Document Title	Explanatory Document for usage of AUTOSAR RunTimeInterface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	896

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

Document Change History			
Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Updated ARTI macro example code • Updated ARXML examples • Minor corrections and updates
2022-11-24	R22-11	AUTOSAR Release Management	• Deprecated compiler abstraction • Minor corrections and updates
2021-11-25	R21-11	AUTOSAR Release Management	• Added examples showing static debugging, CAT1 interrupts, and VFB Hooks • Minor corrections and updates
2020-11-30	R20-11	AUTOSAR Release Management	• Introduced chapter Example Implementations • Updated chapter Example Configuration with ECUC changes
2019-11-28	R19-11	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.

Contents

1 Introduction 5

1.1 Who should read this document? 5
1.2 Objectives 5
1.3 Summary of use-cases 6

2 Run-Time Interface 7

2.1 Trace Techniques 7
2.1.1 Hardware Tracing 7
2.1.2 Software Tracing 8
2.2 Static Debugging 8
2.3 OS Awareness 8
2.4 ARTI at a glance 9

3 OS 12

3.1 State Model 12
3.2 Hooks 13
3.3 ECUC 14

4 RTE 18

4.1 ECU Configuration 18
4.2 BSW Module Description Template 19
4.3 Hooks 22

5 Comparison between ARTI and ORTI 24

5.1 Mapping from ORTI to ARTI 24
5.2 Mapping Vendor Specific Objects and Attributes 26

6 Example Configurations 27

6.1 Static Debugging 27
6.2 OS Task Tracing 31

7 Example Implementations 37

7.1 Example Hook Implementation for Hardware Tracing 37
7.1.1 Common ARTI Header File 38
7.1.2 OS Instrumentation 39
7.1.3 RTE Instrumentation 44
7.2 Example for User Provided CAT1 Interrupt 46
7.2.1 Instrumenting the CAT1 interrupt with ARTI Hooks 47
7.2.2 EcuC representation of CAT1 ARTI Hooks 48

8 Outlook 50

9 Document Information 51

9.1 Related documentation 51
9.1.1 Input documents & related standards and norms 51

9.1.2 Related specification 51

1 Introduction

1.1 Who should read this document?

This EXPlanatory document is intended to describe the steps which are necessary for OS Vendors, RTE Vendors, Debugger Vendors and Timing Tool Vendors to implement the necessary parts to support the AUTOSAR Run-Time Interface Software Specification.

1.2 Objectives

ARTI is a set of standards for debugging and tracing the run-time behavior of embedded systems. Its origin is in the automotive sector, specifically as a concept document developed for the AUTOSAR development partnership, but its scope is not limited to purely automotive systems.

ARTI aims to make it possible for tools from multiple different vendors to collect and exchange runtime data from embedded systems in a standardized way, and hence promote competition and innovation.

ARTI describes interfaces needed to support (a) **static debugging** and (b) **dynamic tracing**.

Static debugging typically involves having an in-circuit debugger connected to the embedded system. Whenever the debugger halts the execution of the system, you can inspect the system's state (registers, stack and data). Decoding the meaning of the state is not necessarily straightforward in any medium or large scale system. ARTI allows the software be described in terms of its architectural concepts and components, so that the debugger can display a much more meaningful representation of the system state. One example is that the debugger could show a list of the tasks in the system along with their state, priority and execution time. It could also show other parts of the system, such as the inter-task messages and their values.

Dynamic tracing on the other hand operates with the embedded system running at normal speed and without interruption. The system records the points in time at which specific events occur and passes this information on to some analysis tool or viewer. As before, ARTI allows these system events to be described so that the analysis tool - viewer can interpret them in terms of architectural concepts and components. Views can be constructed showing the execution pattern of tasks, and statistics based on response times and execution times can be calculated. Dynamic tracing can be achieved with minimal or zero instrumentation of the code where an ARTI compatible in-circuit debugger is available.

The name ARTI was chosen as a nod to a previous automotive standard called ORTI. Whereas ORTI was specified focusing just on debugging embedded operating system, ARTI is intended to be capable of bringing debugging and tracing to all layers of the

software stack. This document first describes the motivation and reasoning for the ARTI specifications without going into the technical details. These are added in later sections.

1.3 Summary of use-cases

The following two use-cases should provide an idea of what is expected by the AUTOSAR Run-Time Interface to enable AUTOSAR aware debugging.

Static debugging A typical use-case is the debugging and stepping through the code execution during integration phase of an ECU. For the analysis of a failure in application software code it is always helpful to understand the OS context, meaning in which task has the error occurred.

Dynamic tracing Continuous testing ECUs on timing of the real time software is not always the standard during software development. Most of the time resource consumption is only measured statically (e.g. memory consumption). But typically some tests, especially affected by stimulating the software with the car environment sporadically fail without no obvious reason. If shifts in the scheduling and misses in timing deadlines are not considered, this source of failure can not be determined. Tracing the dynamic run-time behavior of the software architecture is therefore an important use-case for the AUTOSAR AutosarRunTimeInterface.

2 Run-Time Interface

Capturing the software architecture's dynamic run-time behavior in the field is not easy. The effort to be spend to evaluate simple performance characteristics like CPU load or timing metrics can be tremendously high, depending on the project. There are usually many parts in the tool chain involved which need to work togehter very well in order to provide meaningful and correct results.

In order to explain how the AUTOSAR Run-Time Interface is trying to overcome those generally described problems it is necessary to have a look at the techniques which need to be used to get to the intended result.

2.1 Trace Techniques

For tools to capture the software's behavior at run-time, there are many ways to do so. Each one of them has its advantages and disadvantages. But basically to capture for a longer period the software's dynamic aspects, resources are necessary. Either ECU resources such as processing time or silicon resources such as trace interface and processor pins or even bus resources. The following sub chapters should provide a brief insight into the different trace techniques and therefore an understanding, why ARTI is approached in the way it is introduced later on.

2.1.1 Hardware Tracing

In hardware tracing the main challenge is to lift the information from hardware to system level. Meaning to interpret assembly and low level instructions as execution of an OS Task or any other configuration and software architecture relevant artifacts. There are generally speaking two ways of tracing techniques with some minor silicon vendor and implementation specific differences. Program Flow Tracing (Instruction Trace) and Data Tracing.

Program Flow Tracing: with this approach assembly instructions such as jump, branch or other (vendor dependent) set of instructions create an trace event which is multiplexed and stored on the on-chip buffer or directly streamed off the target. This technique provides a fine granular insight in the code executed on the target. This comes of course at the cost of high bandwidth interfaces or limited measurement time in the on chip case. Important for the ARTI approach is to know: with instruction trace the OS internal state variables can not be seen to the trace unit and therefore the states of Tasks and ISRs can not be derived.

Data Tracing: with the data trace approach data read and write access to most of the memory areas can create a trace event message and export the operation without run-time overhead to the on chip trace buffer or output stream. If the OS internal variables are known, with this approach Task and ISR states can be exported. In a modern AUTOSAR systems there is no state variable or other tracking mechanisms for

Runnables. Therefore, observing Runnables by means of data tracing write accesses to global data object is not natively supported by the RTE. Adding manually a variable to be used in the VFB Hooks for example, only for means of data tracing is often referred to as instrumentation. With this approach trace events are created only by scheduling behavior at run-time.

2.1.2 Software Tracing

Hardware tracing requires that the microcontroller is equipped with an on-chip trace unit/logic. With software based tracing the events can be captured and stored on target in the on-chip memory. Usually in CPU idle, the data can be send off the ECU via the connected bus interfaces, like CAN or Ethernet. The approach is therefore quite similar to data tracing with instrumentation with the difference, that the interface to send the information off the chip is most probably a bus instead of a debug interface.

2.2 Static Debugging

One very important use-case for ECU development and integration is the debugging of the embedded software. For users to debug the software, information about the current running Task or ISR can be pretty helpful. Therefore, the ARTI file provides information to the debugger about the relevant OS internal state variables.

2.3 OS Awareness

One already successful attempt to standardize the access for tracing tools to the OS internals for the OSEK platform is represented by the ORTI (OSEK Run-Time Interface) standard. Information about the Task Stack, Current Running Task, Current Running ISR and the knowledge to decode the Task State from the OS internals was stored in an .ORTI file. For AUTOSAR and especially multi-core projects the standard is not fitting anymore. With the merge from single-core OS to multi-core OS the tracking of Task and ISR states became more complex. Expressions, Pointer handling and so on are difficult to trace during the code execution. Additionally the ORTI standard is not aware about RTE features. But most importantly: Software based tracing is not standardized within ORTI.

The objective of ARTI is to extend this OS awarenss and make the debug, trace and other run-time tools aware of additional AUTOSAR modules such as RTE and SchM.

