
</BSW-MODULE-ENTRY>
<BSW-MODULE-ENTRY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Start
    </SHORT-NAME>
    <FUNCTION-PROTOTYPE-EMITTER>Arti</FUNCTION-PROTOTYPE-EMITTER
        >
    <CALL-TYPE>CALLBACK</CALL-TYPE>
</BSW-MODULE-ENTRY>
...
```

4. ARTI generator updates the RTE's BSWMD with the missing information
   - extends the BswInternalBehavior of the RTE with each ARTI hook

```
...
```

```
<BSW-CALLED-ENTITY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Start
    </SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
        /AUTOSAR_MemMap/SwAddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY"
                           BASE="Rte_BSWMD_BswModuleEntrys">
        Rte_Arti_Runnable_ConsumerComponent_RE2_Start
    </IMPLEMENTED-ENTRY-REF>
</BSW-CALLED-ENTITY>
<BSW-CALLED-ENTITY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Return
    </SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY"
                           BASE="Rte_BSWMD_BswModuleEntrys">
        Rte_Arti_Runnable_ConsumerComponent_RE2_Return
    </IMPLEMENTED-ENTRY-REF>
</BSW-CALLED-ENTITY>
...
```

- extends the BSW-MODULE-ENTRY

```
...
<BSW-MODULE-ENTRY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Return
    </SHORT-NAME>
    <FUNCTION-PROTOTYPE-EMITTER>Arti</FUNCTION-PROTOTYPE-
       EMITTER>
    <CALL-TYPE>CALLBACK</CALL-TYPE>
    <SW-SERVICE-IMPL-POLICY>INLINE</SW-SERVICE-IMPL-POLICY>
</BSW-MODULE-ENTRY>
<BSW-MODULE-ENTRY>
    <SHORT-NAME>
        Rte_Arti_Runnable_ConsumerComponent_RE2_Start
    </SHORT-NAME>
    <FUNCTION-PROTOTYPE-EMITTER>Arti</FUNCTION-PROTOTYPE-
       EMITTER>
    <CALL-TYPE>CALLBACK</CALL-TYPE>
    <SW-SERVICE-IMPL-POLICY>INLINE</SW-SERVICE-IMPL-POLICY>
</BSW-MODULE-ENTRY>
...
```

- add the REQUIRED-ARTIFACTS

```
...
<BSW-IMPLEMENTATION>
   <SHORT-NAME>Rte</SHORT-NAME>
   <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>
   <REQUIRED-ARTIFACTS>
   ...
```

```
                <DEPENDENCY-ON-ARTIFACT>
                    <SHORT-NAME>Rte_Hook_Arti.h</SHORT-NAME>
                    <CATEGORY>MEMMAP</CATEGORY>
                    <ARTIFACT-DESCRIPTOR>
                        <SHORT-LABEL>Rte_Hook_Arti.h</SHORT-LABEL>
                        <CATEGORY>SWHDR</CATEGORY>
                    </ARTIFACT-DESCRIPTOR>
                    <USAGES>
                        <USAGE>COMPILE</USAGE>
                    </USAGES>
                </DEPENDENCY-ON-ARTIFACT>
            ...
            </REQUIRED-ARTIFACTS>
            ...
        </BSW-IMPLEMENTATION>
        ...
```

- specify the RESOURCE-CONSUMPTION

```
        ...
        <RESOURCE-CONSUMPTION>
            ...
            <MEMORY-SECTION>
                <SHORT-NAME>RTE_Arti_CODE</SHORT-NAME>
                <EXECUTABLE-ENTITY-REFS>
                    <EXECUTABLE-ENTITY-REF DEST="BSW-CALLED-ENTITY"
                       BASE="Rte_BSWMD_BswModuleDescriptions">
                        Rte/RteInternalBehavior/
                            Rte_Arti_Runnable_ConsumerComponent_RE2_Return

                    </EXECUTABLE-ENTITY-REF>
                    <EXECUTABLE-ENTITY-REF DEST="BSW-CALLED-ENTITY"
                       BASE="Rte_BSWMD_BswModuleDescriptions">
                        Rte/RteInternalBehavior/
                            Rte_Arti_Runnable_ConsumerComponent_RE2_Start

                    </EXECUTABLE-ENTITY-REF>
                </EXECUTABLE-ENTITY-REFS>
                <PREFIX-REF DEST="SECTION-NAME-PREFIX"
                            BASE="Rte_BSWMD_BswImplementations">
                    Rte/ResConsumption/RTE_Arti
                </PREFIX-REF>
                <SW-ADDRMETHOD-REF DEST="SW-ADDR-METHOD">
                    /AUTOSAR_MemMap/SwAddrMethods/CODE
                </SW-ADDRMETHOD-REF>
                <SYMBOL>CODE</SYMBOL>
            </MEMORY-SECTION>
            ...
        </RESOURCE-CONSUMPTION>
        ...
```

7.5 Error Classification

Chapter [4, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.5.1 Development Errors

[SWS_Arti_91002] Definition of development errors in module Arti

Upstream requirements: SRS_BSW_00337

⌈

Type of error	*Related error code*	*Error value*
Initialization of ARTI module failed	ARTI_E_INIT_FAILED	0x01
API parameter checking: invalid pointer	ARTI_E_PARAM_POINTER	0x02

⌋

7.5.2 Runtime Errors

There are no runtime errors.

7.5.3 Production Errors

There are no production errors.

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_Arti_91006] Definition of imported datatypes of module Arti

Upstream requirements: SRS_BSW_00301

⌈

Module	Header File	Imported Type
There are no imported types.		

⌋

8.2 Type definitions

ARTI does not add any type definitions.

8.3 Symbol definitions

8.3.1 ARTI_STOPWATCH_FLAT, ARTI_STOPWATCH_NESTED

To support diverse profiling scenarios, the specification mandates a mechanism to distinguish between flat and nested measurement modes. This allows the user to explicitly define whether the timing analysis should account for the depth of calls (e.g., recursive algorithms) or treat the interval as a single continuous block.

[SWS_Arti_91000] Definition of datatype ARTI_STOPWATCH_FLAT, ARTI_STOPWATCH_NESTED ⌈

Name	ARTI_STOPWATCH_FLAT, ARTI_STOPWATCH_NESTED		
Kind	Enumeration		
Range	ARTI_STOPWATCH_FLAT	0x00u	The stopwatch ignores nesting.
	ARTI_STOPWATCH_NESTED	0x01u	The stopwatch considers nesting.
Description	This value defines how the start and stop event of a stopwatch is evaluated. The symbols ARTI_STOPWATCH_FLAT and ARTI_STOPWATCH_NESTED shall be defined as follows: 1. define ARTI_STOPWATCH_FLAT 0x00u 2. define ARTI_STOPWATCH_NESTED 0x01u		
Available via	Arti.h		

⌋

8.4 Function definitions

This chapter contains all APIs provided by the module.

8.4.1 Arti_Init

[SWS_Arti_91004] Definition of API function Arti_Init

Upstream requirements: SRS_BSW_00101

⌈

Service Name	Arti_Init
Syntax	`void Arti_Init ( void )`
Service ID [hex]	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Service to initialize the ARTI module
Available via	Arti.h

⌋

The implementation of `Arti_Init` shall be provided by the tool vendor, that implements ARTI hooks.

[SWS_Arti_00009]

Upstream requirements: SRS_BSW_00337, SRS_BSW_00339

⌈If the initialization fails, the function `Arti_Init` shall raise the error `ARTI_E_INIT_FAILED`.⌋

See [SWS_Arti_91002].

8.5 Scheduled functions

ARTI does not have any functions directly called by Basic Software Scheduler.

8.6 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Common Objectives for ARTI Instrumentation

The ARTI tracing and measurement interfaces are designed to ensure consistent, low-overhead system analysis:

Standardization and Tool Interoperability: A primary objective across all ARTI interfaces is to establish a standardized mechanism for instrumentation. This ensures that trace data and timing metrics can be consistently interpreted by diverse external debugging, profiling, and timing analysis tools. By mandating a standardized format, ARTI eliminates the need for developers to rely on proprietary or ad-hoc recording methods.

User-Defined Instrumentation: Multiple interfaces support user-defined instrumentation, allowing developers to manually insert standardized hooks to trace arbitrary time intervals, values, or specific logic paths.

Minimal Intrusion: To ensure instrumentation does not degrade ECU timing or stability, ARTI hooks utilize a fixed format implemented via C-style macros. This design allows the trace points to be easily expanded by consuming tools or completely removed during normal operation, causing no intrusion or runtime overhead.

8.6.2 Mandatory interfaces

8.6.2.1 ARTI Tracing Macro

The implementation utilizes C-style macros rather than standard function calls. This approach allows trace points to be efficiently expanded or removed from the binary, a deviation from standard coding rules explicitly permitted for performance-critical sections.

[SWS_Arti_00017] ARTI Tracing Macro Parameters ⌈ARTI tracing macros shall accept a set of parameters that define the semantics of the macros.⌋

The ARTI tracing macros are used by all modules with ARTI trace capabilities, therefore ARTI based instrumentation can easily be disabled on a global level.

[SWS_Arti_00018] ARTI Tracing Macro Format (single event) ⌈If the event to trace accepts one single event parameter, then ARTI shall use a single event parameter version.⌋

```
ARTI_TRACE(_contextName, _className, _instanceName, instancePa-
rameter, _eventName, eventParameter)
```

If there are more event parameters specified, then use the multiple event parameter version specified in chapter 8.6.2.2.

Note: While MISRA-C rules may advise to not to use function-like macros, the usage of a macro is intentionally specified here to improve runtime behavior. See also [SRS_BSW_00330].

Note: The ARTI tracing macros should not contain any BSW module API as parameter within the macro invocation, as the macro expansion does not guarantee a proper evaluation of API calls. Within the implementation of an ARTI macro, special care shall be taken about nesting and possible recursions if it calls BSW module APIs.

Some of the parameters come as tokens (literal text) rather than as symbolic identifiers. This allows a macro definition to concatenate these parameters to more specific and efficient macros. Passing and evaluating all parameters as symbolic identifiers at run-time would be very costly especially by means of run-time consumption.

Here is a possible implementation of the generic `ARTI_TRACE` macro:

```
#define ARTI_TRACE(_contextName, _className, _instanceName, \
                   instanceParameter, _eventName, eventParameter) \
    ARTI_TRACE ## _ ## _className ## _ ## _instanceName \
               ## _ ## _eventName ## _ ## _contextName \
               ((instanceParameter), (eventParameter))
```

Listing 8.1: Example implementation of `ARTI_TRACE` macro

Such an implementation will generate one hook for all the possible combinations of `_contextName`, `_className`, `_instanceName` and `_eventName` and pass parameters `instanceParameter` and `eventParameter` at run-time only. The parameters' meanings are described in the following.

`_contextName` Token, literal text, name of the context. One of the following:

- **`NOSUSP`** indicating that the hook gets called in a context where all interrupts are disabled that are covered by the OS service SuspendAllInterrupts
- **`SPRVSR`** indicating that the called hook may disable interrupts during this call. The OS functions must not be used for disabling interrupts. Disabling Interrupts can influence the runtime behavior.
- **`USER`** indicating the called hook cannot disable interrupts by itself. If it is necessary to disable interrupts, the appropriate OS functions have to be used. Disabling Interrupts can influence the runtime behavior.

`_className` Token, literal text, name of the class of macros. Classes can be one of the predefined classes (e.g. `AR_CP_OS_TASK`) or user defined. The predefined classes are specified in the SWS of the according BSW module (e.g. SWS_OS).

`_instanceName` Name of an instance

`instanceParameter` Index [uint32] 0..4294967295 of the instance of a particular `_className` and `_instanceName`, the index should start with 0 and be consecutive per `_instanceName`.

`_eventName` Token, literal text, name of the event as defined for a particular class (e.g. `OsTask_Start`).

`eventParameter` A [uint32] 0..4294967295 value as an argument to an event (e.g. Task Index).

All modules which shall support ARTI tracing shall add calls to this macro with the module specific parameters.

The parameters that are marked as *token, literal text* can't be:

- C macros
- Variables
- Constants
- Enumerations

These parameters are meant to be subject of *token concatenation* by the C preprocessor or the trace tool provider (provider of *<Mip>_Arti.h* and *Arti.h*) chooses to map these tokens to symbols within *<Mip>_Arti.h* depending on the trace tool.

Examples:

Examples for _className AR_CP_OS_TASK where _instanceParameter specifies Core ID and _eventParameter specifies Task ID:

1 OS on 2 cores the OS short name is *OsA*, the OS manages three physical CPU cores.

- ```
  ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 0, OsTask_Start,
  0);
  /* OS OsA start of Task with index 0 on its own Core 0
  */
  ```
- ```
  ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 1, OsTask_Start,
  0);
  /* OS OsA start of Task with index 0 on its own Core 1
  */
  ```

2 OSs on 1 physical core the OS short names are *OsA* and *OsB*, both run on the same physical CPU core (e.g. Hypervisor)

- ```
  ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 0, OsTask_Start,
  0);
  /* OS OsA start of Task with index 0 on its own Core 0
  */
  ```
- ```
  ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsB, 0, OsTask_Start,
  0);
  /* OS OsB start of Task with index 0 on its own Core 0
  */
  ```

2 OSs on 4 cores the OS short names are *OsA* and *OsB* each OS manages two physical CPU cores.

- ```
  ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 0, OsTask_Start,
  0);
  ```
  ```

```
/* OS OsA start of Task with index 0 on its own Core 0
*/
```

- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 1, OsTask_Start, 0);`
  `/* OS OsA start of Task with index 0 on its own Core 1 */`
- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsB, 0, OsTask_Start, 0);`
  `/* OS OsB start of Task with index 0 on its own Core 0 */`
- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsB, 1, OsTask_Start, 0);`
  `/* OS OsB start of Task with index 0 on its own Core 1 */`

2 OSs, 2 virtual cores each and 3 physical cores the OS short names are *OsA* and *OsB* each OS manages two virtual CPU cores (e.g. Hypervisor manages the three physical CPU cores).

- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 0, OsTask_Start, 0);`
  `/* OS OsA start of Task with index 0 on its own Core 0 */`
- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsA, 1, OsTask_Start, 0);`
  `/* OS OsA start of Task with index 0 on its own Core 1 */`
- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsB, 0, OsTask_Start, 0);`
  `/* OS OsB start of Task with index 0 on its own Core 0 */`
- `ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OsB, 1, OsTask_Start, 0);`
  `/* OS OsB start of Task with index 0 on its own Core 1 */`

`AMODULE`, a user defined class with a single instance called `AModule1`.

- `ARTI_TRACE(SPRVSR, AMODULE, AModule1, 0, Thing_Start, 123);`

8.6.2.2 ARTI Tracing Macro with Multiple Parameters

If an events accepts more than one parameter, all the specifications in chapter 8.6.2.1 apply, except the following items.

[SWS_Arti_00019] ARTI Tracing Macro Format (multiple events) ⌈If the event to trace accepts more than one parameters, then ARTI shall use the multiple event parameter version.⌋

`ARTI_TRACE_N(_contextName, _className, _instanceName, instanceParameter, _eventName, numEventParameters, ...)`

The event parameters' meaning are described in the following.

`numEventParameters` An integer value that defines how many parameters are given in the eventParameterList.

...(ellipsis) A comma separated list of event parameters. The event parameters are specified in the ArtiHook configuration container by referencing to an appropriate BswModuleEntry (see chapter 10.7.3).