2.4 ARTI at a glance

The general idea of ARTI is more than to achieve easy "OS Awareness" for debuggers. Starting from a common understanding about scheduling state machines, with a common exchange format for debug and trace configuration. Additionally, exchanging run-time measurements up to their interpretation affects the range of the AUTOSAR/ASAM Run-Time Interface. One of the ARTI goals is to achieve a standard to configure, gather and process as well as to evaluate vendor independent AUTOSAR projects run-time behavior. To achieve this workflow and standardization aims, also a trace exchange format has to be standardized as well as common timing parameters. The Timing Parameters and the trace format is covered by the ARTI ASAM standard.

The following list describes the necessary steps to be taken for an ARTI workflow and their artifact files.

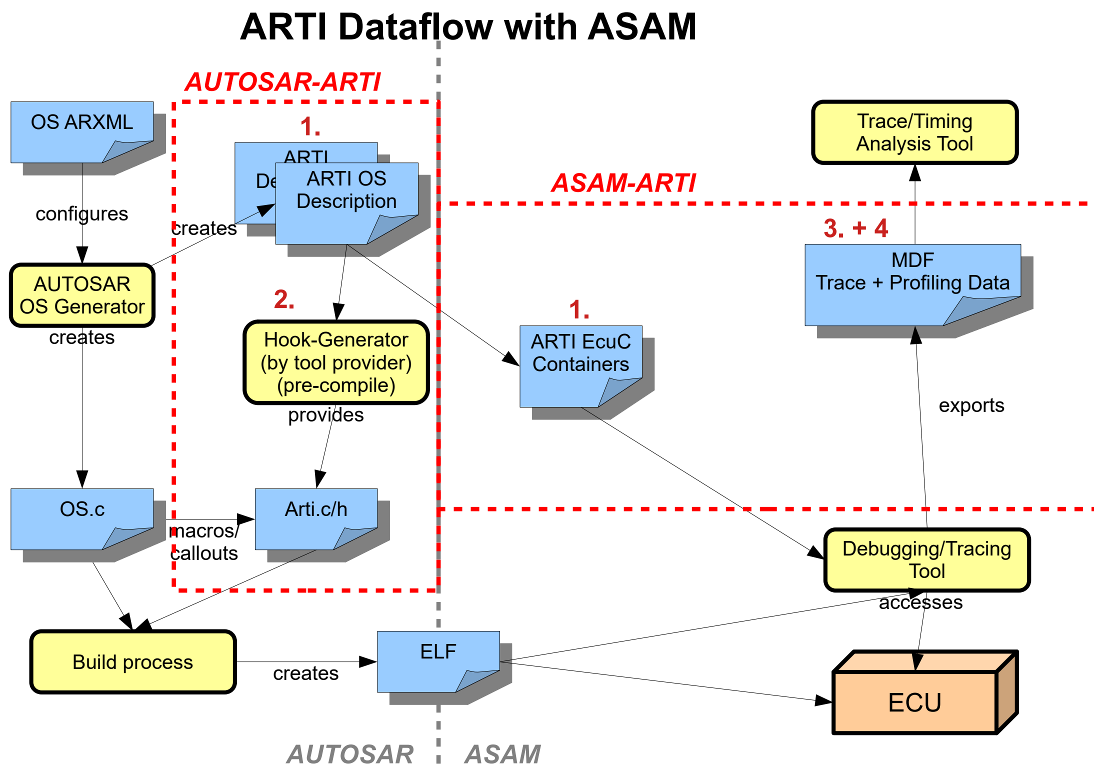

Figure 2.1: ARTI Dataflow

1. **ECU Configuration (ARXML) - AUTOSAR**

   The ARTI ECU Configuration Parameters containers fulfill rather two purposes. For once they store the Trace/Debug Configuration of the AUTOSAR project. Aside of that, the gathering of all ARTI containers replaces the information provided by an ORTI file.

   Currently there are four ARTI ECU Configuration Parameter Containers available within the ARTI ECUC Module Definition: ArtiValues, ArtiGeneric, ArtiHardware and ArtiOs. Depending on the use-case a different set of Parameter Container needs to be configured.

   ArtiValues The ARTI ECUC Container takes care about storing all actual trace and debug information. It is necessary for all ARTI use-cases. It collects the names of all ARTI relevant variables, f.e. the layout of the OS Hooks with a TypeMap to map Task and ISR Ids to the names, or the task state expressions for static debugging, which are referenced from ArtiHardware.

   ArtiOs The ARTI OS container stores basically the OS configuration with a view for tracing and debug tools. It describes mainly all available Tasks and ISRs. Additionally it defines which debug or trace feature is enabled for the referenced OS configuration, while the ARTI container sums up which ARTI hooks, variables and so on are available in the project in total.

   ArtiHardware The ARTI Hardware container stores all references for the currently running Task and ISR OS variables for each core, while the actual variable is stored in the ARTI component. This container is only necessary for static debugging and establishes the connection between CurrentRunningTask, CurrentRunningIsr and the ECU core.

   ArtiGeneric The ARTI Generic container provides the possibility for ARTI OS and RTE vendors to add additional information to the ARTI files, which is not standardized. It can be used to store the start address of an Task for example. The ArtiGeneric container is not mandatory to be used in any use-case.

2. **Hook Generator - AUTOSAR**

   After configuration of the AUTOSAR project, the tracing tool vendor specific hook implementations needs to be generated. Intentionally the trace tool reads therefore the used ARTI ECUC files (split ECUC or merged) and generates out of them the C-code implementation of the ARTI hooks. After adding them to the build process the project is able to be compiled. Background of that workflow is mainly, that the hook macros can be expanded to void and therefore be switched on and off after configuration.

3. **Trace Format (MDF) - ASAM**

   After compilation of the whole project the AUTOSAR Run-Time Interface part of standardization basically ends. The customer is now able to debug or trace the project with a view on OS run-time parameters. During the recording of the Task and ISR transitions signalized by the OS hooks, the tracing vendor stores the timestamp and scheduling event which has taken place. The set of scheduling transitions is defined by the OS hooks and signalize a state transition in one of the state diagrams for the affected scheduling entity (Task, ISR).

   The trace exchange format basically stores the scheduling events with a timestamp and additionally stores the information which scheduling entity is affected and which state diagram is to be used to calculate the possible timing parameters. As exchange format itself the well known ASAM MDF (Measurement Data Format) is used for many reasons: First of all, it can store huge data amounts efficiently, it is well known in the industry as well as it stores data and the description of the data at the same time.

4. **Timing Parameters (MDF) - ASAM**

   After the tracing data is available, the information of interest can be derived. The ASAM Run-Time Interface is therefore focusing on the topic of how interpret the data a set of metrics. With such an workflow the standardization approach should help to cover finding run-time issues in AUTOSAR projects. The timing parameters are intended to be storable in the MDF file, along with the trace data.

The artifact files and the dataflow for one ARTI aware project with both standardization approaches can be seen in figure 2.1.

3 OS

To achieve a vendor independent interpretation of the run-time behavior of an AUTOSAR ECU, ARTI defines state machines. The transitions between those states signalize the OS scheduling at run-time. In other words the state transitions can be understood as the implementation of the OS Hooks. Each time an OS Task is preempted by an ISR for example, the ARTI Hook macro signalizes the change in the OS Task's state transition. This has two advantages: Software based tracing can be used as well as data tracing and the result is the same.

3.1 State Model

The OS State Model, described in the SWS OS is supported by ARTI. But for a proper view with a timing tool, additional state information is necessary. Since there is no distinction between a preempted, released and activated task state an enhanced state machine is introduced. Depending on the features provided by the OS vendor, either the standard OS model can be used or the enhanced one for detailed analysis. To distinguish both state diagrams, the hook macro and therefore the available state transitions shall go by a different state machine description (Class Name). The ARTI Class name **AR_CP_OS_TASK** is to be used for the standard OS Task states and **AR_CP_OSARTI_TASK** for the enhanced state machine.

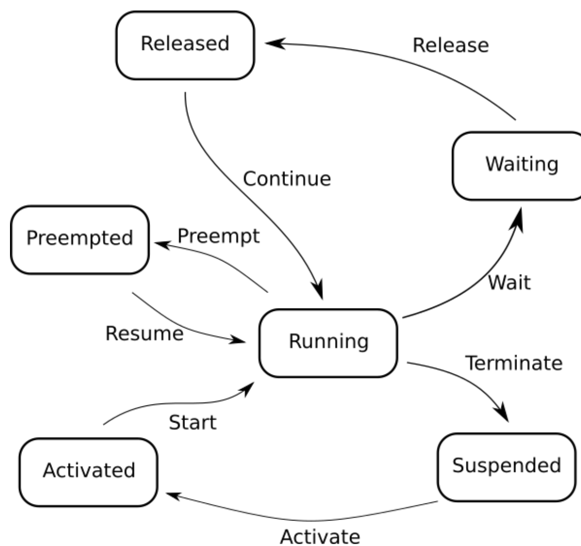

Figure 3.1: Enhanced Task State Diagram AR_CP_OSARTI_TASK

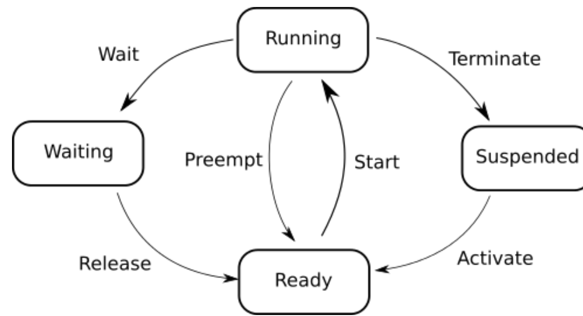

Figure 3.2: Standard Task State Diagram `AR_CP_OS_TASK`

3.2 Hooks

The ARTI hook macros are intended to signalize the state transitions of a schedulable entity. By intention, they are designed to be macros rather than functions to be able to minimize the overhead added in computing. The usage of the ARTI Hook can be switched off in the affected BSW Module. For example, all OS ARTI macros are switched on with the define **`OS_USE_ARTI`**, which implements the OS configuration parameter `OsUseArti` as defined in [1].

```
1 #ifdef OS_USE_ARTI
2 #   include "Os_Arti.h"
3 #else
4 #   define ARTI_TRACE(_contextName, _className, _instanceName,
      _eventName, instanceParameter, eventParameter) ((void)0)
5 #endif
```

Listing 3.1: Enable or disable ARTI with one define

The layout of the ARTI Macro is quite generic. The main idea therefore is that users can also define their own macros. However, this approach is currently not fully standardized. For all standardized scheduling state machines, the **`_className`** maps the macro to the state machine, while **`_eventName`** describes the state transition. The **`_contextName`** should describe whether Interrupts are disabled or not, during the reading of the macro data while execution. Three different modes are possible, **`_USER`** which indicates that the hook implementer can not disable interrupts and needs to provide correcting postprocessing in case of an interruption. **`_NOSUSP`** indicates that the macro will be executed in an context where interrupts are locked. **`_SPRVSR`** indicates that the hook makro can disable the interrupts itself. The **`_instanceName`** should give information to which OS (name) the Task belongs to.

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

Listing 3.2: Preprocessor conversion example for ARTI macros

The **instanceParameter** and **eventParameter** are not literals such as all other macro parts **(_....Name)**. The **instanceParameter** is used to handle the CoreId parameter, while the TaskId can be accessed via the **eventParameter**.

```
1 ARTI_TRACE(NOSUSP, AR_CP_OSARTI_TASK, OS, OsTask_Wait, CoreId, TaskId);
```

Listing 3.3: ARTI trace macro for OS Task Wait

At compile time, the preprocessor will replace the generic macro with all specific ARTI macros, if they have been implemented with an tracing vendor specific instrumentation. The double underscore should help to parse the macros easier.

```
1 #define ARTI_TRACE_NOSUSP_AR_CP_OSARTI_TASK_OS_OsTask_Wait(CoreId, TaskId
    ) {;}
```

Listing 3.4: ARTI trace macro implementation for OS Task Wait

3.3 ECUC

The following section focuses on the ECUC representation of ARTI for OS tracing use-case. To configure and store the ARTI configuration for the hook based tracing, the following ECUC containers are necessary: `ArtiValues` and `ArtiOs`. The `ArtiValues` container stores the Id mapping for Tasks and ISRs as well as all available OS hook macros (Figure 3.3). The `ArtiOs` container stores all configured OS Tasks and references all enabled hook macros in the `ArtiOsInstance` EcucParamConfContainer (see Figure 3.4).