The macro layout allows a macro definition to concatenate the macro parameters to more specific and efficient macros. Here is a possible implementation of the generic `ARTI_TRACE_N` macro:

```
#define ARTI_TRACE_N(_contextName, _className, _instanceName, \
                     instanceParameter, _eventName, \
                     _enventParameterCount, ...) \
    ARTI_TRACE_ ## _enventParameterCount(_contextName, _className, \
                    _instanceName, instanceParameter, _eventName, \
                    __VA_ARGS__)
#define ARTI_TRACE_2(_contextName, _className, _instanceName, \
                     instanceParameter, _eventName, \
                     eventParameter1, eventParameter2) \
    ARTI_TRACE ## _ ## _className ## _ ## _instanceName \
                ## _ ## _eventName ## _2 (instanceParameter, \
                eventParameter1, eventParameter2)
```

Listing 8.2: Example implementation of `ARTI_TRACE_N` macro

Examples using the above macro expansion:

```
#define ARTI_TRACE_AR_CP_RTE_API_SoAd_IfTransmit_Start_2 \
            ( coreId, portId, data ) \
    arti_rteapi_trace = (portId << 16) | (1 << 8) | coreId;
#define ARTI_TRACE_AR_CP_RTE_API_SoAd_IfTransmit_Return_2 \
            ( coreId, portId, data ) \
    arti_rteapi_trace = (portId << 16) | (2 << 8) | coreId;

void Rte_ReadHook_SoAd_Port1_IfTransmit_Start(int data) {
    ARTI_TRACE_N(USER, AR_CP_RTE_API, SoAd, coreId,
        IfTransmit_Start, 2, 1, data);
}
void Rte_ReadHook_SoAd_Port2_IfTransmit_Return(int data) {
    ARTI_TRACE_N(USER, AR_CP_RTE_API, SoAd, coreId,
```

```
14          IfTransmit_Return, 2, 1, data);
15  }
```

Listing 8.3: Example usage of `ARTI_TRACE_N` macro

8.6.3 Optional interfaces

This section defines all interfaces, which are required to fulfill an optional functionality of the module.

8.6.3.1 ARTI Generic Stopwatch

The Generic Stopwatch supports the tracing of arbitrary time intervals defined by distinct Start and Stop events. Developers can manually insert these standardized hooks around specific algorithms to measure their execution duration precisely.

[SWS_Arti_00001] Define USER_STOPWATCH ⌈ARTI shall define a trace class for tracing of arbitrary intervals between a start and a stop event called `USER_STOPWATCH`.⌋

A stopwatch can be used to time between two user defined points in an application. The user can put the corresponding `ARTI_TRACE` macro calls of the class `USER_STOPWATCH` anywhere in the code. An arbitrary number of stopwatches are supported by using different instance names (`_instanceName`). Please note that the trace tool provider might put limits on the number of active stopwatches.

The generic stopwatch provides a flat tracing (ignoring recurrent start/stop events and the task context) and a nested tracing (considering nested start/stop events and the task context).

Nested Stopwatch:

A trace tool tracing a stopwatch shall consider nesting, if the `instanceParameter` is set to `ARTI_STOPWATCH_NESTED`. Timing shall be measured between Start and Stop events belonging to the same nesting level and the same task that emits this event. The task context shall be derived by tracing the OS tasks and Cat2Isrs as defined in the chapter "ARTI Hook Macros" in the SWS_OS. Please note that a started nested stopwatch *cannot* be stopped in another task.

A typical use case of a nested stopwatch is a function trace.

Flat Stopwatch:

A trace tool shall ignore nesting, if the `instanceParameter` is set to `ARTI_STOPWATCH_FLAT`. The trace tool shall at least consider the time between the first Start event and the first Stop event in a given sequence and doesn't need to consider nested Start and Stop events. E.g.

1. **`Start`**
2. *`Start`* (ignored, already started)
3. **`Stop`**
4. *`Stop`* (ignored, no matching `START`)
5. **`Start`**
6. **`Stop`**

Only events in **bold** are considered, time is calculated between *1 and 3* and *5 and 6*.

Note: It is not defined if a stopwatch considers cores. It is implementation dependent, if a stopwatch can cross cores or is bound to a specific core.

[SWS_Arti_00002] Macro USER_STOPWATCH ⌈ARTI macros of the class `USER_STOPWATCH` shall compile one of the following templates:⌋

```
ARTI_TRACE(_contextName, _className, _instanceName, instanceParameter, _eventName, eventParameter);
```

```
ARTI_TRACE_N(_contextName, _className, _instanceName, instanceParameter, _eventName, numEventParameters, ...);
```

Parameter	*Type*	*Description*
`_contextName`	Token, literal text	see 8.6.2.1
`_className`	Token, literal text	`USER_STOPWATCH`
`_instanceName`	Token, literal text	value that identifies the instance of the stopwatch
`instanceParameter`	uint32	`ARTI_STOPWATCH_FLAT`, if nesting is ignored. `ARTI_STOPWATCH_NESTED`, if nesting is considered.
`_eventName`	Token, literal text	value that identifies the event of the timer, one of `Start` or `Stop`
`numEventParameter`	uint32	number of given event parameters
`eventParameter` or `"..."`	-	arbitrary event parameters

Table 8.1: Macro parameters of `USER_STOPWATCH`

A generic stopwatch can be associated with a C function by providing the function name and file name in the ArtiHookAssociatedFunction and ArtiHookFileName attributes of the ArtiHook container.

Example 8.1

flat stopwatch

```
1 ARTI_TRACE(USER, USER_STOPWATCH, myStopwatch, ARTI_STOPWATCH_FLAT,
      Start, 0);
```

```
2 ARTI_TRACE(USER, USER_STOPWATCH, myStopwatch, ARTI_STOPWATCH_FLAT, Stop
    , 0);
```

Listing 8.4: Example usage of `USER_STOPWATCH` as flat stopwatch

Example 8.2

nested stopwatch

```
1  static int myFunction (int par1)
2  {
3      int ret;
4      ARTI_TRACE_N(USER, USER_STOPWATCH, myArbFunction1,
           ARTI_STOPWATCH_NESTED, Start, 1, par1);
5      do_something_1();
6      ARTI_TRACE_N(USER, USER_STOPWATCH, myArbStopWatch1,
           ARTI_STOPWATCH_NESTED, Start, 1, myVar);
7      ret = do_something_2();
8      ARTI_TRACE_N(USER, USER_STOPWATCH, myArbStopWatch1,
           ARTI_STOPWATCH_NESTED, Stop, 1, myVar);
9      do_something_3();
10     ARTI_TRACE_N(USER, USER_STOPWATCH, myArbFunction1,
           ARTI_STOPWATCH_NESTED, Stop, 1, ret);
11     return ret;
12 }
```

Listing 8.5: Example usage of `USER_STOPWATCH` as nested stopwatch

8.6.3.2 ARTI Generic Dataflow Stopwatch

The Generic Dataflow Stopwatch analyzes the dataflow and temporal validity of variables within the system. It fulfills the need to trace arbitrary intervals defined by a single Start event (a data write) and multiple subsequent Stop events (data reads) to calculate critical metrics like minimum and maximum data age.

[SWS_Arti_00003] Define USER_DATAFLOW_STOPWATCH ⌈ARTI shall define a trace class for tracing of arbitrary intervals between a start and several stop events, with the aim to provide insides to a dataflow, called `USER_DATAFLOW_STOPWATCH`.⌋

A dataflow stopwatch can be used to time between *write* and *read* accesses to a given variable. The user can put the corresponding `ARTI_TRACE` macro calls of the class `USER_DATAFLOW_STOPWATCH` anywhere in the code. An arbitrary number of dataflow stopwatches are supported by using different instance names (`_instanceName`). Please note that the trace tool provider might put limits on the number of active dataflow stopwatches.

The trace tool shall at least consider the time between the last `Write` event, the first `Read` and the last `Read` event in a given sequence and doesn't need to consider nested `Write` and `Read` events. E.g.

1. *`Write`* (ignored as it gets overwritten in 2)

2. **`Write`**
3. **`Read`**
4. **`Write`**
5. **`Read`** (min)
6. *`Read`* (ignored, if only consider min and max)
7. **`Read`** (max)

Only events in **bold** are considered, time is calculated between *2 and 3* and *4 and 5/7*. The time between 4 and the 5 yields the **min** data age time, likewise the time between 4 and 7 yields the **max** data age time for the second sequence.

[SWS_Arti_00004] Macro USER_DATAFLOW_STOPWATCH ⌈ARTI macros of the class `USER_DATAFLOW_STOPWATCH` shall compile the following template:⌋

```
ARTI_TRACE(_contextName, _className, _instanceName, instancePa-
rameter, _eventName, eventParameter);
```

Parameter	*Type*	*Description*
`_contextName`	Token, literal text	see 8.6.2.1
`_className`	Token, literal text	`USER_DATAFLOW_STOPWATCH`
`_instanceName`	Token, literal text	value that identifies the instance of the dataflow stopwatch
`instanceParameter`	uint32	Not used, should be set to 0
`_eventName`	Token, literal text	value that identifies the event of the timer, one of `Write` or `Read`
`eventParameter`	uint32	Not used, should be set to 0

Table 8.2: Macro parameters of `USER_DATAFLOW_STOPWATCH`

Example 8.3

```
1 ARTI_TRACE(USER, USER_DATAFLOW_STOPWATCH, myDataflowStopwatch, 0, Write
      , 0);
2 myVariable = 1;
3 ...
4 uint32 temp = myVariable;
5 ARTI_TRACE(USER, USER_DATAFLOW_STOPWATCH, myDataFlowStopwatch, 0, Read,
       0);
```

Listing 8.6: Example usage of `USER_DATAFLOW_STOPWATCH`

8.6.3.3 ARTI Generic Datapoint

The Generic Datapoint interface traces arbitrary values at specific, user-selected points within the application code, to capture the instantaneous state of variables or logic paths during runtime.

[SWS_Arti_00005] Define USER_DATAPOINT ⌈ARTI shall define a trace class for tracing of arbitrary values, called `USER_DATAPOINT`.⌋

A *datapoint* provides the possibility to record different values at user defined locations in the code. The user can put the corresponding `ARTI_TRACE` macro calls of the class `USER_DATAPOINT` anywhere in the code. An arbitrary number of data points are supported by using different instance names (`_instanceName`). Please note that the trace tool provider might put limits on the number of active data points. There are predefined event names (`_eventName`) for different data types as defined by AUTOSAR (see AUTOSAR_SWS_PlatformTypes, e.g. `UINT32`) this information might be used by the trace tool for optimized storage and visualization.

[SWS_Arti_00006] Macro USER_DATAPOINT ⌈ARTI macros of the class `USER_DATAPOINT` shall compile the following template:⌋

```
ARTI_TRACE(_contextName, _className, _instanceName, instanceParameter, _eventName, eventParameter);
```

Parameter	*Type*	*Description*
_contextName	Token, literal text	see 8.6.2.1
_className	Token, literal text	USER_DATAPOINT
_instanceName	Token, literal text	value that identifies the instance of the data point
instanceParameter	uint32	Not used, should be set to 0
_eventName	Token, literal text	Value that identifies the type of the datapoint. The type is a hint for the tool vendor how to interpret the eventParameter, which is always 32bit wide. Shall be one of the following: • BOOLEAN • UINT8 • UINT16 • UINT32 • SINT8 • SINT16 • SINT32 • FLOAT32
eventParameter	uint32	Value that shall be recorded by the event (up to 32-bits)

Table 8.3: Macro parameters of USER_DATAPOINT

Example 8.4

```
1 ARTI_TRACE(USER, USER_DATAPOINT, myDatapoint0, 0, UINT32, 2ul);
2 ARTI_TRACE(USER, USER_DATAPOINT, myDatapoint1, 0, SINT8, s8_Data);
```

Listing 8.7: Example usage of USER_DATAPOINT

8.6.3.4 ARTI Category 1 Interrupts

This interface records and traces hardware-level interrupts that operate outside the management of the operating system. By defining specific trace classes, it enables the accurate measurement of execution times and state transitions for high-priority interrupt routines.

[SWS_Arti_00007] Define AR_CP_ARTI_CAT1ISR ⌈ARTI shall define a trace class for tracing of category 1 interrupts, called AR_CP_ARTI_CAT1ISR.⌋

ARTI needs to trace all states of category 1 interrupts and all its state transitions. For some timing parameters (e.g. the interrupt pending time), the simple interrupt start/stop

is not enough. Tools evaluating the timings need to reconstruct a more complex state diagram by calculating the transitions from history. To be compatible to standard software, AR_CP_ARTI_CAT1ISR refers to this state model, knowing that tools need to postprocess the event flow to get all relevant information. However, if an OS implementation can provide a more detailed state diagram, ARTI allows to define more events that won't need postprocessing and allow earlier synchronization of the trace if it is truncated (limited trace buffers). This state diagram is then handled with the class AR_CP_ARTIEXT_CAT1ISR. If possible, the second state machine is to be preferred.

AR_CP_ARTI_CAT1ISR :

The class AR_CP_ARTI_CAT1ISR contains events allowing the tracing of category 1 interrupts.

The following state diagram shows the states and transitions:

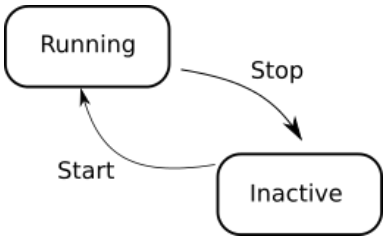

Figure 8.1: ARTI CAT1ISR state machine

Transitions used by ARTI for class AR_CP_ARTI_CAT1ISR:

Name	*Transition*	*Event Name*
Start	Inactive -> Running	OsCat1Isr_Start
Stop	Running -> Inactive	OsCat1Isr_Stop

Table 8.4: Transitions of AR_CP_ARTI_CAT1ISR

AR_CP_ARTIEXT_CAT1ISR :

The class AR_CP_ARTIEXT_CAT1ISR contains events allowing the tracing of category 1 interrupts with an enhanced state model.

The following state diagram shows the state machine as used by ARTI:

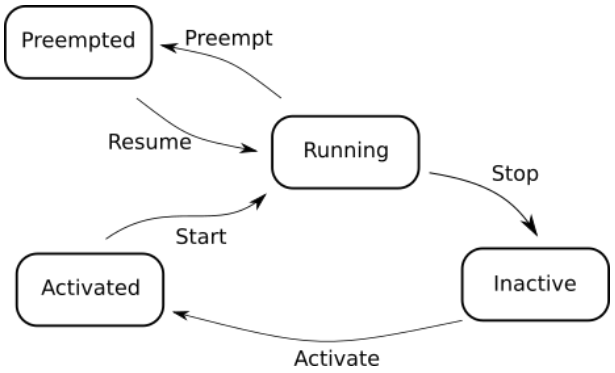

Figure 8.2: ARTI EXT CAT1ISR state machine

States used by ARTI for class `AR_CP_ARTIEXT_CAT1ISR`:

ARTI	*OS*
Inactive	Inactive
Activated	Inactive
Running	Running
Preempted	Running

Table 8.5: States of `AR_CP_ARTIEXT_CAT1ISR`

Transitions used by ARTI for class `AR_CP_ARTIEXT_CAT1ISR`:

Name	*Transition*	*Event Name*
Activate	Inactive -> Activated	`OsCat1Isr_Activate`
Start	Activated -> Running	`OsCat1Isr_Start`
Preempt	Running -> Preempted	`OsCat1Isr_Preempt`
Resume	Preempted -> Running	`OsCat1Isr_Resume`
Stop	Running -> Inactive	`OsCat1Isr_Stop`

Table 8.6: Transitions of `AR_CP_ARTIEXT_CAT1ISR`

[SWS_Arti_00008] Macro AR_CP_ARTI_CAT1ISR ⌈ARTI macros of the classes `AR_CP_ARTI_CAT1ISR` and `AR_CP_ARTIEXT_CAT1ISR` shall compile the following template:⌋

```
1 ARTI_TRACE(_contextName, AR_CP_ARTI_CAT1ISR, <OS Short Name>, <Core
      Index>, <Event Name>, <Cat1Isr Index>)
2 ARTI_TRACE(_contextName, AR_CP_ARTIEXT_CAT1ISR, <OS Short Name>, <Core
      Index>, <Event Name>, <Cat1Isr Index>)
```

The <Core Index> for any event shall represent the core index where the corresponding Cat1Isr is scheduled on.