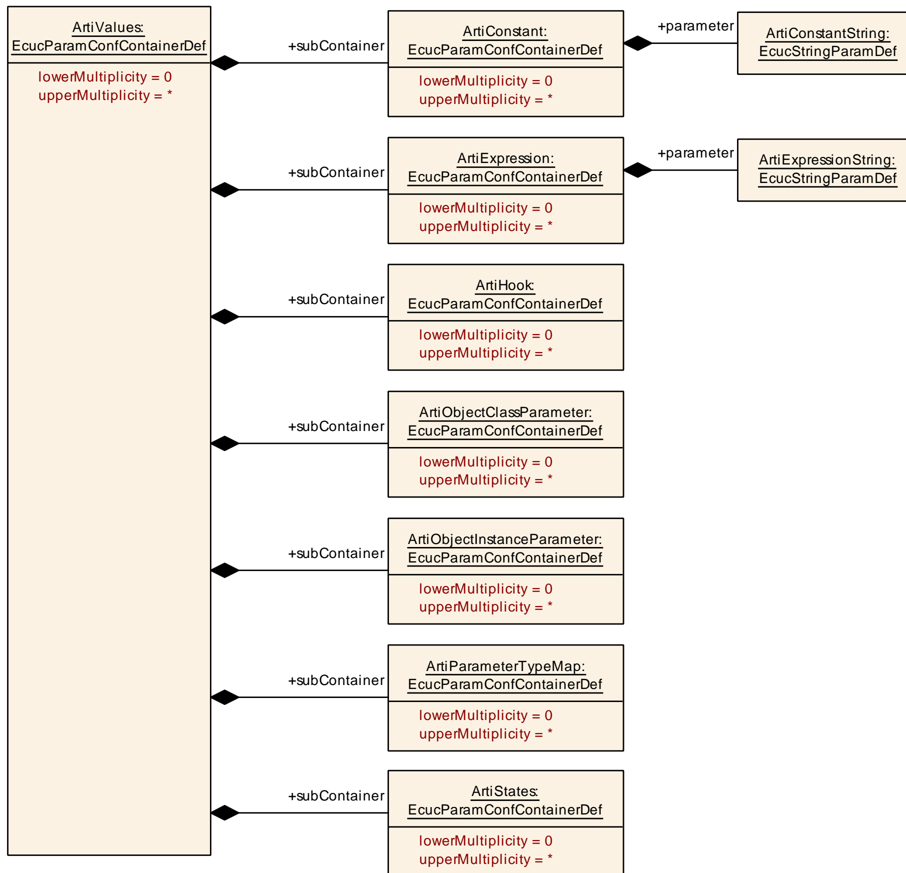

Figure 3.3: ArtiValues Ecuc Module Definition Class Diagram

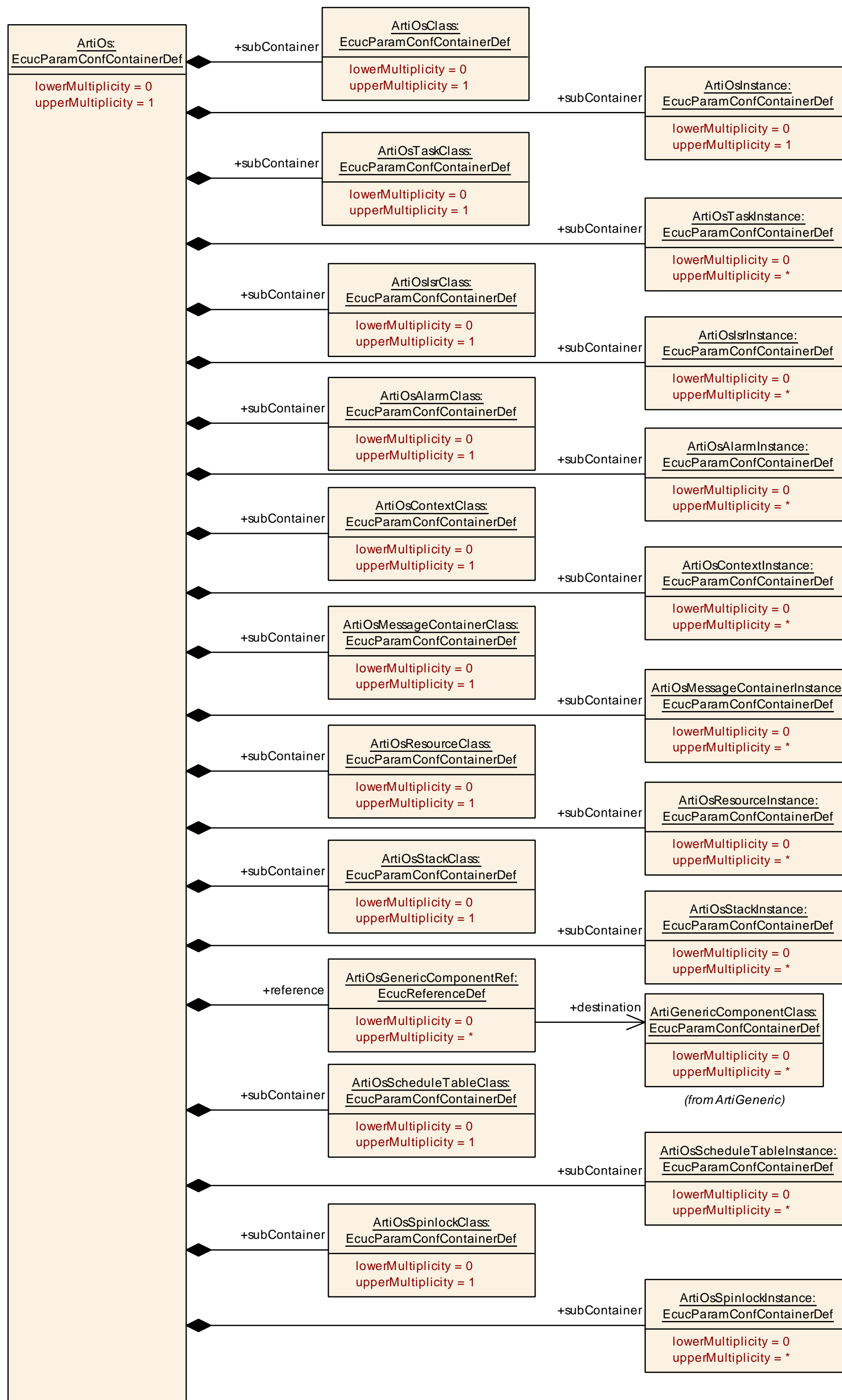

Figure 3.4: ArtiOs Ecuc Module Definition Class Diagram

Examples for this use-case are available for the `Arti` EcucParamConfContainer in Listing 6.4 and for the `ArtiOs` EcucParamConfContainer in Listing 6.5.

class ArtiValues	
ECUC Container Value	**Note**
ArtiConstant	This container holds a constant value.
ArtiExpression	Contains the OS expression string such as Task State expression. This C like expression can be evaluated by a debugger and is similar to what is already done in ORTI.
ArtiHook	Model representation of the ARTI Hook macros
ArtiObjectClassParameter	Describes the properties of an ARTI object, such as the CurrentRunningTask needs to reference the TypeMap to evaluate the ID or expression
ArtiObjectInstanceParameter	Instantiates an ARTI object, like CurrentRunningTask variable for Core 0
ArtiParameterTypeMap	Contains a mapping for translating the numerical ID of an ARTI object, e.g., of a task to its literatal name
ArtiStates	This container contains all states of tasks, isrs... that the EcuC uses

Table 3.1: ArtiValues Class ECUC Container Values

class ArtiOs	
ECUC Container Value	**Use-Case**
ArtiOsClass	
ArtiOsInstance	OS Expressions, such as task state
ArtiOsTaskClass	Model representation of the ARTI Hook macro for a task
ArtiOsTaskInstance	Instantiates a task with reference to the OS
ArtiOsIsrClass	Model representation of an ARTI Hook macro for an Isr
ArtiOsIsrInstance	Instantiates an Isr with reference to the OS
ArtiOsAlarmClass	Model representation of the ARTI Hook macro for an alarm
ArtiOsAlarmInstance	Instantiates an alarm with reference to the OS
ArtiOsContextClass	Model representation of the ARTI Hook macro for a context
ArtiOsContextInstance	Instantiates a context with reference to the OS
ArtiOsMessageContainerClass	Model representation of the ARTI Hook macro for a message container
ArtiOsMessageContainerInstance	Instantiates a message container with reference to the OS
ArtiOsResourceClass	Model representation of the ARTI Hook macro for a resource
ArtiOsResourceInstance	Instantiates a resource with reference to the OS
ArtiOsStackClass	Model representation of the ARTI Hook macro for a stack
ArtiOsStackInstance	Instantiates a stack with reference to the OS

Table 3.2: ArtiOs Class ECUC Container Values

4 RTE

ARTI uses the VFB Tracing Hooks as an own trace client. A detailed description on how to configure an ARTI compliant RTE specification is available in Chapter 7.4 of [2, SWS ARTI].

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

4.1 ECU Configuration

ARTI creates a VFB trace client called `Arti` and provides the configuration for the trace client using RTE's ECU Configuration Container `RteVfbTraceClient`. Within this container, a *wish* list of hook functions to be traced is generated using the EcuC Parameter `RteVfbTraceFunction`. Thereby, the following use cases can be distinguished:

- to enable trace of all `BswSchedulableEntity` hooks, `RteVfbTraceFunction` contains `Rte_Arti_SchM`
- to enable trace of all `RunnableEntity` hooks, `RteVfbTraceFunction` contains `Rte_Arti_Runnable`
- to enable trace of all `RunnableEntity` hooks of a specific component, `RteVfbTraceFunction` contains `Rte_Arti_Runnable_MyComponentType`, where `MyComponentType` is referring to a `RteSwComponentType` EcuC Container.
- to enable trace hooks of a specific `RunnableEntity` within a specific component, `RteVfbTraceFunction` contains `Rte_Arti_Runnable_MyComponentType_MyRunnable`, where `MyComponentType` is referring to a `RteSwComponentType` EcuC Container and `MyRunnable` to a `RunnableEntity` in the SWComponentTemplate.

Example 4.1

This example shows how the configuration of trace hook generation for all `RunnableEntity`s of a specific component looks like. To do so, the EcuC must contain a representation of the `SwComponentPrototype` that is located on the configured ECU. This is done with the ECU Container `RteSwComponentInstance` as part of the module configuration of the Rte.

```
...
<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>
```

```
...
```

The following listing shows, then, the part of the EcuC that enables the trace hook generation of all runnables of `ConsumerComponent`, the software component instance introduced above.

```
...
<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>
...
```

4.2 BSW Module Description Template

Based on the configuration, the RTE generator creates the `BswModuleEntry`s containing the trace hooks. The `functionPrototypeEmitter` entry `Arti` identifies them as ARTI hooks. ARTI generates, then, the final trace client based on the `BswModuleEntry`'s for the ARTI trace client. This results in the following changes:

- expansion of the `BswInternalBehavior` by a `BswCalledEntity` for each hook and assigning a common memory section to its code object via the referenced `SwAddrMethod`
- set of the correct implementation policy (`MACRO`, `INLINE`, or `STANDARD`) for the `BswModuleEntry` of each hook.
- addition of the header file with the function declarations of the hooks as a required artifact to the `BswImplementation`
- specification of the `ResourceConsumption` in the `BswImplementation` by adding a `MemorySection` that contains references to all `BswCalledEntity`s of the hooks as well as a prefix for the memory section's namespace in the code via `SectionNamePrefix`.

Example 4.2

Example 4.1 started off with the ECU Configuration to enable the trace hook generation of all runnables the software component instance `ConsumerComponent`. This example continues now and presents in detail the subsequent changes to the BSW Module Description Template. As mentioned above, a `BswModuleEntry` is created for each trace hook. In this example, the software component instance `ConsumerComponent` contains a runnable called `RE2`. ARTI consequently generates a start and return hook for this runnable which assembles to `Rte_Arti_Runnable_ConsumerComponent_RE2_Start` and `Rte_Arti_Runnable_ConsumerComponent_RE2_Return`, respectively:

```
...
<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>
<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>
...
```

Besides the `BswModuleEntry`s, also `BswCalledEntity`s are created by ARTI, which reflects in the following way:

```
...
<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>
...
```

Next, the generated header file (`Rte_Hook_Arti.h`) with the function declarations of the hooks are added, which looks, then, like this:

```
...
<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>
...
```

Finally, the according memory section for these hooks is defined:

```
...
<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_Start
</EXECUTABLE-ENTITY-REF>
<EXECUTABLE-ENTITY-REF DEST="BSW-CALLED-ENTITY" BASE="
    Rte_BSWMD_BswModuleDescriptions">
Rte/RteInternalBehavior/Rte_Arti_Runnable_ConsumerComponent_RE2_Return
</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>
...
```

4.3 Hooks

As already mentioned above, ARTI also generates the header file containing definitions for VFB tracing. These functions are, then, used for mapping the VFB trace hooks to the `ARTI_TRACE` macro. A possible implementation of the `ARTI_TRACE` macro will be discussed in detail later in Section 7.1.3.