The <Event Name> should follow the transition table above.

The <Cat1Isr Index> shall be a numeric identifier of the Cat1Isr.

8.6.3.5 ARTI RTE VFB Trace Client

This client establishes a standardized VFB tracing interface within the Run-Time Environment. It maps high-level RTE hooks, such as the start and return of Executable Entities, and RTE API calls, directly to ARTI_TRACE macros. To comply with compile-time configurability, unused or disabled VFB hooks are mapped to empty statements (AR_CP_VOID).

[SWS_Arti_00011] Generate RTE VFB Trace Client ⌈The ARTI module shall generate an RTE VFB trace client that implements the Runnable Entity Trace Events [3] Chapter 5.11.5.6.1 *"Runnable Entity Invocation"* and BSW Schedulable Entities Trace Events [3] Chapter 5.11.5.7 *"BSW Schedulable Entities Trace Events"*.⌋

[SWS_Arti_00012] Define AR_CP_RTE_RUNNABLE ⌈The Runnable Entity Trace Events shall be mapped to the ARTI Trace Class `AR_CP_RTE_RUNNABLE`.⌋

[SWS_Arti_00020] Define AR_CP_RTE_API ⌈The RTE API Trace Events shall be mapped to the ARTI Trace Class `AR_CP_RTE_API`.⌋

[SWS_Arti_00013] Define AR_CP_SCHM_SCHEDULABLE ⌈The BSW Schedulable Entity Trace Events shall be mapped to the ARTI Trace Class `AR_CP_SCHM_SCHEDULABLE`.⌋

Runnable Entity Trace Events: AR_CP_RTE_RUNNABLE

- `Rte_Arti_Runnable_<cts>_<reName>_Start`
- `Rte_Arti_Runnable_<cts>_<reName>_Return`

RTE API Trace Events: AR_CP_RTE_API

- `Rte_Arti_<api>Hook_<cts>_<ap>_Start`
- `Rte_Arti_<api>Hook_<cts>_<ap>_Return`
- `Rte_Arti_<api>Hook_<cts>_<ap>_Terminate`

BSW Schedulable Entities Trace Events: AR_CP_SCHM_SCHEDULABLE

- `SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>_Start`
- `SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>_Return`

8.6.3.5.1 Trace Class – AR_CP_RTE_RUNNABLE

[SWS_Arti_00014] Macro AR_CP_RTE_RUNNABLE ⌈ARTI macros of the class `AR_CP_RTE_RUNNABLE` shall compile the following template:⌋

Runnable Entity Invocation

```
#define Rte_Arti_Runnable_<cts>_<reName>_Start( \
    [const_Rte_CDS_<cts>_ptr]) \
    ARTI_TRACE(_contextName, \
                AR_CP_RTE_RUNNABLE, \
                shortNameOf(<cts>), \
                [const_Rte_CDS_<cts>_ptr]|0, \
                RteRunnable_Start, idOf(<reName>))
```

Listing 8.8: Template of `AR_CP_RTE_RUNNABLE` invocation

Runnable Entity Termination

```
#define Rte_Arti_Runnable_<cts>_<reName>_Return( \
    [const_Rte_CDS_<cts>_ptr]) \
    ARTI_TRACE(_contextName, \
                AR_CP_RTE_RUNNABLE, \
                shortNameOf(<cts>), \
                [const_Rte_CDS_<cts>_ptr]|0, \
                RteRunnable_Return, idOf(<reName>))
```

Listing 8.9: Template of `AR_CP_RTE_RUNNABLE` termination

<cts> Specifies the component type that is emitted by the RTE. For each component type the mapping is created.

<reName> is the name of the Runnable Entity. For each name the mapping is created.

shortNameOf() is a hint of the ARTI module the extract use the short name of the element in question.

idOf() is a function of the ARTI module to create an 32-bit ID out of an element. This mapping will also be stored in a type map within ArtiValues and will be referenced by the hook descriptions.

[] are optional parameters issued by the RTE. If they do exist then they have to be used. If they do not exist they will be replaced by 0 in the `ARTI_TRACE` macro.

Parameter	*Type*	*Description*
_contextName	Token, literal text	see 8.6.2.1, usually this is USER for Runnable Entities.
_className	Token, literal text	AR_CP_RTE_RUNNABLE
_instanceName	Token, literal text	Is the short name of the <cts>, the component type symbol of the AtomicSwComponentType
instanceParameter	uint32	Is used in case of multiple instantiation. In this case the instance handle as specified in the RTE VFB trace client is used. If single instantiation is used this parameter is 0.
_eventName	Token, literal text	value that identifies the event type of the Runnable Entity • RteRunnable_Start • RteRunnable_Return
eventParameter	uint32	represents the ID of the <reName>, the ID of the Runnable Entity which is generated by the ARTI module.

Table 8.7: Macro parameters of AR_CP_RTE_RUNNABLE

8.6.3.5.2 Trace Class – AR_CP_SCHM_SCHEDULABLE

[SWS_Arti_00015] Macro AR_CP_SCHM_SCHEDULABLE ⌈ARTI macros of the class AR_CP_SCHM_SCHEDULABLE shall compile the following template:⌋

BSW Schedulable Entities Invocation

```
#define SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>
    _Start() \
     ARTI_TRACE(_contextName, \
                AR_CP_SCHM_SCHEDULABLE, \
                <bsnp>, \
                idOf([<vi>_<ai>])|0, \
                SchMSchedulable_Start, \
                idOf(<entityName>))
```

Listing 8.10: Template of AR_CP_SCHM_SCHEDULABLE invocation

BSW Schedulable Entities Termination

```
#define SchM_Arti_Schedulable_<bsnp>_[<vi>_<ai>_]<entityName>
    _Return() \
     ARTI_TRACE(_contextName, \
                AR_CP_SCHM_SCHEDULABLE, \
                <bsnp>, \
                idOf([<vi>_<ai>])|0, \
```

```
6                   SchMSchedulable_Return, \
7                   idOf(<entityName>))
```

Listing 8.11: Template of `AR_CP_SCHM_SCHEDULABLE` termination

As defined in the RTE specification:

<bsnp> specifies the Basic Software Name Prefix

<vi> is the Vendor ID of the basic software module

<ai> is the Vendor API infix of the basic software module

<entityName> is the name of the BSW Schedulable Entity or Callable Entity

idOf() is a function of the ARTI module to create an 32-bit ID out of an element. This mapping will also be stored in a type map within ArtiValues and will be referenced by the hook descriptions.

[] are optional parameters issued by the RTE. If they do exist then they have to be used. If they do not exist they will be replaced by 0 in the `ARTI_TRACE` macro.

Parameter	*Type*	*Description*
`_contextName`	Token, literal text	see 8.6.2.1 usually this is `NOSUSP` for Schedulable Entities.
`_className`	Token, literal text	`AR_CP_SCHM_SCHEDULABLE`
`_instanceName`	Token, literal text	The `<bsnp>`, the BSW Scheduler Name Prefix of the basic software module.
`instanceParameter`	uint32	Is used when vendorId and vendorApiInfix of the BSW module are specified. In this case the ARTI module generated an ID for the used pair of vendorIp and vendorApiInfix. If vendorId and vendorApiInfix is not given this parameter is 0.
`_eventName`	Token, literal text	value that identifies the event type of the Schedulable Entity • SchmSchedulable_Start • SchmSchedulable_Return
`eventParameter`	uint32	represents the ID of the `<entityName>`, the ID of the Schedulable Entity which is generated by the ARTI module.

Table 8.8: Macro parameters of `AR_CP_SCHM_SCHEDULABLE`

8.6.3.5.3 Trace Class – AR_CP_RTE_API

[SWS_Arti_00021] Macro AR_CP_RTE_API ⌈ARTI macros of the class `AR_CP_RTE_API` shall compile the following template:⌋

RTE API Start

```
#define Rte_Arti_<api>Hook_<cts>_<ap>_Start( \
    [const_Rte_CDS_<cts> * instance_ptr], param...) \
    ARTI_TRACE_N(USER, \
                 AR_CP_RTE_API, \
                 shortNameOf(<cts>), \
                 <instance_ptr|0>, \
                 <api>_Start, \
                 numberOf(param...) + 1, \
                 idOf(<ap>), \
                 <param...>)
```

Listing 8.12: Template of `AR_CP_RTE_API` start

RTE API Return

```
#define Rte_Arti_<api>Hook_<cts>_<ap>_Return( \
    [const_Rte_CDS_<cts> * instance_ptr], param...) \
    ARTI_TRACE_N(USER, \
                 AR_CP_RTE_API, \
                 shortNameOf(<cts>), \
                 <instance_ptr|0>, \
                 <api>_Return, \
                 numberOf(param...) + 1, \
                 idOf(<ap>), \
                 <param...>)
```

Listing 8.13: Template of `AR_CP_RTE_API` return

RTE API Terminate

```
#define Rte_Arti_<api>Hook_<cts>_<ap>_Terminate( \
    [const_Rte_CDS_<cts> * instance_ptr], param...) \
    ARTI_TRACE_N(USER, \
                 AR_CP_RTE_API, \
                 shortNameOf(<cts>), \
                 <instance_ptr|0>, \
                 <api>_Terminate, \
                 numberOf(param...) + 1, \
                 idOf(<ap>), \
                 <param...>)
```

Listing 8.14: Template of `AR_CP_RTE_API` terminate

As defined in the RTE specification:

<api> Specifies the name of the function this hook relates to.

<cts> Specifies the component type symbol of the AtomicSwComponentType.

<ap> Specifies the access point name (e.g. port and data element or operation name, exclusive area name, etc.).

shortNameOf() is a hint of the ARTI module the extract use the short name of the element in question.

idOf() is a function of the ARTI module to create an 32-bit ID out of an element. This mapping will also be stored in a type map within ArtiValues and will be referenced by the hook descriptions.

numberOf(params...) is the number of parameters of the related RTE trace hook. This number can be determined by the referenced BswModuleEntry.arguments.

[] are parameters issued by the RTE as defined by BswModuleEntry.arguments. If they do exist then they have to be used. If they do not exist they will be replaced by 0 in the `ARTI_TRACE` macro.

Parameter	*Type*	*Description*
`_contextName`	Token, literal text	see 8.6.2.1 usually this is `USER`.
`_className`	Token, literal text	`AR_CP_RTE_API`
`_instanceName`	Token, literal text	Is the short name of the `<cts>`, the component type symbol of the `AtomicSwComponentType`
`instanceParameter`	uint32	Is used in case of multiple instantiation. In this case the instance handle as specified in the RTE VFB trace client is used. If single instantiation is used this parameter is 0.
`_eventName`	Token, literal text	value that identifies the event type of the API • <api>_Start • <api>_Return • <api>_Terminate
`numEventParameter`	uint32	the number of parameters plus one for the ID of the access port
. . . (ellipsis)		First argument is uint32 which represents the ID of the access port which is generated by the ARTI module. Additional arguments are the parameters of the related RTE hook.

Table 8.9: Macro parameters of `AR_CP_RTE_API`

8.6.3.5.4 Trace Class – AR_CP_VOID

[SWS_Arti_00016] Macro AR_CP_VOID ⌈ARTI shall provide a macro of the class `AR_CP_VOID` that compiles to nothing.⌋

`AR_CP_VOID` is used to map VFB tracing hooks that are not used by ARTI. Expanding `ARTI_TRACE` with trace class `AR_CP_VOID` should result in empty statement that results in no code at all.

Parameter	*Type*	*Description*
_contextName	Token, literal text	see 8.6.2.1 this should be USER for AR_CP_VOID.
_className	Token, literal text	AR_CP_VOID
_instanceName	Token, literal text	Not used, set to ""
instanceParameter	uint32	Not used, should be set to 0
_eventName	Token, literal text	Not used, set to ""
eventParameter	uint32	Not used, should be set to 0

Table 8.10: Macro parameters of AR_CP_VOID

8.6.3.6 ARTI BSW Module Interface

This interface provides a standardized mechanism for recording and tracing the entry and exit events of Basic Software modules. By defining a specific trace class, it correlates BSW execution time with application behavior, contributing to a fully standardized tracing interface.

[SWS_Arti_00022] Define AR_CP_BSW_API ⌈ARTI shall define a trace class for tracing entries and exits of BSW modules, called AR_CP_BSW_API.⌋

To allow tracing of BSW modules, the entry and exit points of BSW modules should be populated with ARTI_TRACE macros.

[SWS_Arti_00023] Macro AR_CP_BSW_API ⌈ARTI macros of the class AR_CP_BSW_API shall compile the following template:⌋

```
ARTI_TRACE_N(_contextName, \
             AR_CP_BSW_API, \
             <moduleImplementationPrefix>, \
             0, \
             Bsw_<Start|Return>, \
             numberOf(param...) + 1, \
             idOf(<service>), \
             <param...>)
```

Listing 8.15: Template of AR_CP_BSW_API

<moduleImplementationPrefix> represents a module, formed as specified in [4] Chapter 5.1.1 *"Module implementation prefix"* [SWS_BSW_00102].

numberOf(params...) specifies the number of parameters following in the parantheses. This number can be determined by the referenced BswModuleEntry.arguments.

idOf(<service) represents the service ID as specified in the function definition of the BSW module.

<param...> is a comma separated list of parameters that are passed to the service (in/inout) or returned by the service (return/out/inout) as defined by BswModuleEntry.arguments.

Parameter	*Type*	*Description*
`_contextName`	Token, literal text	see 8.6.2.1.
`_className`	Token, literal text	`AR_CP_BSW_API`
`_instanceName`	Token, literal text	Is the module implementation prefix of the module being called
`instanceParameter`	uint32	Reserved. Set to 0.
`_eventName`	Token, literal text	value that identifies the event type of the API • Bsw_Start • Bsw_Return
`numEventParameter`	uint32	the number of parameters plus one for the ID of the service
`...`(ellipsis)		First argument is uint32 which represents the ID of the service. Additional arguments are the parameters of the related BSW hook.

Table 8.11: Macro parameters of `AR_CP_BSW_API`

For example:

```
1 Std_ReturnType SoAd_IfTransmit(PduIdType TxPduId, const PduInfoType*
      PduInfoPtr)
2 {
3     Std_ReturnType ret = E_OK;
4     ARTI_TRACE_N(USER, AR_CP_BSW_API, SoAd_42_ArbitraryApiInfix, 0,
          Bsw_Start, 3, 0x49, TxPduId, PduInfoPtr);
5     // do SoAd_IfTransmit actions
6     ARTI_TRACE_N(USER, AR_CP_BSW_API, SoAd_42_ArbitraryApiInfix, 0,
          Bsw_Return, 2, 0x49, ret);
7     return ret;
8 }
```

Listing 8.16: Example usage of `AR_CP_BSW_API`

The BSW Module has to provide an BswModuleEntry description defining the arguments and return value of the BSW API. The generated ArtiHook description shall refer to this BswModuleEntry. (See chapter 10.7.3 "ArtiHook").

8.6.4 Callout interfaces

ARTI does not define callout interfaces.

8.6.5 Configurable interfaces

ARTI does not define configurable interfaces.