Example 4.3

Again, the example started with Example 4.1 and continued in Example 4.2 is picked up. The following listing shows the header file containing the generated function definitions of the start and return hooks `Rte_Arti_Runnable_ConsumerComponent_RE2_Start` and `Rte_Arti_Runnable_ConsumerComponent_RE2_Return`, respectively for the runnable `RE2` of the software component instance `CustomerComponent`.

```
1  /**********************************************************************
2   * -------------------------------------------------------------------
3   * FILE DESCRIPTION
4   * -------------------------------------------------------------------
5   * File: Rte_Hook_Arti.h
6   *
7   * Description: Header file containing definitions for VFB tracing
8   **********************************************************************/
9
10
11 /*********************************************************************
12  * Names of available VFB-Trace-Hooks
13  *********************************************************************
14  *
15  * Configured:
16  *
17  * Rte_Arti_Runnable_ConsumerComponent_RE2_Start
18  * Rte_Arti_Runnable_ConsumerComponent_RE2_Return
19  *
20  * Not configured:
21  *
22  *
23  **********************************************************************/
24
25  /* double include prevention */
26  #ifndef _RTE_HOOK_H
27  # define _RTE_HOOK_H
28
29  # include "Os.h"
30
31  # include "Rte_Type.h"
32  # include "Rte_Cfg.h"
33
34  void Rte_Arti_Runnable_ConsumerComponent_RE2_Start(void);
35
36  void Rte_Arti_Runnable_ConsumerComponent_RE2_Return(void);
37
38  #endif /* _RTE_HOOK_H */
39
```

```
40  /*
       *********************************************************************
41   * END OF FILE: Rte_Hook_Arti.h
42   *********************************************************************/
```

Listing 4.1: Example function declarations of VFB hooks

All that is left now it to use the ARTI macro within the generated VFB Tracing Hooks. This is examplarily shown in Listing 4.2, where the start and return hooks of the runnable, which is assigned the ID 1, are implemented.

```
1  /*********************************************************************
2   * ------------------------------------------------------------------
3   * FILE DESCRIPTION
4   * ------------------------------------------------------------------
5   * File: Rte_Hook_Arti.c
6   *
7   * Description: Example Implementation of VFB Tracing Hooks with ARTI
       Macros.
8   *********************************************************************/
9
10  /*
       *********************************************************************
11   * INCLUDES
12   *********************************************************************/
13 #include "Std_Types.h"
14 #include "Rte_Hook_Arti.h"
15 #include "Rte_Arti.h"
16
17  /*
       *********************************************************************
18   * VFB HOOK IMPLEMENTATIONS
19   *********************************************************************/
20 #if !defined (Rte_Arti_Runnable_ConsumerComponent_RE2_Start)
21 void Rte_Arti_Runnable_ConsumerComponent_RE2_Start(void) {
22     /* #ID Rte_Arti_Runnable_ConsumerComponent_RE2_Start 1*/
23     ARTI_TRACE(USER, AR_CP_RTE_RUNNABLE, ct, 0, RteRunnable_Start, 1);
24 }
25 #endif
26
27 #if !defined (Rte_Arti_Runnable_ConsumerComponent_RE2_Return)
28 void Rte_Arti_Runnable_ConsumerComponent_RE2_Return(void) {
29     ARTI_TRACE(USER, AR_CP_RTE_RUNNABLE, ct, 0, RteRunnable_Return, 1);
30 }
31
32  /*
       *********************************************************************
33   * END OF FILE: Rte_Hook_Arti.c
34   *********************************************************************/
```

Listing 4.2: Example Implementation of VFB Tracing Hooks with ARTI Macros

5 Comparison between ARTI and ORTI

ARTI can be imagined as a successor of ORTI. ARTI may contain all information of an ORTI file. The file format has been changed from the OSEK-based syntax of ORTI to the model-driven ARXML syntax in ARTI. Additionally, ARTI covers new elements such as event-based tracing with hooks and multi-core aspects.

5.1 Mapping from ORTI to ARTI

All the elements in an ORTI file can be mapped to ARTI. The objects in the implementation section of ORTI are mapped to the ARXML sub-containers named `*Class` (ARTI classes). The objects in the information section are mapped to ARXML sub-containers named `*Instance` (ARTI instances). ARTI classes are only needed if enumerations or mappings are declared. Thus, not all ORTI objects have related ARTI classes. Table 5.1 shows the mapping of the defined ORTI objects and its attributes to the ARTI elements.

ORTI-attribute	
Class:	**ARTI-class**
Instance:	**ARTI-instance**
OS.RUNNINGTASK	
Class:	ArtiHardwareCoreClass.ArtiHardwareCoreClassCurrentTaskRef
Instance:	ArtiHardwareCoreInstance.ArtiHardwareCoreInstanceCurrentTaskRef
OS.RUNNINGTASKPRIORITY	
Class:	ArtiHardwareCoreClass.ArtiHardwareCoreClassRunningTaskPriorityRef
Instance:	ArtiHardwareCoreInstance.ArtiHardwareCoreInstanceRunningTaskPriorityRef
OS.RUNNINGISR2	
Class:	ArtiHardwareCoreClass.ArtiHardwareCoreClassCurrentIsrRef
Instance:	ArtiHardwareCoreInstance.ArtiHardwareCoreInstanceCurrentIsrRef
OS.SERVICETRACE	
Class:	ArtiOsClass.ArtiOsClassServiceTraceRef
Instance:	ArtiOsInstance.ArtiOsInstanceServiceTraceRef
OS.LASTERROR	
Class:	ArtiHardwareCoreClass.ArtiHardwareCoreClassLastErrorRef
Instance:	ArtiHardwareCoreInstance.ArtiHardwareCoreInstanceLastErrorRef
OS.CURRENTAPPMODE	
Class:	ArtiOsClass.ArtiOsClassAppModeRef
Instance:	ArtiOsInstance.ArtiOsInstanceAppModeRef
OS.VALID	
Class:	ArtiOsInstance.ArtiOsInstanceValidRef
Instance:	ArtiHardwareCoreInstance.ArtiHardwareCoreInstanceValidRef
TASK.PRIORITY	
Class:	ArtiOsTaskClass.ArtiOsTaskClassPriorityRef
Instance:	ArtiOsTaskInstance.ArtiOsTaskInstancePriorityRef
TASK.STATE	
Class:	ArtiOsTaskClass.ArtiOsTaskClassCurrentTaskStateRef
Instance:	ArtiOsTaskInstance.ArtiOsTaskInstanceCurrentTaskStateRef
TASK.STACK	
Class:	ArtiOsTaskClass.ArtiOsTaskClassStackRef
Instance:	ArtiOsTaskInstance.ArtiOsTaskInstanceStackRef

Attribute	Level	Reference
TASK.CURRENTACTIVATIONS		
	Instance:	ArtiOsTaskInstance.ArtiOsTaskInstanceCurrentActivationsRef
TASK.CONTEXT		
	Class:	ArtiOsTaskClass.ArtiOsTaskClassContextRef
	Instance:	ArtiOsTaskInstance.ArtiOsTaskInstanceContextRef
TASK.VALID		
	Instance:	ArtiOsTaskInstance.ArtiOsTaskInstanceValidRef
CONTEXT.ADDRESS		
	Instance:	ArtiOsContextInstance.ArtiOsContextInstanceAddressRef
CONTEXT.SIZE		
	Instance:	ArtiOsContextInstance.ArtiOsContextInstanceSizeRef
CONTEXT.VALID		
	Instance:	ArtiOsContextInstance.ArtiOsContextInstanceValidRef
STACK.SIZE		
	Instance:	ArtiOsStackInstance.ArtiOsStackInstanceSizeRef
STACK.BASEADDRESS		
	Instance:	ArtiOsStackInstance.ArtiOsStackInstanceBaseAddressRef
STACK.STACKDIRECTION		
	Instance:	ArtiOsStackInstance.ArtiOsStackInstanceSDirection
STACK.FILLPATTERN		
	Instance:	ArtiOsStackInstance.ArtiOsStackInstanceFillPatternRef
STACK.VALID		
	Instance:	ArtiOsStackInstance.ArtiOsStackInstanceValidRef
ALARM.ALARMTIME		
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceAlarmTimeRef
ALARM.CYCLETIME		
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceCycleTimeRef
ALARM.STATE		
	Class:	ArtiOsAlarmClass.ArtiOsAlarmClassStateRef
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceStateRef
ALARM.ACTION		
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceAction
ALARM.COUNTER		
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceCounter
ALARM.VALID		
	Instance:	ArtiOsAlarmInstance.ArtiOsAlarmInstanceValidRef
RESOURCE.STATE		
	Class:	ArtiOsResourceClass.ArtiOsResourceClassStateRef
	Instance:	ArtiOsResourceInstance.ArtiOsResourceInstanceStateRef
RESOURCE.LOCKER		
	Class:	ArtiOsResourceClass.ArtiOsResourceClassLockerRef
	Instance:	ArtiOsResourceInstance.ArtiOsResourceInstanceLockerRef
RESOURCE.PRIORITY		
	Instance:	ArtiOsResourceInstance.ArtiOsResourceInstancePriority
RESOURCE.VALID		
	Instance:	ArtiOsResourceInstance.ArtiOsResourceInstanceValidRef
MESSAGECONTAINER.MSGNAME		
	Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceMsgName
MESSAGECONTAINER.MSGTYPE		
	Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceMstType
MESSAGECONTAINER.QUEUESIZE		
	Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceQueueSizeRef
MESSAGECONTAINER.QUEUECOUNT		
	Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceQueueCountRef

MESSAGECONTAINER.FIRSTELEMENT	
Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceFirstElementRef
MESSAGECONTAINER.VALID	
Instance:	ArtiOsMessageContainerInstance.ArtiOsMessageContainerInstanceValidRef

Table 5.1: ORTI-object to ARTI-mapping

All the ARTI attributes are references to the parameters that are finally stored in `ArtiValues` sub-container. The ARTI instance parameters are `ArtiConstant`s or `ArtiExpression`s. The ARTI class parameters are `ArtiParamterTypeMap`s which declare the mapping from a value or an expression to the final displayable text or reference.

5.2 Mapping Vendor Specific Objects and Attributes

Vendor-specific objects are mapped to `ArtiGenericComponentClass` and `ArtiGenericComponentInstance`. These classes and instances are placed in the ARTI sub-container `ArtiGeneric`. Currently, there is only one exception namely that of modelled interrupts. In ARTI there is already an `ArtiOsIsrClass` and `ArtiOsIsrInstance` defined. So, vendor-specific objects in ORTI that are describing interrupts, are mapped to `ArtiOsIsr*`.

In contrast to ORTI, vendor-specific attributes cannot just be added to the related elements in ARTI. In ARTI, additional steps are necessary. At first, `ArtiGenericComponent`s needs to be defined. These components contain all the vendor-specific attributes. Finally, these additional components have to be referred by the related ARTI instance and ARTI class. So, in case a vendor-specific enum attribute like `vs_Enum_Task_Type` has to be added to the `ArtiOsTask*`-object, then

- an `ArtiGenericComponentClass` has to be created. This class has to contain the parameter definition `EnumTaskType` as `ArtiParameterTypeMap`.
- an `ArtiGenericComponentIntance` referring the created ArtiGenericComponentClass has to be created. The instance fills the parameter `EnumTaskType`.
- the `ArtiOsTaskClass` has to refer the created `ArtiGenericComponentClass`
- the `ArtiOsTaskInstance` has to refer the `ArtiGenericComponentInstance`.

Constant attributes do not need to be modeled in the ARTI class. For such attributes, it is enough to define that parameter in the ARTI-instance. This is possible if no mapping (i.e. reference mapping or enum mapping) is needed.

6 Example Configurations

The following two examples are intended to provide an example ARTI ECUC configuration for two use-cases.

6.1 Static Debugging

The ARTI ECU Configuration Parameter Container are intended to be configurable in such way that not all container parameters are necessary to be configured. The following listings show a minimal example how the three containers can be configured.

ARTI ECUConfiguration Parameters Container Arti

The ArtiValues container stores the available Os variables or expressions to track the task states.

```
<ECUC-CONTAINER-VALUE S="">
  <SHORT-NAME>ArtiValues</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
      /EcucDefs/Arti/ArtiValues</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiExpression_Core0_CurrentTask</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>OsCfg_Trace_OsCore_Core0_Dyn.CurrentTask</
              VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>
          ArtiObjectClassParameter_ArtiHwCore_CurrentTask</SHORT
          -NAME>
      <DESC>
        <L-2 L="EN">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">/
                StaticDebugging_ARTI_ECUC/Arti/ArtiValues/
                ArtiParameterTypeMap_TaskExpr</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </ECUC-CONTAINER-VALUE>
      <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>ArtiObjectInstanceParameter_Core0_CurrentTask
            </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">/
                StaticDebugging_ARTI_ECUC/Arti/ArtiValues/
                ArtiExpression_Core0_CurrentTask</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </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>ArtiParameterTypeMapPair_IdleTask_C0</
                SHORT-NAME>
            <DEFINITION-REF DEST="ECUC-CONTAINER-DEF">/AUTOSAR/
                EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap/
                ArtiParameterTypeMapPair</DEFINITION-REF>
            <PARAMETER-VALUES>
              <ECUC-NUMERICAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/
                    AUTOSAR/EcucDefs/Arti/ArtiValues/
                    ArtiParameterTypeMap/ArtiParameterTypeMapPair/
                    ArtiParameterTypeMapPairInput</DEFINITION-REF>
                <VALUE>0</VALUE>
              </ECUC-NUMERICAL-PARAM-VALUE>
              <ECUC-TEXTUAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/
                    AUTOSAR/EcucDefs/Arti/ArtiValues/
                    ArtiParameterTypeMap/ArtiParameterTypeMapPair/
                    ArtiParameterTypeMapPairOutput</DEFINITION-REF
                    >
                <VALUE>IdleTask_C0</VALUE>
```

```
            </ECUC-TEXTUAL-PARAM-VALUE>
          </PARAMETER-VALUES>
          <REFERENCE-VALUES>
            <ECUC-REFERENCE-VALUE>
              <DEFINITION-REF DEST="ECUC-CHOICE-REFERENCE-DEF">
                  /AUTOSAR/EcucDefs/Arti/ArtiValues/
                  ArtiParameterTypeMap/ArtiParameterTypeMapPair/
                  ArtiParameterTypeMapPairOutputRef</DEFINITION-
                  REF>
              <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                  StaticDebugging_ARTI_ECUC/Arti/ArtiOs/
                  VendorArtiOsTaskInstance_IdleTask_C0</VALUE-
                  REF>
            </ECUC-REFERENCE-VALUE>
          </REFERENCE-VALUES>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 6.1: ARTI ECUC Container ARXML Listing for ArtiValues

ARTI ECUConfiguration Parameters Container ArtiHardware

The ArtiHardware container is necessary to describe all available cores for debugging. Additionally each core can reference the core depended CurrentRunningTask variable.

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiHardware</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
      /EcucDefs/Arti/ArtiHardware</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiHardwareCoreClass</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Description</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
          AUTOSAR/EcucDefs/Arti/ArtiHardware/
          ArtiHardwareCoreClass</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiHardware/ArtiHardwareCoreClass/
              ArtiHardwareCoreClassCurrentTaskRef</DEFINITION-
              REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
              StaticDebugging_ARTI_ECUC/Arti/ArtiValues/
              ArtiObjectClassParameter_ArtiHwCore_CurrentTask</
              VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
```

```
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>VendorArtiHwCore_0</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
             AUTOSAR/EcucDefs/Arti/ArtiHardware/
             ArtiHardwareCoreInstance</DEFINITION-REF>
          <REFERENCE-VALUES>
            <ECUC-REFERENCE-VALUE>
              <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                 EcucDefs/Arti/ArtiHardware/
                 ArtiHardwareCoreInstance/
                 ArtiHardwareCoreInstanceCurrentTaskRef</DEFINITION
                 -REF>
              <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                 StaticDebugging_ARTI_ECUC/Arti/ArtiValues/
                 ArtiObjectInstanceParameter_Core0_CurrentTask</
                 VALUE-REF>
            </ECUC-REFERENCE-VALUE>
            <ECUC-REFERENCE-VALUE>
              <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                 EcucDefs/Arti/ArtiHardware/
                 ArtiHardwareCoreInstance/
                 ArtiHardwareCoreInstanceEcucCoreRef</DEFINITION-
                 REF>
              <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/
                 EcuC/EcucHardware/EcucCoreDefinition_C0</VALUE-REF
                 >
            </ECUC-REFERENCE-VALUE>
          </REFERENCE-VALUES>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
```

Listing 6.2: ARTI ECUC Container ARXML Listing for ArtiHardware

ARTI ECUConfiguration Parameters Container ArtiOs

The ArtiOs container in this simple example is just necessary to describe which task should be tracked for Os debugging in this example and references the task in the Os container. This is basically a duplication of information, but used to substitute the ORTI file.

```
<ECUC-CONTAINER-VALUE S="">
  <SHORT-NAME>ArtiOs</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
     /EcucDefs/Arti/ArtiOs</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiOsTaskInstance_IdleTask_C0</SHORT-
         NAME>
      <DESC>
```

```
        <L-2 L="EN">ARTI representation of EcuC Task "
          IdleTask_C0".</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
        AUTOSAR/EcucDefs/Arti/ArtiOs/ArtiOsTaskInstance</
        DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
            EcucDefs/Arti/ArtiOs/ArtiOsTaskInstance/
            ArtiOsTaskInstanceEcucRef</DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/Os
            /IdleTask_C0</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 6.3: ARTI ECUC Container ARXML Listing for ArtiOs

6.2 OS Task Tracing

ARTI ECUConfiguration Parameters Container Arti

The Arti container stores the TypeMaps for TaskId and CoreId, as well as the available OS Hooks and their layout.

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiValues</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
    /EcucDefs/Arti/ArtiValues</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiHook_ArtiOs_TaskRelease</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/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/ArtiHookClass</
            DEFINITION-REF>
          <VALUE>AR_CP_OS_TASKSCHEDULER</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_Release</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>VectorOsOs</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">/
          OsTaskTracing_ARTI_ECUC/Arti/ArtiValues/
          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">/
          OsTaskTracing_ARTI_ECUC/Arti/ArtiValues/
          ArtiParameterTypeMap_CoreId</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiHook_ArtiOs_TaskRelease</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/
              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/
              ArtiHookClass</DEFINITION-REF>
          <VALUE>AR_CP_OS_TASKSCHEDULER</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_Release</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>VectorOsOs</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">/
              OsTaskTracing_ARTI_ECUC/Arti/ArtiValues/
              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">/
              OsTaskTracing_ARTI_ECUC/Arti/ArtiValues/
              ArtiParameterTypeMap_CoreId</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiParameterTypeMap_CoreId</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
      AUTOSAR/EcucDefs/Arti/ArtiValues/ArtiParameterTypeMap<
      /DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>Core0</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>OsCore_Core0</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
```

```
        </PARAMETER-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>INVALID_TASK</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>22</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>INVALID_TASK</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
        </PARAMETER-VALUES>
      </ECUC-CONTAINER-VALUE>
      <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>IdleTask_C0</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>10</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>IdleTask_C0</VALUE>
          </ECUC-TEXTUAL-PARAM-VALUE>
```

```
          </PARAMETER-VALUES>
          <REFERENCE-VALUES>
            <ECUC-REFERENCE-VALUE>
              <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/
                  AUTOSAR/EcucDefs/Arti/ArtiOs/
                  ArtiOsTaskInstance/ArtiOsTaskInstanceEcucRef</
                  DEFINITION-REF>
              <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                  OsTaskTracing_ARTI_ECUC/Arti/ArtiOs/
                  VendorArtiOsTaskInstance_IdleTask_C0</VALUE-
                  REF>
            </ECUC-REFERENCE-VALUE>
          </REFERENCE-VALUES>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>notask</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>0xff</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>NO_TASK</VALUE>
            </ECUC-TEXTUAL-PARAM-VALUE>
          </PARAMETER-VALUES>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 6.4: ARTI ECUC Container ARXML Listing for ArtiValues

ARTI ECUConfiguration Parameters Container ArtiOs

The ArtiOs container in this simple example is just necessary to describe which task should be tracked for OS tracing in this example and references the task in the OS container. This is basically a duplication of information, but used to substitute the ORTI file.

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

```
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
     /EcucDefs/Arti/ArtiOs</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiOsInstance</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Actual values of the Vector OS</L-2>
      </DESC>
      <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">/ActiveEcuC/Os
             /OsOS</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">/
             OsTaskTracing_ARTI_ECUC/Arti/ArtiValues/
             ArtiHook_ArtiOs_TaskRelease</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiOsTaskInstance_IdleTask_C0</SHORT-
         NAME>
      <DESC>
        <L-2 L="EN">ARTI representation of EcuC Task &quot;
           IdleTask_C0&quot;.</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
         AUTOSAR/EcucDefs/Arti/ArtiOs/ArtiOsTaskInstance</
         DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
             EcucDefs/Arti/ArtiOs/ArtiOsTaskInstance/
             ArtiOsTaskInstanceEcucRef</DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/Os
             /IdleTask_C0</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

Listing 6.5: ARTI ECUC Container ARXML Listing for ArtiOs

7 Example Implementations

This chapter contains sample implementations of ARTI hook macros for typical scheduling tracing techniques or use-cases.

The ARTI hook macros provide the possibility to add tracing instrumentation to the AUTOSAR project with preserving freedom of implementation for tracing vendors. In addition to that the AUTOSAR Run-Time Interface proposes also an standardized implementation for tracing vendors and users to lean on.

7.1 Example Hook Implementation for Hardware Tracing

The following ARTI hook implementation focuses on the hardware tracing use-case. Generally, this implementation example is designed to overcome certain obstacles for hardware trace users:

1. Microcontroller specific limitation on traceable amount of cores
2. Minimize runtime impact on the AUTOSAR project
3. Minimize impact on the hardware trace interface
4. Ease hardware tracing setup
5. Freedom from interference to the application code

The data trace instrumentation is therefore an microcontroller atomic variable in order to guarantee minimum overhead while writing to it without any data stability mechanism being necessary. Each write access to it signalizes a state transition in one of the scheduling objects. It captures therefore the CoreId, scheduling entity Id and the state transition. While the ARTI trace macro ensures the injection at the correct places into the BSW without adding additional runtime overhead. In order to guarantee freedom from interference to the AUTOSAR project it is recommended to link the ARTI trace variables to a global accessible memory which is intended for measurement techniques such as the Infineon AURIX TriCore OLDA (Online Data Acquisition) memory or instrumentation data trace memory.

7.1.1 Common ARTI Header File

The following two examples do use a common ARTI header file, in which the top-level `ARTI_TRACE( )` macro is defined and expanded to module / *class* specific macros.

Listing 7.1 shows the code of the common ARTI header file.

```
/*********************************************************************
 *
 * AUTOSAR example implementation of ARTI Hooks for H/W trace.
 * Arti.h
 *
 *********************************************************************/

#ifndef ARTI_H
#define ARTI_H

/**
 *  \brief Common ARTI trace macro
 *
 *  This macro acts as a top level macro that
 *  - discards the parameter \p _contextName
 *  - takes the parameter \p _className to create a new module specific
     macro
 *
 *  Macro parameters:
 *  - \param[in] _contextName       Discarded
 *  - \param[in] _className         Used as token
 *  - \param[in] _instanceName      Passed to new module specific macro
 *  - \param[in] instanceParameter  Passed to new module specific macro
 *  - \param[in] _eventName         Passed to new module specific macro
 *  - \param[in] eventParameter     Passed to new module specific macro
 */
#define ARTI_TRACE( _contextName, \
                    _className, \
                    _instanceName, \
                    instanceParameter, \
                    _eventName, \
                    eventParameter \
                  ) \
    ARTI_TRACE_ ## _className ( \
        _instanceName, \
        (instanceParameter), \
        _eventName, \
        (eventParameter) \
    )

#endif
```

Listing 7.1: Common ARTI Example Header File

7.1.2 OS Instrumentation

The ARTI hook macros can be enabled or disabled by the global parameter **OS_USE_ARTI**. This allows to switch off the OS feature also after code generation without any remaining impact on the AUTOSAR project.

The **arti_os_trace** variable layout encodes the ThreadId, which is the umbrella term in this case for OS Tasks and Category 2 ISRs, the microcontroller CoreId, as well as the ARTI state transition to which the scheduling entity transits to. With that approach the same tracing variable can be used for Task and ISR scheduling trace events. To ensure that the variable is in use, a validation bit is added. This helps to detect whether memory initialization accesses to the trace variable have triggered trace messages, to be determined as no valid state transition at startup measurements.