8.7 Service Interfaces

ARTI does not provide any service interfaces.

9 Sequence diagrams

Not applicable yet.

10 Configuration specification

This chapter defines configuration parameters and their clustering into containers.

Containers and parameters that are related to the OS module are specified in [2] Chapter 10.2 *"Containers and configuration parameters"*.

10.1 How to read this chapter

For details refer to the [4] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 ARTI Parameters

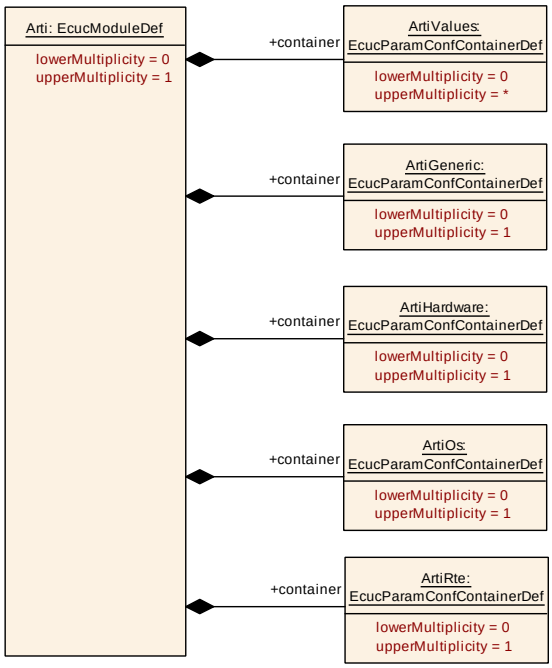

Figure 10.1: Arti Ecuc Module Definition Class Diagram

[ECUC_Arti_00001] Definition of EcucModuleDef Arti ⌈

Module Name	Arti
Description	The Arti Module serves as a superordinate container collecting all information and parameters concerning ARTI.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiGeneric	0..1	The ArtiGeneric container contains definitions for generic objects, i.e. not belonging to a standard AUTOSAR module.
ArtiHardware	0..1	The ArtiHardware container contains ARTI extensions to the EcucHardware module.
ArtiOs	0..1	The ArtiOs container contains ARTI extensions to the EcucDefs/Os module.
ArtiRte	0..1	The ArtiRte Container contains all parameters for ARTI that are filled by the generators RTE.
ArtiValues	0..*	The ArtiValues container collects all parameter values for ARTI that are filled by the generators (OS, RTE, ...)

⌋

10.3 ARTI Generic Container

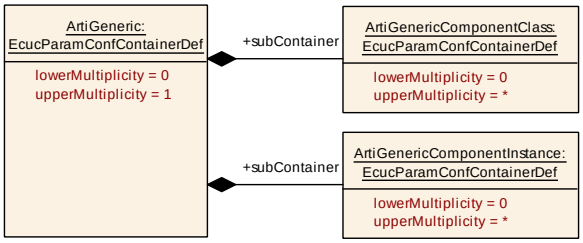

Figure 10.2: ArtiGeneric Ecuc Module Definition Class Diagram

[ECUC_Arti_00042] Definition of EcucParamConfContainerDef ArtiGeneric ⌈

Container Name	ArtiGeneric
Parent Container	Arti
Upstream mapping defined	–
Description	The ArtiGeneric container contains definitions for generic objects, i.e. not belonging to a standard AUTOSAR module.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiGenericComponentClass	0..*	The class definition describes the layout of the object (similar to a "class" definition in C++).
ArtiGenericComponentInstance	0..*	The instance definition describes a specific instantiated object.

⌋

Example 10.1

Example Values for the ArtiGeneric Container

```
<AR-PACKAGE>
  <SHORT-NAME>Vendor1</SHORT-NAME>
  <ELEMENTS>
    <ECUC-MODULE-CONFIGURATION-VALUES>
      <SHORT-NAME>Vendor1Arti</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Arti</
          DEFINITION-REF>
      <CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>Vendor1ArtiGeneric</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiGeneric</DEFINITION-REF>
          <SUB-CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentClass_AModule</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentClass_RteWiperSwc</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentClass_Vendor1Task</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentInstance_AModule1</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentInstance_TaskHighPriority</
                  SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiGenericComponentInstance_Wiper</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance</
                  DEFINITION-REF>
            </ECUC-CONTAINER-VALUE>
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
      </CONTAINERS>
    </ECUC-MODULE-CONFIGURATION-VALUES>
  </ELEMENTS>
```

```
</AR-PACKAGE>
```

Listing 10.1: ARTI **EcuC Container** ARXML **Listing for ArtiGeneric**

10.3.1 ArtiGenericComponentClass

[ECUC_Arti_00043] Definition of EcucParamConfContainerDef ArtiGenericComponentClass ⌈

Container Name	ArtiGenericComponentClass
Parent Container	ArtiGeneric
Upstream mapping defined	–
Description	The class definition describes the layout of the object (similar to a "class" definition in C++).
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
ArtiGenericComponentClassName	1	[ECUC_Arti_00044]

Included Containers		
Container Name	Multiplicity	Description
ArtiGenericComponentClass Parameter	0..*	Parameter definition of a class.

⌋

[ECUC_Arti_00044] Definition of EcucStringParamDef ArtiGenericComponentClassName ⌈

Parameter Name	ArtiGenericComponentClassName
Parent Container	ArtiGenericComponentClass
Upstream mapping defined	–
Description	Name of the class.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00045] Definition of EcucParamConfContainerDef ArtiGenericComponentClassParameter ⌈

Container Name	ArtiGenericComponentClassParameter
Parent Container	ArtiGenericComponentClass
Upstream mapping defined	–
Description	Parameter definition of a class.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
ArtiGenericComponentClassParameterName	1	[ECUC_Arti_00046]
ArtiGenericComponentClassParameterTypeMapRef	0..1	[ECUC_Arti_00053]

No Included Containers

⌋

[ECUC_Arti_00046] Definition of EcucStringParamDef ArtiGenericComponentClassParameterName ⌈

Parameter Name	ArtiGenericComponentClassParameterName
Parent Container	ArtiGenericComponentClassParameter
Upstream mapping defined	–
Description	Name of the parameter.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00053] Definition of EcucReferenceDef ArtiGenericComponentClassParameterTypeMapRef ⌈

Parameter Name	ArtiGenericComponentClassParameterTypeMapRef
Parent Container	ArtiGenericComponentClassParameter
Upstream mapping defined	–
Description	Refers to a parameter type to interpret the parameter value.
Multiplicity	0..1
Type	Reference to ArtiParameterTypeMap
Pre-Compile only	true

⌋

Example 10.2

Example Values for an ArtiGenericComponentClass Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiGenericComponentClass_AModule</SHORT-NAME>
```

```
<DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
    Arti/ArtiGeneric/ArtiGenericComponentClass</DEFINITION-REF>
<PARAMETER-VALUES>
  <ECUC-TEXTUAL-PARAM-VALUE>
    <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
        ArtiGeneric/ArtiGenericComponentClass/
        ArtiGenericComponentClassName</DEFINITION-REF>
    <VALUE>AModule</VALUE>
  </ECUC-TEXTUAL-PARAM-VALUE>
</PARAMETER-VALUES>
<SUB-CONTAINERS>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>AModule_RunningThing</SHORT-NAME>
    <DESC>
      <L-2 S="" L="FOR-ALL">Running Thing</L-2>
    </DESC>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
        EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass/
        ArtiGenericComponentClassParameter</DEFINITION-REF>
    <PARAMETER-VALUES>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
            Arti/ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassParameter/
            ArtiGenericComponentClassParameterName</DEFINITION-REF>
        <VALUE>RunningThing</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
    </PARAMETER-VALUES>
    <REFERENCE-VALUES>
      <ECUC-REFERENCE-VALUE>
        <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti
            /ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassParameter/
            ArtiGenericComponentClassParameterTypeMapRef</DEFINITION-REF
            >
        <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
            Vendor1ArtiValues/ArtiParameterTypeMap_RunningThing</VALUE-
            REF>
      </ECUC-REFERENCE-VALUE>
    </REFERENCE-VALUES>
  </ECUC-CONTAINER-VALUE>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>AModule_ThingStart</SHORT-NAME>
    <DESC>
      <L-2 S="" L="FOR-ALL">Thing Start</L-2>
    </DESC>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
        EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass/
        ArtiGenericComponentClassParameter</DEFINITION-REF>
    <PARAMETER-VALUES>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
            Arti/ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassParameter/
            ArtiGenericComponentClassParameterName</DEFINITION-REF>
        <VALUE>ThingStart</VALUE>
```

```
        </ECUC-TEXTUAL-PARAM-VALUE>
      </PARAMETER-VALUES>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti
              /ArtiGeneric/ArtiGenericComponentClass/
              ArtiGenericComponentClassParameter/
              ArtiGenericComponentClassParameterTypeMapRef</DEFINITION-REF
              >
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
              Vendor1ArtiValues/ArtiParameterTypeMap_ThingStart</VALUE-REF
              >
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 10.2: ARTI **EcuC Container** ARXML **Listing for ArtiGenericComponentClass**

10.3.2 ArtiGenericComponentInstance

[ECUC_Arti_00049] Definition of EcucParamConfContainerDef ArtiGenericComponentInstance ⌈

Container Name	ArtiGenericComponentInstance
Parent Container	ArtiGeneric
Upstream mapping defined	–
Description	The instance definition describes a specific instantiated object.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiGenericComponentInstanceName	1	[ECUC_Arti_00050]
ArtiGenericComponentInstanceClassRef	1	[ECUC_Arti_00048]

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiGenericComponentInstance Parameter	0..*	Parameter definition of an instance.

⌋

[ECUC_Arti_00050] Definition of EcucStringParamDef ArtiGenericComponentInstanceName ⌈

Parameter Name	ArtiGenericComponentInstanceName
Parent Container	ArtiGenericComponentInstance
Upstream mapping defined	–
Description	Name of the instance.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00048] Definition of EcucReferenceDef ArtiGenericComponentInstanceClassRef ⌈

Parameter Name	ArtiGenericComponentInstanceClassRef
Parent Container	ArtiGenericComponentInstance
Upstream mapping defined	–
Description	Refers to a ArtGenericClass of which this object is instantiated.
Multiplicity	1
Type	Reference to ArtiGenericComponentClass
Pre-Compile only	true

⌋

[ECUC_Arti_00051] Definition of EcucParamConfContainerDef ArtiGenericComponentInstanceParameter ⌈

Container Name	ArtiGenericComponentInstanceParameter
Parent Container	ArtiGenericComponentInstance
Upstream mapping defined	–
Description	Parameter definition of an instance.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiGenericComponentInstanceParameterClassParameterRef	0..*	[ECUC_Arti_00047]
ArtiGenericComponentInstanceParameterConstantRef	0..1	[ECUC_Arti_00040]
ArtiGenericComponentInstanceParameterExpressionRef	0..1	[ECUC_Arti_00041]
ArtiGenericComponentInstanceParameterHookRef	0..1	[ECUC_Arti_00052]

No Included Containers

⌋

[ECUC_Arti_00047] Definition of EcucReferenceDef ArtiGenericComponentInstanceParameterClassParameterRef ⌈

Parameter Name	ArtiGenericComponentInstanceParameterClassParameterRef
Parent Container	ArtiGenericComponentInstanceParameter
Upstream mapping defined	–
Description	Refers to an ArtiGenericComponentClassParameter that defines this parameter.
Multiplicity	0..*
Type	Reference to ArtiGenericComponentClassParameter
Pre-Compile only	true

⌋

[ECUC_Arti_00040] Definition of EcucReferenceDef ArtiGenericComponentInstanceParameterConstantRef ⌈

Parameter Name	ArtiGenericComponentInstanceParameterConstantRef
Parent Container	ArtiGenericComponentInstanceParameter
Upstream mapping defined	–
Description	Refers to an ArtiConstant that represents the value of this parameter.
Multiplicity	0..1
Type	Reference to ArtiConstant
Pre-Compile only	true

⌋

[ECUC_Arti_00041] Definition of EcucReferenceDef ArtiGenericComponentInstanceParameterExpressionRef ⌈

Parameter Name	ArtiGenericComponentInstanceParameterExpressionRef
Parent Container	ArtiGenericComponentInstanceParameter
Upstream mapping defined	–
Description	Refers to an ArtiExpression that evaluates the value of this parameter.
Multiplicity	0..1
Type	Reference to ArtiExpression
Pre-Compile only	true

⌋

[ECUC_Arti_00052] Definition of EcucReferenceDef ArtiGenericComponentInstanceParameterHookRef ⌈

Parameter Name	ArtiGenericComponentInstanceParameterHookRef
Parent Container	ArtiGenericComponentInstanceParameter
Upstream mapping defined	–
Description	Refers to a hook that records this parameter.
Multiplicity	0..1
Type	Reference to ArtiHook
Pre-Compile only	true

⌋

Example 10.3

Example Values for an ArtiGenericComponentInstance Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiGenericComponentInstance_AModule1</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiGeneric/ArtiGenericComponentInstance</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceName</DEFINITION-REF>
      <VALUE>AModule1</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceClassRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          Vendor1ArtiGeneric/ArtiGenericComponentClass_AModule</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>AModule1_RunningThing</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceParameter</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti
              /ArtiGeneric/ArtiGenericComponentInstance/
              ArtiGenericComponentInstanceParameter/
              ArtiGenericComponentInstanceParameterExpressionRef</
              DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
              Vendor1ArtiValues/
              ArtiExpression_ArtiGeneric_AModule1_RunningThing</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti
              /ArtiGeneric/ArtiGenericComponentInstance/
              ArtiGenericComponentInstanceParameter/
              ArtiGenericComponentInstanceParameterClassParameterRef</
              DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
              Vendor1ArtiGeneric/ArtiGenericComponentClass_AModule/
              AModule_RunningThing</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
```

```
</ECUC-CONTAINER-VALUE>
```

Listing 10.3: ARTI **EcuC Container** ARXML **Listing for ArtiGenericComponentInstance**

10.4 ARTI Hardware Container

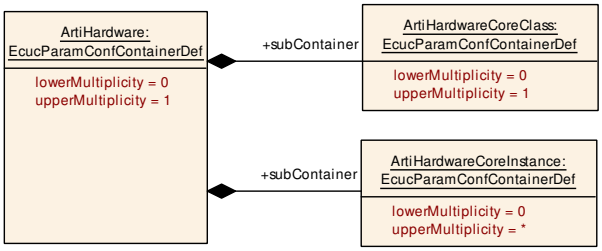

Figure 10.3: ArtiHardware Ecuc Module Definition Class Diagram

The ArtiHardware container is specified in SWS_OS.

10.5 ARTI Os Container

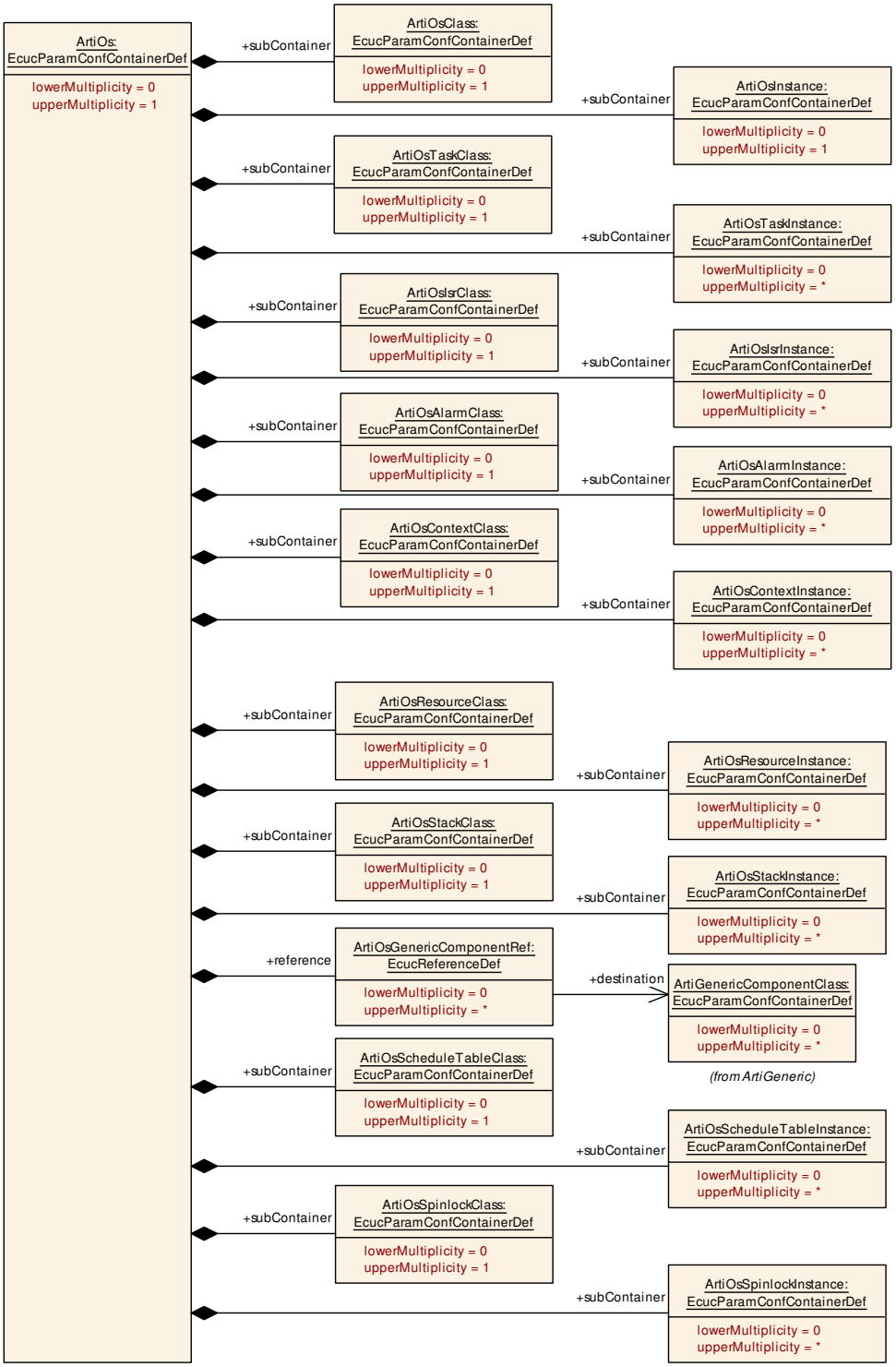

Figure 10.4: ArtiOs Ecuc Module Definition Class Diagram

The ArtiOs container is specified in SWS_OS.

10.6 ARTI Rte Container

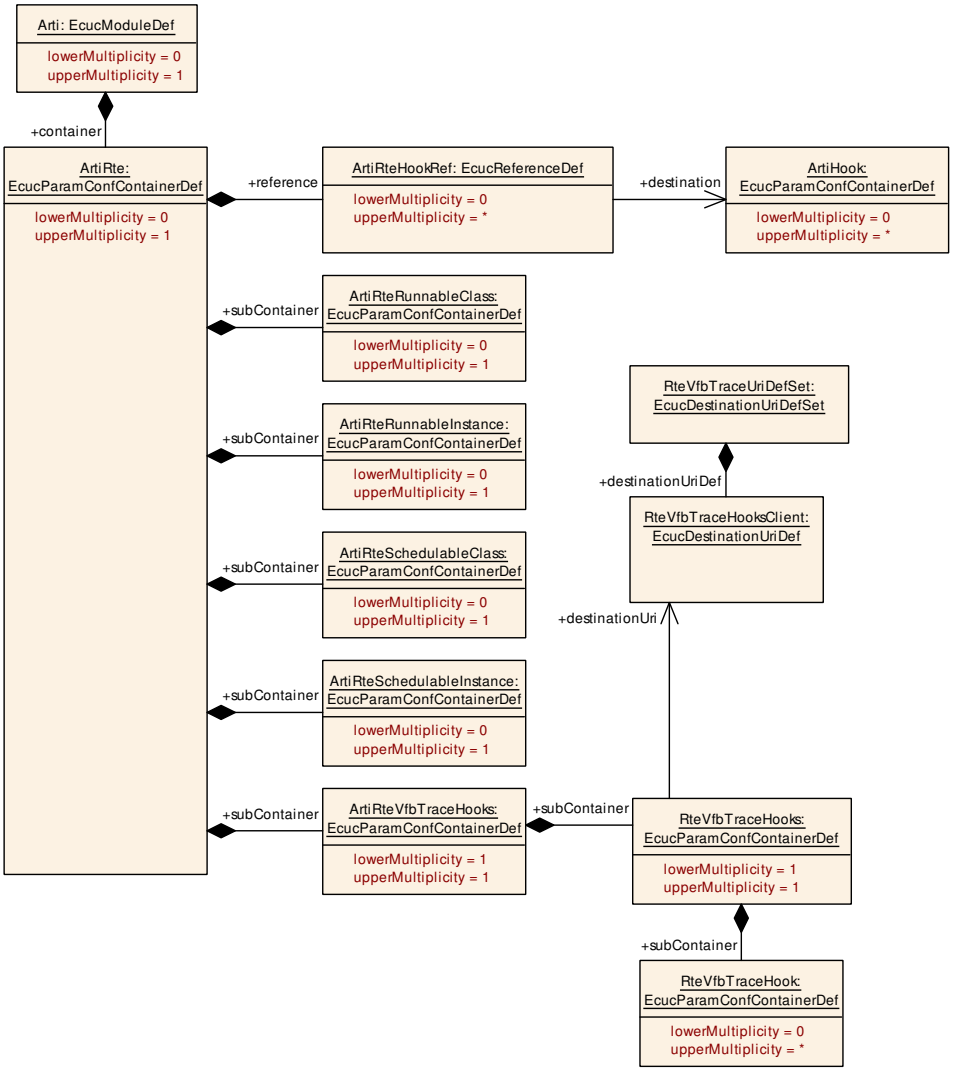

Figure 10.5: ArtiRte Ecuc Module Definition Class Diagram

[ECUC_Arti_00158] Definition of EcucParamConfContainerDef ArtiRte ⌈

Container Name	ArtiRte
Parent Container	Arti
Upstream mapping defined	–
Description	The ArtiRte Container contains all parameters for ARTI that are filled by the generators RTE.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiRteHookRef	0..*	[ECUC_Arti_00159]

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiRteRunnableClass	0..1	Contains the layout of an ArtiRteRunnable object.
ArtiRteRunnableInstance	0..1	Represents an instance of an ArtiRteRunnable object, extending the BswM RunnableEntity.
ArtiRteSchedulableClass	0..1	Contains the layout of an ArtiRteSchedulable object.
ArtiRteSchedulableInstance	0..1	Represents an instance of an ArtiRteSchedulable object, extending the Rte Schedulable Entity.
ArtiRteVfbTraceHooks	1	This container defines the parent container to which all trace hook containers are added.

⌋

[ECUC_Arti_00159] Definition of EcucReferenceDef ArtiRteHookRef ⌈

Parameter Name	ArtiRteHookRef
Parent Container	ArtiRte
Upstream mapping defined	–
Description	Refers to an arti hook which is called by the RTE.
Multiplicity	0..*
Type	Reference to ArtiHook
Pre-Compile only	true

⌋

10.6.1 ArtiRteRunnableClass

[ECUC_Arti_00160] Definition of EcucParamConfContainerDef ArtiRteRunnable Class ⌈

Container Name	ArtiRteRunnableClass
Parent Container	ArtiRte
Upstream mapping defined	–
Description	Contains the layout of an ArtiRteRunnable object.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiRteRunnableClassGenericComponentClassRef	0..1	[ECUC_Arti_00164]
ArtiRteRunnableIdRef	1	[ECUC_Arti_00165]

No Included Containers

⌋

[ECUC_Arti_00164] Definition of EcucReferenceDef ArtiRteRunnableClass GenericComponentClassRef ⌈

Parameter Name	ArtiRteRunnableClassGenericComponentClassRef
Parent Container	ArtiRteRunnableClass
Upstream mapping defined	–
Description	Refers to an ArtiGenericComponentClass that extends the ArtiRteRunnableClass.
Multiplicity	0..1
Type	Reference to ArtiGenericComponentClass
Pre-Compile only	true

⌋

[ECUC_Arti_00165] Definition of EcucReferenceDef ArtiRteRunnableIdRef ⌈

Parameter Name	ArtiRteRunnableIdRef
Parent Container	ArtiRteRunnableClass
Upstream mapping defined	–
Description	Refers to the ArtiObjectClassParameter that declares the attribute ArtiRteRunnableId Ref in ArtiRteRunnableEntityInstances. This attribute specifies the idOf(reName) mapping.
Multiplicity	1
Type	Reference to ArtiObjectClassParameter
Pre-Compile only	true

⌋

10.6.2 ArtiRteRunnableInstance

[ECUC_Arti_00161] Definition of EcucParamConfContainerDef ArtiRteRunnable Instance ⌈

Container Name	ArtiRteRunnableInstance
Parent Container	ArtiRte
Upstream mapping defined	–
Description	Represents an instance of an ArtiRteRunnable object, extending the BswM Runnable Entity.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiRteRunnableInstanceSymbol	0..1	[ECUC_Arti_00166]
ArtiRteRunnableInstanceBswRef	0..1	[ECUC_Arti_00167]
ArtiRteRunnableInstanceGenericComponentInstanceRef	0..1	[ECUC_Arti_00168]

No Included Containers

⌋

[ECUC_Arti_00166] Definition of EcucStringParamDef ArtiRteRunnableInstance Symbol ⌈

Parameter Name	ArtiRteRunnableInstanceSymbol
Parent Container	ArtiRteRunnableInstance
Upstream mapping defined	–
Description	Specifies the symbol / function name that implements the runnable.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00167] Definition of EcucForeignReferenceDef ArtiRteRunnableInstanceBswRef ⌈

Parameter Name	ArtiRteRunnableInstanceBswRef
Parent Container	ArtiRteRunnableInstance
Upstream mapping defined	–
Description	Refers to an Rte Runnable that is beeing extended.
Multiplicity	0..1
Type	Foreign reference to RUNNABLE-ENTITY
Pre-Compile only	true

⌋

[ECUC_Arti_00168] Definition of EcucReferenceDef ArtiRteRunnableInstance GenericComponentInstanceRef ⌈

Parameter Name	ArtiRteRunnableInstanceGenericComponentInstanceRef
Parent Container	ArtiRteRunnableInstance
Upstream mapping defined	–
Description	Refers to an ArtiGenericComponentInstance that extends the ArtiRteRunnable Instance.
Multiplicity	0..1
Type	Reference to ArtiGenericComponentInstance
Pre-Compile only	true

⌋

10.6.3 ArtiRteSchedulableClass

[ECUC_Arti_00162] Definition of EcucParamConfContainerDef ArtiRteSchedulableClass ⌈

Container Name	ArtiRteSchedulableClass
Parent Container	ArtiRte
Upstream mapping defined	–
Description	Contains the layout of an ArtiRteSchedulable object.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiRteSchedulableClassGenericComponentClassRef	0..1	[ECUC_Arti_00169]
ArtiRteSchedulableIdRef	1	[ECUC_Arti_00170]

No Included Containers

⌋

[ECUC_Arti_00169] Definition of EcucReferenceDef ArtiRteSchedulableClass GenericComponentClassRef ⌈

Parameter Name	ArtiRteSchedulableClassGenericComponentClassRef
Parent Container	ArtiRteSchedulableClass
Upstream mapping defined	–
Description	Refers to an ArtiGenericComponentClass that extends the ArtiRteSchedulableClass.
Multiplicity	0..1
Type	Reference to ArtiGenericComponentClass
Pre-Compile only	true

⌋

[ECUC_Arti_00170] Definition of EcucReferenceDef ArtiRteSchedulableIdRef ⌈

Parameter Name	ArtiRteSchedulableIdRef
Parent Container	ArtiRteSchedulableClass
Upstream mapping defined	–
Description	Refers to the ArtiObjectClassParameter that declares the attribute ArtiRteSchmEntityId Ref in ArtiRteSchedulableInstances. This attribute specifies the idOf(entityName) mapping.
Multiplicity	1
Type	Reference to ArtiObjectClassParameter
Pre-Compile only	true

⌋

10.6.4 ArtiRteSchedulableInstance

[ECUC_Arti_00163] Definition of EcucParamConfContainerDef ArtiRteSchedulableInstance ⌈

Container Name	ArtiRteSchedulableInstance
Parent Container	ArtiRte
Upstream mapping defined	–
Description	Represents an instance of an ArtiRteSchedulable object, extending the Rte Schedulable Entity.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
ArtiRteSchedulableInstanceSymbol	0..1	[ECUC_Arti_00171]
ArtiRteSchedulableInstanceBswRef	0..1	[ECUC_Arti_00172]
ArtiRteSchedulableInstanceGenericComponentInstanceRef	0..1	[ECUC_Arti_00173]

No Included Containers

⌋

[ECUC_Arti_00171] Definition of EcucStringParamDef ArtiRteSchedulableInstanceSymbol ⌈

Parameter Name	ArtiRteSchedulableInstanceSymbol
Parent Container	ArtiRteSchedulableInstance
Upstream mapping defined	–
Description	Specifies the symbol / function name that implements the schedulable.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00172] Definition of EcucForeignReferenceDef ArtiRteSchedulableInstanceBswRef ⌈

Parameter Name	ArtiRteSchedulableInstanceBswRef
Parent Container	ArtiRteSchedulableInstance
Upstream mapping defined	–
Description	Refers to an Rte Schedulable that is beeing extended.
Multiplicity	0..1
Type	Foreign reference to BSW-SCHEDULABLE-ENTITY
Pre-Compile only	true

⌋

[ECUC_Arti_00173] Definition of EcucReferenceDef ArtiRteSchedulableInstance GenericComponentInstanceRef ⌈

Parameter Name	ArtiRteSchedulableInstanceGenericComponentInstanceRef
Parent Container	ArtiRteSchedulableInstance
Upstream mapping defined	–
Description	Refers to an ArtiGenericComponentInstance that extends the ArtiRteSchedulable Instance.
Multiplicity	0..1
Type	Reference to ArtiGenericComponentInstance
Pre-Compile only	true

⌋

10.6.5 ArtiRteVfbTraceHooks

[ECUC_Arti_00177] Definition of EcucParamConfContainerDef ArtiRteVfbTrace Hooks ⌈

Container Name	ArtiRteVfbTraceHooks
Parent Container	ArtiRte
Upstream mapping defined	–
Description	This container defines the parent container to which all trace hook containers are added.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	Multiplicity	Description
RteVfbTraceHooks	1	This container defines the parent container to which all trace hook containers are added.

⌋

10.7 ARTI Values Container

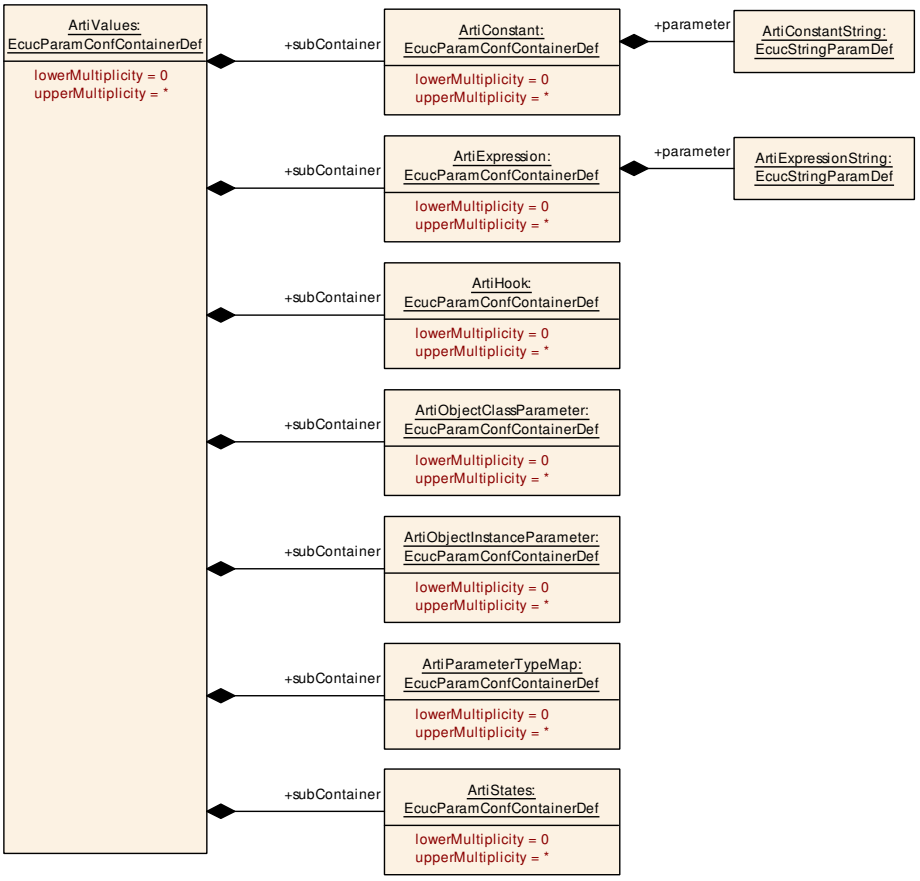

Figure 10.6: ArtiValues Ecuc Module Definition Class Diagram

[ECUC_Arti_00002] Definition of EcucParamConfContainerDef ArtiValues ⌈

Container Name	ArtiValues
Parent Container	Arti
Upstream mapping defined	–
Description	The ArtiValues container collects all parameter values for ARTI that are filled by the generators (OS, RTE, ...)
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiConstant	0..*	This container holds a constant value.
ArtiExpression	0..*	This container holds a C like expression that a debugger can evaluate. This is similar to what is already done in ORTI.
ArtiHook	0..*	This container represents an ARTI hook that is present in the module.

▽

△

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiObjectClassParameter	0..*	This container represents a parameter of an Arti object class definition.
ArtiObjectInstanceParameter	0..*	This container represents a parameter of an Arti object instance.
ArtiParameterTypeMap	0..*	A map of key/value pairs to map a parameter value to a display string and/or an Arti or EcuC object.
ArtiStates	0..*	This container contains all states of tasks, isrs... that the EcuC uses.

⌋

Example 10.4

Example Values for the ArtiValues Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>Vendor1ArtiValues</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE S="">
      <SHORT-NAME>ArtiConstant_ArtiSwc_WiperLocation_Front</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiConstant</DEFINITION-REF>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiExpression_ArtiHwCore_CurrentTaskOnCore0</SHORT-
          NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiExpression</DEFINITION-REF>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE UUID="">
      <SHORT-NAME>ArtiHook_ArtiOs_TaskStart</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiHook</DEFINITION-REF>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiObjectClassParameter_ArtiHwCore_CurrentApplication<
          /SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiObjectClassParameter</DEFINITION-
          REF>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiObjectInstanceParameter_CurrentApplicationOnCore0</
          SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiObjectInstanceParameter</DEFINITION
          -REF>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiParameterTypeMap_Core</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap</DEFINITION-REF>
```

```
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 10.4: ARTI **EcuC Container** ARXML **Listing for ArtiValues**

10.7.1 ArtiConstant

[ECUC_Arti_00006] Definition of EcucParamConfContainerDef ArtiConstant ⌈

Container Name	ArtiConstant
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container holds a constant value.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiConstantString	1	[ECUC_Arti_00008]

No Included Containers

⌋

[ECUC_Arti_00008] Definition of EcucStringParamDef ArtiConstantString ⌈

Parameter Name	ArtiConstantString
Parent Container	ArtiConstant
Upstream mapping defined	–
Description	This is the constant value for a specific parameter.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

Example 10.5

Example Values for an ArtiConstant Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiConstant_ArtiSwc_WiperLocation_Front</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiConstant</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiConstant/ArtiConstantString</DEFINITION-REF>
```

```
      <VALUE>Front</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
</ECUC-CONTAINER-VALUE>
```

Listing 10.5: ARTI **EcuC Container** ARXML **Listing for ArtiConstant**

10.7.2 ArtiExpression

[ECUC_Arti_00009] Definition of EcucParamConfContainerDef ArtiExpression ⌈

Container Name	ArtiExpression
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container holds a C like expression that a debugger can evaluate. This is similar to what is already done in ORTI.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiExpressionString	1	[ECUC_Arti_00011]

No Included Containers

⌋

[ECUC_Arti_00011] Definition of EcucStringParamDef ArtiExpressionString ⌈

Parameter Name	ArtiExpressionString	
Parent Container	ArtiExpression	
Upstream mapping defined	–	
Description	This string represents a C like expression that a debugger can evaluate.	
Multiplicity	1	
Type	EcucStringParamDef	
Default value	–	
Regular Expression	–	
Post-Build Value	-	Selectable

⌋

Example 10.6

Example Values for an ArtiExpression Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiExpression_ArtiHwCore_CurrentTaskOnCore0</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiExpression</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
```

```
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
            ArtiValues/ArtiExpression/ArtiExpressionString</DEFINITION-REF>
        <VALUE>Os_ControlledCoreInfo[0U].RunningTask</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
    </PARAMETER-VALUES>
  </ECUC-CONTAINER-VALUE>
```

Listing 10.6: ARTI **EcuC Container** ARXML **Listing for ArtiExpression**

10.7.3 ArtiHook

[ECUC_Arti_00012] Definition of EcucParamConfContainerDef ArtiHook ⌈

Container Name	ArtiHook
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container represents an ARTI hook that is present in the module.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiHookAssociatedFunction	0..1	[ECUC_Arti_00180]
ArtiHookClass	1	[ECUC_Arti_00013]
ArtiHookContext	1	[ECUC_Arti_00014]
ArtiHookEventName	1	[ECUC_Arti_00015]
ArtiHookFileName	0..1	[ECUC_Arti_00181]
ArtiHookInstance	1	[ECUC_Arti_00017]
ArtiHookEntryRef	0..1	[ECUC_Arti_00179]
ArtiHookEventParameterTypeRef	0..1	[ECUC_Arti_00016]
ArtiHookInstanceParameterTypeRef	0..1	[ECUC_Arti_00018]

No Included Containers

⌋

[ECUC_Arti_00180] Definition of EcucFunctionNameDef ArtiHookAssociated Function ⌈

Parameter Name	ArtiHookAssociatedFunction
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Name of the C function that is associated with this hook.
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–

▽

△

Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00013] Definition of EcucStringParamDef ArtiHookClass ⌈

Parameter Name	ArtiHookClass
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Name of the (schedule) class of macros. Classes can be one of the predefined classes or user defined.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00014] Definition of EcucStringParamDef ArtiHookContext ⌈

Parameter Name	ArtiHookContext
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Name of the execution context. One of NOSUSP, SPRVSR, or USER. See also chapter "ARTI Tracing Macro".
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00015] Definition of EcucStringParamDef ArtiHookEventName ⌈

Parameter Name	ArtiHookEventName
Parent Container	ArtiHook
Upstream mapping defined	–
Description	The name of the event as defined for a particular class, or an arbitrary name for generic classes.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00181] Definition of EcucStringParamDef ArtiHookFileName ⌈

Parameter Name	ArtiHookFileName
Parent Container	ArtiHook
Upstream mapping defined	–
Description	File name (possibly with path) where this ArtiHook is implemented.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00017] Definition of EcucStringParamDef ArtiHookInstance ⌈

Parameter Name	ArtiHookInstance
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Name of an instance of the (schedule) class.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00179] Definition of EcucForeignReferenceDef ArtiHookEntryRef ⌈

Parameter Name	ArtiHookEntryRef
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Reference to the BswModuleEntry that describes the parameters of this hook.
Multiplicity	0..1
Type	Foreign reference to BSW-MODULE-ENTRY
Pre-Compile only	true

⌋

[ECUC_Arti_00016] Definition of EcucReferenceDef ArtiHookEventParameterTypeRef ⌈

Parameter Name	ArtiHookEventParameterTypeRef
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Refers to a parameter type to interpret the hook event number.
Multiplicity	0..1
Type	Reference to ArtiParameterTypeMap
Pre-Compile only	true

⌋

[ECUC_Arti_00018] Definition of EcucReferenceDef ArtiHookInstanceParameterTypeRef ⌈

Parameter Name	ArtiHookInstanceParameterTypeRef
Parent Container	ArtiHook
Upstream mapping defined	–
Description	Refers to a parameter type to interpret the hook instance number.
Multiplicity	0..1
Type	Reference to ArtiParameterTypeMap
Pre-Compile only	true

⌋

Example 10.7

Example Values for an ArtiHook Container

```
<ECUC-CONTAINER-VALUE UUID="">
  <SHORT-NAME>ArtiHook_ArtiOs_TaskStart</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiHook</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookClass</DEFINITION-REF>
      <VALUE>AR_CP_OS_TASK</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookContext</DEFINITION-REF>
      <VALUE>NOSUSP</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookEventName</DEFINITION-REF>
      <VALUE>OsTask_Start</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookInstance</DEFINITION-REF>
      <VALUE>Vendor1OsCore</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookEventParameterTypeRef</DEFINITION-
          REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          Vendor1ArtiValues/ArtiParameterTypeMap_TaskId</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookInstanceParameterTypeRef</DEFINITION
          -REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          Vendor1ArtiValues/ArtiParameterTypeMap_Core</VALUE-REF>
```

```
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Listing 10.7: ARTI **EcuC Container** ARXML **Listing for ArtiHook**

10.7.4 ArtiObjectClassParameter

[ECUC_Arti_00020] Definition of EcucParamConfContainerDef ArtiObjectClass Parameter ⌈

Container Name	ArtiObjectClassParameter
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container represents a parameter of an Arti object class definition.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiObjectClassParameterTypeMapRef	0..1	[ECUC_Arti_00028]

No Included Containers

⌋

[ECUC_Arti_00028] Definition of EcucReferenceDef ArtiObjectClassParameter TypeMapRef ⌈

Parameter Name	ArtiObjectClassParameterTypeMapRef
Parent Container	ArtiObjectClassParameter
Upstream mapping defined	–
Description	Refers to a parameter type to interpret the instance parameter value.
Multiplicity	0..1
Type	Reference to ArtiParameterTypeMap
Pre-Compile only	true

⌋

Example 10.8

Example Values for an ArtiObjectClassParameter Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiObjectClassParameter_ArtiHwCore_CurrentTask</SHORT-NAME>
  <DESC>
    <L-2>Current Running AUTOSAR Task</L-2>
  </DESC>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
     Arti/ArtiValues/ArtiObjectClassParameter</DEFINITION-REF>
```

```
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiObjectClassParameter/
          ArtiObjectClassParameterTypeMapRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          Vendor1ArtiValues/ArtiParameterTypeMap_TaskExpr</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Listing 10.8: ARTI **EcuC Container** ARXML **Listing for ArtiObjectClassParameter**

10.7.5 ArtiObjectInstanceParameter

[ECUC_Arti_00021] Definition of EcucParamConfContainerDef ArtiObjectInstanceParameter ⌈

Container Name	ArtiObjectInstanceParameter
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container represents a parameter of an Arti object instance.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiObjectInstanceParameterConstantRef	0..1	[ECUC_Arti_00007]
ArtiObjectInstanceParameterExpressionRef	0..1	[ECUC_Arti_00010]
ArtiObjectInstanceParameterHookRef	0..1	[ECUC_Arti_00019]

No Included Containers

⌋

[ECUC_Arti_00007] Definition of EcucReferenceDef ArtiObjectInstanceParameterConstantRef ⌈

Parameter Name	ArtiObjectInstanceParameterConstantRef
Parent Container	ArtiObjectInstanceParameter
Upstream mapping defined	–
Description	Refers to a constant representing the value of this parameter.
Multiplicity	0..1
Type	Reference to ArtiConstant
Pre-Compile only	true

⌋

[ECUC_Arti_00010] Definition of EcucReferenceDef ArtiObjectInstanceParameterExpressionRef ⌈

Parameter Name	ArtiObjectInstanceParameterExpressionRef
Parent Container	ArtiObjectInstanceParameter
Upstream mapping defined	–
Description	Refers to an expression that evaluates the value of this parameter.
Multiplicity	0..1
Type	Reference to ArtiExpression
Pre-Compile only	true

⌋

[ECUC_Arti_00019] Definition of EcucReferenceDef ArtiObjectInstanceParameterHookRef ⌈

Parameter Name	ArtiObjectInstanceParameterHookRef
Parent Container	ArtiObjectInstanceParameter
Upstream mapping defined	–
Description	Refers to a hook that records this parameter.
Multiplicity	0..1
Type	Reference to ArtiHook
Pre-Compile only	true

⌋

Example 10.9

Example Values for an ArtiObjectInstanceParameter Container

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiObjectInstanceParameter_CurrentTaskOnCore0</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiObjectInstanceParameter</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiObjectInstanceParameter/
          ArtiObjectInstanceParameterExpressionRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          Vendor1ArtiValues/ArtiExpression_ArtiHwCore_CurrentTaskOnCore0</
          VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Listing 10.9: ARTI **EcuC Container** ARXML **Listing for ArtiObjectInstanceParameter**

10.7.6 ArtiParameterTypeMap

[ECUC_Arti_00022] Definition of EcucParamConfContainerDef ArtiParameterTypeMap ⌈

Container Name	ArtiParameterTypeMap
Parent Container	ArtiValues
Upstream mapping defined	–
Description	A map of key/value pairs to map a parameter value to a display string and/or an Arti or EcuC object.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	Multiplicity	Description
ArtiParameterTypeMapPair	1..*	A key/value pair to map a parameter value to a display string and/or an Arti or EcuC object.

⌋

[ECUC_Arti_00023] Definition of EcucParamConfContainerDef ArtiParameterTypeMapPair ⌈

Container Name	ArtiParameterTypeMapPair
Parent Container	ArtiParameterTypeMap
Upstream mapping defined	–
Description	A key/value pair to map a parameter value to a display string and/or an Arti or EcuC object.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
ArtiParameterTypeMapPairInput	0..1	[ECUC_Arti_00024]
ArtiParameterTypeMapPairOutput	0..1	[ECUC_Arti_00026]
ArtiParameterTypeMapPairInputExpressionRef	0..1	[ECUC_Arti_00025]
ArtiParameterTypeMapPairOutputRef	0..1	[ECUC_Arti_00027]

No Included Containers

⌋

[ECUC_Arti_00024] Definition of EcucIntegerParamDef ArtiParameterTypeMapPairInput ⌈

Parameter Name	ArtiParameterTypeMapPairInput
Parent Container	ArtiParameterTypeMapPair
Upstream mapping defined	–

▽

△

Description	The numerical value given by a parameter to translate. When used with ArtiHooks, this parameter is mandatory (multiplicity 1) and its value is limited to the range of 0..65535. This parameter may be used to map the values given by "instanceParameter" and/or the "eventParameter" of the ARTI_TRACE macro.
Multiplicity	0..1
Type	EcucIntegerParamDef
Range	0 .. 18446744073709551615
Default value	–
Pre-Compile only	true

⌋

[ECUC_Arti_00026] Definition of EcucStringParamDef ArtiParameterTypeMapPairOutput ⌈

Parameter Name	ArtiParameterTypeMapPairOutput
Parent Container	ArtiParameterTypeMapPair
Upstream mapping defined	–
Description	The string to display for the Input value.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Arti_00025] Definition of EcucReferenceDef ArtiParameterTypeMapPairInputExpressionRef ⌈

Parameter Name	ArtiParameterTypeMapPairInputExpressionRef
Parent Container	ArtiParameterTypeMapPair
Upstream mapping defined	–
Description	Refers to an expression that evaluates to a numerical value to translate.
Multiplicity	0..1
Type	Reference to ArtiExpression
Pre-Compile only	true

⌋

[ECUC_Arti_00027] Definition of EcucChoiceReferenceDef ArtiParameterTypeMapPairOutputRef ⌈

Parameter Name	ArtiParameterTypeMapPairOutputRef
Parent Container	ArtiParameterTypeMapPair
Upstream mapping defined	–
Description	Choice Reference to ArtiOsTaskInstance, ArtiOsIsrInstance, ArtiStatesTaskState, Os AppMode, ArtiOsContextInstance, or ArtiOsStackInstance.

▽

△

Multiplicity	0..1
Type	Choice reference to [ArtiOsContextInstance, ArtiOsIsrInstance, ArtiOsStackInstance, ArtiOsTaskInstance, ArtiStatesScheduleTableState, ArtiStatesSpinlockOwnerType, ArtiStatesSpinlockState, ArtiStatesTaskState, OsAppMode]
Pre-Compile only	true

⌋

Example 10.10

Example Values for an ArtiParameterTypeMap Containers

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiParameterTypeMap_OsAppMode</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
     Arti/ArtiValues/ArtiParameterTypeMap</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>AppModeDefault</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap/
         ArtiParameterTypeMapPair</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-TEXTUAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
             Arti/ArtiValues/ArtiParameterTypeMap/
             ArtiParameterTypeMapPair/ArtiParameterTypeMapPairInput</
             DEFINITION-REF>
          <VALUE>1</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
        <ECUC-TEXTUAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
             Arti/ArtiValues/ArtiParameterTypeMap/
             ArtiParameterTypeMapPair/ArtiParameterTypeMapPairOutput</
             DEFINITION-REF>
          <VALUE>OSDEFAULTAPPMODE</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>AppModeNone</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap/
         ArtiParameterTypeMapPair</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-TEXTUAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
             Arti/ArtiValues/ArtiParameterTypeMap/
             ArtiParameterTypeMapPair/ArtiParameterTypeMapPairInput</
             DEFINITION-REF>
          <VALUE>0</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
        <ECUC-TEXTUAL-PARAM-VALUE>
```

```
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiValues/ArtiParameterTypeMap/
                ArtiParameterTypeMapPair/ArtiParameterTypeMapPairOutput</
                DEFINITION-REF>
            <VALUE>OS_APPMODE_NONE</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>ArtiParameterTypeMap_TaskExpr</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
       Arti/ArtiValues/ArtiParameterTypeMap</DEFINITION-REF>
    <SUB-CONTAINERS>
      <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>Task_1</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap/
           ArtiParameterTypeMapPair</DEFINITION-REF>
        <PARAMETER-VALUES>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiValues/ArtiParameterTypeMap/
                ArtiParameterTypeMapPair/ArtiParameterTypeMapPairOutput</
                DEFINITION-REF>
            <VALUE>Task_1</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
        <REFERENCE-VALUES>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti
                /ArtiValues/ArtiParameterTypeMap/ArtiParameterTypeMapPair/
                ArtiParameterTypeMapPairInputExpressionRef</DEFINITION-REF>
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
                Vendor1ArtiValues/ArtiExpression_AddressOfTask1</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>ArtiParameterTypeMap_TaskId</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
       Arti/ArtiValues/ArtiParameterTypeMap</DEFINITION-REF>
    <SUB-CONTAINERS>
      <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>TaskHighPrio</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap/
           ArtiParameterTypeMapPair</DEFINITION-REF>
        <PARAMETER-VALUES>
          <ECUC-TEXTUAL-PARAM-VALUE>
```

```
            <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiValues/ArtiParameterTypeMap/
                ArtiParameterTypeMapPair/ArtiParameterTypeMapPairInput</
                DEFINITION-REF>
            <VALUE>1</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiValues/ArtiParameterTypeMap/
                ArtiParameterTypeMapPair/ArtiParameterTypeMapPairOutput</
                DEFINITION-REF>
            <VALUE>HighPriority</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
```

Listing 10.10: ARTI **EcuC Container** ARXML **Listing for ArtiParameterTypeMap**

10.7.7 ArtiStates

[ECUC_Arti_00029] Definition of EcucParamConfContainerDef ArtiStates ⌈

Container Name	ArtiStates
Parent Container	ArtiValues
Upstream mapping defined	–
Description	This container contains all states of tasks, isrs... that the EcuC uses.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiStatesTaskEnhanced	1	[ECUC_Arti_00032]

Included Containers		
Container Name	**Multiplicity**	**Description**
ArtiStatesScheduleTableState	0..*	Each state used by the OsScheduleTable has to be listed as Arti StatesScheduleTableState parameter with a choice of the states.
ArtiStatesSpinlockOwnerType	0..*	Each owner type used by the OsSpinlock has to be listed as Arti StatesSpinlockOwnerType parameter with a choice of the states.
ArtiStatesSpinlockState	0..*	Each state used by the OsSpinlock has to be listed as ArtiStates SpinlockState parameter with a choice of the states.
ArtiStatesTaskState	0..*	Each state used by the OS has to be listed as ArtiStatesTask State Parameter with a choice of the states.

⌋

[ECUC_Arti_00032] Definition of EcucBooleanParamDef ArtiStatesTaskEnhanced ⌈

Parameter Name	ArtiStatesTaskEnhanced
Parent Container	ArtiStates
Upstream mapping defined	–
Description	Set to true, if the OS provides an "enhanced" state model with "READY" split to "Activated", "Preempted", "Released".
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Arti_00205] Definition of EcucParamConfContainerDef ArtiStatesScheduleTableState ⌈

Container Name	ArtiStatesScheduleTableState
Parent Container	ArtiStates
Upstream mapping defined	–
Description	Each state used by the OsScheduleTable has to be listed as ArtiStatesScheduleTable State parameter with a choice of the states.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiStatesScheduleTableStateEnum	1	[ECUC_Arti_00206]

No Included Containers

⌋

[ECUC_Arti_00206] Definition of EcucEnumerationParamDef ArtiStatesScheduleTableStateEnum ⌈

Parameter Name	ArtiStatesScheduleTableStateEnum	
Parent Container	ArtiStatesScheduleTableState	
Upstream mapping defined	–	
Description	ArtiStatesScheduleTableState choice of the states.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ArtiScheduleTableStateNext	activated
	ArtiScheduleTableStateRunning	activated
	ArtiScheduleTableStateRunning AndSynchronous	activated
	ArtiScheduleTableStateStopped	activated

▽

△

	ArtiScheduleTableStateWaiting	activated
Pre-Compile only	true	

⌋

[ECUC_Arti_00207] Definition of EcucParamConfContainerDef ArtiStatesSpinlockOwnerType ⌈

Container Name	ArtiStatesSpinlockOwnerType
Parent Container	ArtiStates
Upstream mapping defined	–
Description	Each owner type used by the OsSpinlock has to be listed as ArtiStatesSpinlockOwner Type parameter with a choice of the states.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiStatesSpinlockOwnerTypeEnum	1	[ECUC_Arti_00208]

No Included Containers

⌋

[ECUC_Arti_00208] Definition of EcucEnumerationParamDef ArtiStatesSpinlockOwnerTypeEnum ⌈

Parameter Name	ArtiStatesSpinlockOwnerTypeEnum	
Parent Container	ArtiStatesSpinlockOwnerType	
Upstream mapping defined	–	
Description	ArtiStatesSpinlockOwnerType choice of the states.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ArtiSpinlockOwnerTypeIsr2	activated
	ArtiSpinlockOwnerTypeTask	activated
Pre-Compile only	true	

⌋

[ECUC_Arti_00209] Definition of EcucParamConfContainerDef ArtiStatesSpinlockState ⌈

Container Name	ArtiStatesSpinlockState
Parent Container	ArtiStates
Upstream mapping defined	–

▽

△

Description	Each state used by the OsSpinlock has to be listed as ArtiStatesSpinlockState parameter with a choice of the states.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiStatesSpinlockStateEnum	1	[ECUC_Arti_00210]

No Included Containers

⌋

[ECUC_Arti_00210] Definition of EcucEnumerationParamDef ArtiStatesSpinlockStateEnum ⌈

Parameter Name	ArtiStatesSpinlockStateEnum	
Parent Container	ArtiStatesSpinlockState	
Upstream mapping defined	–	
Description	ArtiStatesSpinlockState choice of the states.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ArtiSpinlockStateLocked	activated
	ArtiSpinlockStateUnlocked	activated
Pre-Compile only	true	

⌋

[ECUC_Arti_00030] Definition of EcucParamConfContainerDef ArtiStatesTaskState ⌈

Container Name	ArtiStatesTaskState
Parent Container	ArtiStates
Upstream mapping defined	–
Description	Each state used by the OS has to be listed as ArtiStatesTaskState Parameter with a choice of the states.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
ArtiStatesTaskStateEnum	1	[ECUC_Arti_00033]

No Included Containers

⌋

[ECUC_Arti_00033] Definition of EcucEnumerationParamDef ArtiStatesTaskStateEnum ⌈

Parameter Name	ArtiStatesTaskStateEnum	
Parent Container	ArtiStatesTaskState	
Upstream mapping defined	–	
Description	ArtiStatesTaskState choice of the states.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	ArtiTaskStateActivated	activated
	ArtiTaskStatePreempted	preempted
	ArtiTaskStateReady	ready
	ArtiTaskStateReleased	released
	ArtiTaskStateRunning	running
	ArtiTaskStateSuspended	suspended
	ArtiTaskStateWaiting	waiting
Pre-Compile only	true	

⌋

10.8 Published Information

For details refer to [4] Chapter 10.3 *“Published Information”*.

A Not applicable requirements

No content.

B Example

The example provided in this chapter demonstrates how to apply ARTI to an operating system. It also shows how the generic `ARTI_TRACE` macro can be mapped to different tracing implementations. In the example, these first tracing implementations is provided by `VENDOR_A` the second by `VENDOR_B`.

The C code of the example compiles but is not functional. The operating system is boiled down to three functions: `SuspendAllInterrupts`, `ResumeAllInterrupts` and `StartOS`. The application code defined the `main` function and two tasks: `Task_Cylinder0` and `Task_Cylinder1`.

Section B.1 holds all the C code demonstrating the ARTI instrumentation and section B.2 contains the corresponding ARXML code.

B.1 ARTI Instrumentation

B.1.1 ARTI Tool Binding (Arti.h)

```
#ifndef _ARTI_H_
#define _ARTI_H_

#include <stdint.h>

#if defined VENDOR_A
/* ARTI Trace Macro */
#   define ARTI_TRACE(_contextName, _className, _instanceName, instanceParameter, _eventName, event_value) \
        (void)TraceImpl ## _ ## _className ## _ ## _eventName ## _ ## _instanceName ## _ ## _contextName((instanceParameter), (event_value))

#elif defined VENDOR_B
/* ARTI Trace Macro */
#   define ARTI_TRACE(_contextName, _className, _instanceName, instanceParameter, _eventName, event_value) \
        (void)TraceImpl ## _ ## _className ## _ ## _contextName((_instanceName), (instanceParameter), (_eventName), (event_value))

#else
#   define ARTI_TRACE(_contextName, _className, _instanceName, instanceParameter, _eventName, event_value) (void)0
#endif


#endif /* _ARTI_H_ */
```

Listing B.1: Example for Arti.h

```
#ifndef _OS_ARTI_H_
#define _OS_ARTI_H_
```

```
3
4  #include "stdint.h"
5  #include "Arti.h"
6
7  #if defined VENDOR_A
8
9  /* Prototypes for AR_CP_OS_TASK */
10 void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_SPRVSR(uint32_t
       instanceParameter, uint32_t event_value);
11 void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_SPRVSR(uint32_t
       instanceParameter, uint32_t event_value);
12
13 void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_USER(uint32_t
       instanceParameter, uint32_t event_value);
14 void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_USER(uint32_t
       instanceParameter, uint32_t event_value);
15
16 void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_NOSUSP(uint32_t
       instanceParameter, uint32_t event_value);
17 void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_NOSUSP(uint32_t
       instanceParameter, uint32_t event_value);
18
19 #elif defined VENDOR_B
20
21 /* Prototypes for AR_CP_OS_TASK */
22 void TraceImpl_AR_CP_OS_TASK_SPRVSR(uint32_t instanceName, uint32_t
       instanceParameter, uint32_t eventName, uint32_t event_value);
23 void TraceImpl_AR_CP_OS_TASK_USER(uint32_t instanceName, uint32_t
       instanceParameter, uint32_t eventName, uint32_t event_value);
24 void TraceImpl_AR_CP_OS_TASK_NOSUSP(uint32_t instanceName, uint32_t
       instanceParameter, uint32_t eventName, uint32_t event_value);
25
26 #endif
27
28 #endif /* _OS_ARTI_H_ */
```

Listing B.2: Example for Os_Arti.h

```
1  #include <stdint.h>
2
3  #include "Os.h"
4  #include "Arti.h"
5  #include "Os_Arti.h"
6
7  /* Stubs for intrinsics */
8  #define __disable() ((void)(0))
9  #define __enable() ((void)(0))
10
11 #if defined VENDOR_A
12
13 void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_SPRVSR(uint32_t
       instanceParameter, uint32_t event_value)
14 {
15     __disable();
16     TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_NOSUSP(
           instanceParameter, event_value);
```

```
17      __enable();
18  }
19
20  void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_SPRVSR(uint32_t
        instanceParameter, uint32_t event_value)
21  {
22      __disable();
23      TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_NOSUSP(
            instanceParameter, event_value);
24      __enable();
25  }
26
27  void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_USER(uint32_t
        instanceParameter, uint32_t event_value)
28  {
29      SuspendAllInterrupts();
30      TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_NOSUSP(
            instanceParameter, event_value);
31      ResumeAllInterrupts();
32  }
33
34  void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_USER(uint32_t
        instanceParameter, uint32_t event_value)
35  {
36      SuspendAllInterrupts();
37      TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_NOSUSP(
            instanceParameter, event_value);
38      ResumeAllInterrupts();
39  }
40
41  void TraceImpl_AR_CP_OS_TASK_OsTask_Start_OS_SHORT_NAME_NOSUSP(uint32_t
        instanceParameter, uint32_t event_value)
42  {
43      (void)instanceParameter; // avoid warning "unused parameter"
44      (void)event_value; // avoid warning "unused parameter"
45
46      // actual tracing code goes here
47  }
48
49  void TraceImpl_AR_CP_OS_TASK_OsTask_Stop_OS_SHORT_NAME_NOSUSP(uint32_t
        instanceParameter, uint32_t event_value)
50  {
51      (void)instanceParameter; // avoid warning "unused parameter"
52      (void)event_value; // avoid warning "unused parameter"
53
54      // actual tracing code goes here
55  }
56
57  #elif defined VENDOR_B
58
59  void TraceImpl_AR_CP_OS_TASK_SPRVSR(uint32_t instanceName, uint32_t
        instanceParameter, uint32_t eventName, uint32_t event_value)
60  {
61      __disable();
62      TraceImpl_AR_CP_OS_TASK_NOSUSP(instanceName, instanceParameter,
            eventName, event_value);
```

```
63      __enable();
64  }
65
66  void TraceImpl_AR_CP_OS_TASK_USER(uint32_t instanceName, uint32_t
        instanceParameter, uint32_t eventName, uint32_t event_value)
67  {
68      SuspendAllInterrupts();
69      TraceImpl_AR_CP_OS_TASK_NOSUSP(instanceName, instanceParameter,
            eventName, event_value);
70      ResumeAllInterrupts();
71  }
72
73  void TraceImpl_AR_CP_OS_TASK_NOSUSP(uint32_t instanceName, uint32_t
        instanceParameter, uint32_t eventName, uint32_t event_value)
74  {
75      (void)instanceName; // avoid warning "unused parameter"
76      (void)instanceParameter; // avoid warning "unused parameter"
77      (void)eventName; // avoid warning "unused parameter"
78      (void)event_value; // avoid warning "unused parameter"
79
80      // actual tracing code goes here
81  }
82
83
84  #else
85
86  #endif
```

Listing B.3: Example for Arti.c

B.1.2 ARTI OS Instrumentation

```
1   #ifndef _OS_H_
2   #define _OS_H_
3
4   #define TASK(_taskname)      void OS_TASK ## _ ## _taskname(void)
5
6   void SuspendAllInterrupts(void);
7   void ResumeAllInterrupts(void);
8
9   void StartOS(void);
10
11  #endif
```

Listing B.4: Example for OS instrumentation header

```
1   #include "user_main.h"
2   #include "Arti.h"
3
4   void SuspendAllInterrupts(void)
5   {
6       // ...
7   }
8
9   void ResumeAllInterrupts(void)
10  {
```

```
    // ...
}

void StartOS(void)
{
    const int myCoreId = 0;
    const int OS_TASK_Task_Cylinder0_ID = 2;

    // for testing the ARTI interface, we call the task UserTask1 here
       directly (rather than implementing an OS)
    ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OS_SHORT_NAME, myCoreId,
       OsTask_Start, OS_TASK_Task_Cylinder0_ID);
    OS_TASK_Task_Cylinder0();
    ARTI_TRACE(NOSUSP, AR_CP_OS_TASK, OS_SHORT_NAME, myCoreId,
       OsTask_Stop,  OS_TASK_Task_Cylinder0_ID);
}
```

Listing B.5: Example for OS instrumentation source

B.1.3 ARTI User Code

```
#ifndef _USER_MAIN_H_
#define _USER_MAIN_H_

#include "os.h"
extern TASK(Task_Cylinder0);
extern TASK(Task_Cylinder1);

#endif
```

Listing B.6: Example for user code header

```
#include <stdlib.h>

#include "os.h"

TASK(Task_Cylinder0)
{
    // inject
}

TASK(Task_Cylinder1)
{
    // inject
}

int main(void)
{
    StartOS();

    exit(EXIT_SUCCESS);

    return -1;
}
```

Listing B.7: Example for user code source

B.2 ARXML Representation of Instrumentation

Example B.1

Example Values for the ArtiHook container for OsTask_Start

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiHook_ArtiOs_TaskStart</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiHook</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookClass</DEFINITION-REF>
      <VALUE>AR_CP_OS_TASK</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookEventName</DEFINITION-REF>
      <VALUE>OsTask_Start</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookInstance</DEFINITION-REF>
      <VALUE>OS_SHORT_NAME</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookEventParameterTypeRef</DEFINITION-
          REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          ArtiParameterTypeMap_TaskCylinderId</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiValues/ArtiHook/ArtiHookInstanceParameterTypeRef</DEFINITION
          -REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          ArtiParameterTypeMap_Core</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Example B.2

Example Values for the ArtiOsInstance container using the hooks

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiOsInstance_Conf</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiOs/ArtiOsInstance</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiOs/ArtiOsInstance/ArtiOsInstanceEcucRef</DEFINITION-REF>
```

```
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1EcucOs/
          Vendor1Os</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiOs/ArtiOsInstance/ArtiOsInstanceHookRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          ArtiHook_ArtiOs_TaskStart</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiOs/ArtiOsInstance/ArtiOsInstanceHookRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
          ArtiHook_ArtiOs_TaskStop</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Example B.3

Example Values for the ArtiHook container for arbitrary use

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiHook_IgnitionControl_Cyl0_IgnitionStart</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiValues/ArtiHook</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookClass</DEFINITION-REF>
      <VALUE>Ignition_Control</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookEventName</DEFINITION-REF>
      <VALUE>IgnitionStart</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiValues/ArtiHook/ArtiHookInstance</DEFINITION-REF>
      <VALUE>Cylinder0</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
</ECUC-CONTAINER-VALUE>
```

Example B.4

Example Values for an ArtiGenericComponentClass container with parameters holding hooks

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiGenericComponentClass_IgnitionControl</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiGeneric/ArtiGenericComponentClass</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
```

```
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
            /ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassName</DEFINITION-REF>
        <VALUE>ADIFFERENT</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
    </PARAMETER-VALUES>
    <SUB-CONTAINERS>
      <ECUC-CONTAINER-VALUE UUID="">
        <SHORT-NAME>IgnitionStart</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
            EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassParameter</DEFINITION-REF>
        <PARAMETER-VALUES>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentClass/
                ArtiGenericComponentClassParameter/
                ArtiGenericComponentClassParameterDescription</DEFINITION-
                REF>
            <VALUE>Ignition Start</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentClass/
                ArtiGenericComponentClassParameter/
                ArtiGenericComponentClassParameterName</DEFINITION-REF>
            <VALUE>IGNITION_START</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
      <ECUC-CONTAINER-VALUE UUID="">
        <SHORT-NAME>IgnitionStop</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
            EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentClass/
            ArtiGenericComponentClassParameter</DEFINITION-REF>
        <PARAMETER-VALUES>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentClass/
                ArtiGenericComponentClassParameter/
                ArtiGenericComponentClassParameterDescription</DEFINITION-
                REF>
            <VALUE>Ignition Stop</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
          <ECUC-TEXTUAL-PARAM-VALUE>
            <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentClass/
                ArtiGenericComponentClassParameter/
                ArtiGenericComponentClassParameterName</DEFINITION-REF>
            <VALUE>IGNITION_STOP</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
```

Example B.5

Example Values for an ArtiGenericComponentInstance container using the hooks

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiGenericComponentInstance_IgnitionCyl0</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Arti/ArtiGeneric/ArtiGenericComponentInstance</DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-TEXTUAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/EcucDefs/Arti
          /ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceName</DEFINITION-REF>
      <VALUE>Ignition Cylinder 0</VALUE>
    </ECUC-TEXTUAL-PARAM-VALUE>
  </PARAMETER-VALUES>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Arti/
          ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceClassRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1ArtiGeneric/
          ArtiGenericComponentClass_IgnitionControl</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>IgnitionCyl0Start</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance/
          ArtiGenericComponentInstanceParameter</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/
              Arti/ArtiGeneric/ArtiGenericComponentInstance/
              ArtiGenericComponentInstanceParameter/
              ArtiGenericComponentInstanceParameterClassParameterRef</
              DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/
              Vendor1ArtiGeneric/ArtiGenericComponentClass_IgnitionControl
              /IgnitionStart</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/
              Arti/ArtiGeneric/ArtiGenericComponentInstance/
              ArtiGenericComponentInstanceParameter/
              ArtiGenericComponentInstanceParameterHookRef</DEFINITION-REF
              >
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
              ArtiHook_IgnitionControl_Cyl0_IgnitionStart</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>IgnitionCyl0Stop</SHORT-NAME>
```

```
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
            EcucDefs/Arti/ArtiGeneric/ArtiGenericComponentInstance/
            ArtiGenericComponentInstanceParameter</DEFINITION-REF>
        <REFERENCE-VALUES>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentInstance/
                ArtiGenericComponentInstanceParameter/
                ArtiGenericComponentInstanceParameterClassParameterRef</
                DEFINITION-REF>
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/
                Vendor1ArtiGeneric/ArtiGenericComponentClass_IgnitionControl
                /IgnitionStop</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/
                Arti/ArtiGeneric/ArtiGenericComponentInstance/
                ArtiGenericComponentInstanceParameter/
                ArtiGenericComponentInstanceParameterHookRef</DEFINITION-REF
                >
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/Vendor1/Vendor1Arti/
                ArtiHook_IgnitionControl_Cyl0_IgnitionStop</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </ECUC-CONTAINER-VALUE>
    </SUB-CONTAINERS>
  </ECUC-CONTAINER-VALUE>
```

C Expression Syntax

This section describes the grammar of Expressions using the Extended Backus-Naur Form.

```
expression                = logical_OR_expression { '?' expression ':' expression } ;
logical_OR_expression     = logical_AND_expression { '||' logical_AND_expression } ;
logical_AND_expression    = inclusive_OR_expression { '&&' inclusive_OR_expression } ;
inclusive_OR_expression   = exclusive_OR_expression { '|' exclusive_OR_expression } ;
exclusive_OR_expression   = AND_expression { '^' AND_expression } ;
AND_expression            = equality_expression { '&' equality_expression } ;
equality_expression       = relational_expression { ( '==' | '!=' ) relational_expression } ;
relational_expression     = shift_expression { ( '<' | '>' | '<=' | '>=' ) shift_expression } ;
shift_expression          = additive_expression { ( '<<' | '>>' ) additive_expression } ;
additive_expression       = multiplicative_expression { ( '+' | '-' ) multiplicative_expression } ;
multiplicative_expression = cast_expression { ( '*' | '/' | '%' ) cast_expression } ;
cast_expression           = { '(' type_name ')' } unary_expression ;
unary_expression          = postfix_expression | unary_operator cast_expression | 'sizeof' unary_expression | 'sizeof' '(' type_name ')' ;
unary_operator            = '&' | '*' | '+' | '-' | '~' | '!' ;
postfix_expression        = primary_expression { '[' expression ']' | ( '.' | '->' ) appl_identifier } ;
primary_expression        = appl_identifier | constant | '(' expression ')' ;
constant                  = integer_constant | character_constant | floating_constant | enumeration_constant ;
type_name                 = type_specifier { type_specifier } [ '*' ] ;
type_specifier            = 'void' | 'char' | 'short' | 'int' | 'long' | 'float' | 'double' | 'signed' | 'unsigned' | type_def_name ;
type_def_name             = appl_identifier;
```

Where:

integer_constant	represents an integer number, where the standard C convention is used for decimal, hexadecimal and octal notation.
character_constant	follows the C definition for a character, including the support of all standard escape sequences, such as '\n', '\t' etc.
floating_constant	follows the C definition for a floating point number.
enumeration_constant	follows the C definition for an "enum" constant.
appl_identifier	represents any C identifier and represents application symbols. These symbols rely on symbolic information retrieved from the debug information of the application and must have 'external linkage' scope (e.g. global C variables). The symbol value is only valid after the application has executed its initialization phase (typically this is the system startup code before reaching the applications entry point, which is main() in C). The only exception to this constraint is when using the unary address-operator (&).

Table C.1: Base types of the grammar

Further rules:

- Whitespace (blank, TAB) between terminals is ignored.
- All keywords and identifiers are case-sensitive.

D Change history of AUTOSAR traceable items

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

D.1.1 Added Specification Items in R22-11

Number	Heading
[SWS_Arti_00011]	Generate RTE VFB Trace Client
[SWS_Arti_00012]	Define AR_CP_RTE_RUNNABLE
[SWS_Arti_00013]	Define AR_CP_SCHM_SCHEDULABLE
[SWS_Arti_00014]	Macro AR_CP_RTE_RUNNABLE
[SWS_Arti_00015]	Macro AR_CP_SCHM_SCHEDULABLE
[SWS_Arti_00016]	Macro AR_CP_VOID
[SWS_Arti_00017]	ARTI Tracing Macro Parameters
[SWS_Arti_00018]	ARTI Tracing Macro Format (single event)
[SWS_Arti_00019]	ARTI Tracing Macro Format (multiple events)
[SWS_Arti_00020]	Define AR_CP_RTE_API
[SWS_Arti_00021]	Macro AR_CP_RTE_API
[SWS_Arti_00022]	Define AR_CP_BSW_API
[SWS_Arti_00023]	Macro AR_CP_BSW_API

Table D.1: Added Specification Items in R22-11

D.1.2 Changed Specification Items in R22-11

Number	Heading
[SWS_Arti_00009]	
[SWS_Arti_00010]	
[SWS_Arti_91002]	
[SWS_Arti_91004]	
[SWS_Arti_91005]	

Table D.2: Changed Specification Items in R22-11

D.1.3 Deleted Specification Items in R22-11

none

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

D.2.1 Added Specification Items in R23-11

Number	Heading
[SWS_Arti_91000]	Definition of datatype ARTI_STOPWATCH_FLAT, ARTI_STOPWATCH_NESTED
[SWS_Arti_91006]	Definition of imported datatypes of module Arti

Table D.3: Added Specification Items in R23-11

D.2.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Arti_00002]	Macro USER_STOPWATCH

Table D.4: Changed Specification Items in R23-11

D.2.3 Deleted Specification Items in R23-11

none

D.3 Traceable item history of this document according to AUTOSAR Release R24-11

D.3.1 Added Specification Items in R24-11

Number	Heading
[ECUC_Arti_00205]	Definition of EcucParamConfContainerDef ArtiStatesScheduleTableState
[ECUC_Arti_00206]	Definition of EcucEnumerationParamDef ArtiStatesScheduleTableState Enum
[ECUC_Arti_00207]	Definition of EcucParamConfContainerDef ArtiStatesSpinlockOwnerType
[ECUC_Arti_00208]	Definition of EcucEnumerationParamDef ArtiStatesSpinlockOwnerType Enum
[ECUC_Arti_00209]	Definition of EcucParamConfContainerDef ArtiStatesSpinlockState
[ECUC_Arti_00210]	Definition of EcucEnumerationParamDef ArtiStatesSpinlockStateEnum

Table D.5: Added Specification Items in R24-11

D.3.2 Changed Specification Items in R24-11

Number	Heading
[ECUC_Arti_00027]	Definition of EcucChoiceReferenceDef ArtiParameterTypeMapPairOutputRef
[ECUC_Arti_00029]	Definition of EcucParamConfContainerDef ArtiStates
[ECUC_Arti_00177]	Definition of EcucParamConfContainerDef ArtiRteVfbTraceHooks

Table D.6: Changed Specification Items in R24-11

D.3.3 Deleted Specification Items in R24-11

none

D.4 Traceable item history of this document according to AUTOSAR Release R25-11

D.4.1 Added Specification Items in R25-11

none

D.4.2 Changed Specification Items in R25-11

none

D.4.3 Deleted Specification Items in R25-11

none

D.5 Traceable item history of this document according to AUTOSAR Release R26-11

D.5.1 Added Specification Items in R26-11

none

D.5.2 Changed Specification Items in R26-11

none

D.5.3 Deleted Specification Items in R26-11

Number	Heading
[SWS_Arti_00010]	
[SWS_Arti_91005]	Definition of API function Arti_GetVersionInfo

Table D.7: Deleted Specification Items in R26-11