To identify the trace event and the respective hook called in the OS, the state transition enumerations for the trace variable are proposed to be kept in the following manor:

Enhanced Task State Diagram	Standard Task State Diagram	Define
ARTI_OSARTITASK_ACTIVATE	ARTI_OSTASK_ACTIVATE	0
ARTI_OSARTITASK_START	ARTI_OSTASK_START	1
ARTI_OSARTITASK_WAIT	ARTI_OSTASK_WAIT	2
ARTI_OSARTITASK_RELEASE	ARTI_OSTASK_RELEASE	3
ARTI_OSARTITASK_PREEMPT	ARTI_OSTASK_PREEMPT	4
ARTI_OSARTITASK_TERMINATE	ARTI_OSTASK_TERMINATE	5
ARTI_OSARTITASK_RESUME		6
ARTI_OSARTITASK_CONTINUE		7

Table 7.1: Task State Transition Defines for the Enhanced and Standard State Model

Enhanced Cat2Isr State Diagram	Standard Cat2Isr State Diagram	Define
ARTI_OSARTICAT2ISR_START	ARTI_OSCAT2ISR_START	16
ARTI_OSARTICAT2ISR_STOP	ARTI_OSCAT2ISR_STOP	17
ARTI_OSARTICAT2ISR_ACTIVATE		18
ARTI_OSARTICAT2ISR_PREEMPT		19
ARTI_OSARTICAT2ISR_RESUME		20

Table 7.2: Category 2 ISR State Transition Defines

Listing 7.2 below shows an example ARTI header file to be included or implemented by the OS. It lists a subset of all necessary and available hooks to illustrate the instrumentation concept.

```
/***********************************************************************
 *
 * AUTOSAR example implementation of ARTI Hooks for H/W trace.
 * Os_Arti.h
 *
 ***********************************************************************/

#ifndef OS_ARTI_H
#define OS_ARTI_H

```

```
11 #include "Platform_Types.h"
12 #include "Arti.h"
13
14 #ifdef OS_USE_ARTI
15 #   define ARTI_TRACE_AR_CP_OS_TASK( \
16         _instanceName, \
17         instanceParameter, \
18         _eventName, \
19         eventParameter \
20     ) \
21     ARTI_TRACE_AR_CP_OS_TASK_ ## _eventName( \
22         (instanceParameter), \
23         (eventParameter) \
24     )
25
26 #   define ARTI_TRACE_AR_CP_OS_CAT2ISR( \
27         _instanceName, \
28         instanceParameter, \
29         _eventName, \
30         eventParameter \
31     ) \
32     ARTI_TRACE_AR_CP_OS_CAT2ISR_ ## _eventName( \
33         (instanceParameter), \
34         (eventParameter) \
35     )
36
37 #   define ARTI_TRACE_AR_CP_ARTI_CAT1ISR( \
38         _instanceName, \
39         instanceParameter, \
40         _eventName, \
41         eventParameter \
42     ) \
43     ARTI_TRACE_AR_CP_ARTI_CAT1ISR_ ## _eventName( \
44         (instanceParameter), \
45         (eventParameter) \
46     )
47 #else
48 #   define ARTI_TRACE_AR_CP_OS_TASK( \
49         _instanceName, \
50         instanceParameter, \
51         _eventName, \
52         eventParameter \
53     ) \
54     do{ \
55         (void)instanceParameter; \
56         (void)eventParameter; \
57     }while(0)
58
59 #   define ARTI_TRACE_AR_CP_OS_CAT2ISR( \
60         _instanceName, \
61         instanceParameter, \
62         _eventName, \
63         eventParameter \
64     ) \
65     do{ \
66         (void)instanceParameter; \
```

```
67          (void)eventParameter; \
68      }while(0)
69
70  #   define ARTI_TRACE_AR_CP_ARTI_CAT1ISR( \
71          _instanceName, \
72          instanceParameter, \
73          _eventName, \
74          eventParameter \
75      ) \
76      do{ \
77          (void)instanceParameter; \
78          (void)eventParameter; \
79      }while(0)
80  #endif
81
82  extern volatile uint32 arti_os_trace;
83
84  /** arti_os_trace encoding:
85
86   0000 80 00
87   ---- -- --
88      | || |
89      | || \ CoreId
90      | ||
91      | |\ StateId
92      | \ Bit 16 always written to 1 ("ValidOsWriteFlag)
93      \ ThreadId (16-bit)
94  **/
95
96  /** ARTI OS Task/ISR state transitions for AR_CP_OSARTI_TASK **/
97  /** The state transition for the standard state diagram uses the same
        defines **/
98  /** AR_CP_OS_TASK **/
99  #define ARTI_OSARTITASK_ACTIVATE     (0x00u)
100 #define ARTI_OSARTITASK_START        (0x01u)
101 #define ARTI_OSARTITASK_WAIT         (0x02u)
102 #define ARTI_OSARTITASK_RELEASE      (0x03u)
103 #define ARTI_OSARTITASK_PREEMPT      (0x04u)
104 #define ARTI_OSARTITASK_TERMINATE    (0x05u)
105 #define ARTI_OSARTITASK_RESUME       (0x06u)
106 #define ARTI_OSARTITASK_CONTINUE     (0x07u)
107 /** AR_CP_OS_CAT2ISR  **/
108 #define ARTI_OSARTICAT2ISR_START     (0x10u)
109 #define ARTI_OSARTICAT2ISR_STOP      (0x11u)
110 #define ARTI_OSARTICAT2ISR_ACTIVATE  (0x12u)
111 #define ARTI_OSARTICAT2ISR_PREEMPT   (0x13u)
112 #define ARTI_OSARTICAT2ISR_RESUME    (0x14u)
113 /** AR_CP_ARTI_CAT1ISR  **/
114 #define ARTI_OSARTICAT1ISR_START     (0x20u)
115 #define ARTI_OSARTICAT1ISR_STOP      (0x21u)
116
117 /** Bit 16 of art_os_trace is always written to 1 in order to identify a
        valid write of the OS
118  *  (not by e.g. data init routine of C-startup). **/
119 #define ARTI_VALID_OS_SIGNALING      (0x80u)
120
```

```
121
122 /** ARTI OS Hooks example implementations **/
123
124 /**
125  * \brief Expanded ARTI macro for OS task preemption event.
126  *
127  * State transition: Running -> Ready
128  *
129  * ARTI_TRACE() arguments:
130  *       - _contextName       = Unused
131  *       - _className         = AR_CP_OS_TASK
132  *       - _instanceName      = Unused
133  *       - instanceParameter = CoreId
134  *       - _eventName         = OsTask_Preempt
135  *       - eventParameter     = TaskId
136  *
137  * - \param CoreId ID of the core that the macro is executed on
138  * - \param TaskId ID of the preempted task
139  */
140 #define ARTI_TRACE_AR_CP_OS_TASK_OsTask_Preempt(CoreId, TaskId) \
141  { arti_os_trace = ((TaskId)<<16) | (ARTI_VALID_OS_SIGNALING<<8) | (
        ARTI_OSARTITASK_PREEMPT<<8) | (CoreId); }
142
143 /**
144  * \brief Expanded ARTI macro for OS task start event.
145  *
146  * State transition: Ready -> Running
147  *
148  * ARTI_TRACE() arguments:
149  *       - _contextName       = Unused
150  *       - _className         = AR_CP_OS_TASK
151  *       - _instanceName      = Unused
152  *       - instanceParameter = CoreId
153  *       - _eventName         = OsTask_Start
154  *       - eventParameter     = TaskId
155  *
156  * - \param CoreId ID of the core that the macro is executed on
157  * - \param TaskId ID of the starting task
158  */
159 #define ARTI_TRACE_AR_CP_OS_TASK_OsTask_Start(CoreId, TaskId) \
160     do{ \
161         arti_os_trace = ((TaskId)<<16) | (ARTI_VALID_OS_SIGNALING<<8) | (
               ARTI_OSARTITASK_START<<8) | (CoreId); \
162     }while(0)
163
164 /**
165  * \brief Expanded ARTI macro for OS ISR2 stop event.
166  *
167  * State transition: Running -> Inactive
168  *
169  * ARTI_TRACE() arguments:
170  *       - _contextName       = Unused
171  *       - _className         = AR_CP_OS_CAT2ISR
172  *       - _instanceName      = Unused
173  *       - instanceParameter = CoreId
174  *       - _eventName         = OsCat2Isr_Stop
```

```
175  *       - eventParameter     = Isr2Id
176  *
177  *  - \param CoreId ID of the core that the macro is executed on
178  *  - \param Isr2Id ID of the stopping CAT2 ISR
179  */
180 #define ARTI_TRACE_AR_CP_OS_CAT2ISR_OsCat2Isr_Stop(CoreId, Isr2Id) \
181     do{ \
182         arti_os_trace = ((Isr2Id)<<16) | (ARTI_VALID_OS_SIGNALING<<8) | (
               ARTI_OSARTICAT2ISR_STOP<<8) | (CoreId); \
183     }while(0)
184
185 /**
186  *  \brief Expanded ARTI macro for ISR1 stop event.
187  *
188  *  State transition: Inactive -> Running
189  *
190  *  ARTI_TRACE() arguments:
191  *       - _contextName       = Unused
192  *       - _className         = AR_CP_ARTI_CAT1ISR
193  *       - _instanceName      = Unused
194  *       - instanceParameter = CoreId
195  *       - _eventName         = OsCat1Isr_Start
196  *       - eventParameter     = Isr1Id
197  *
198  *  - \param CoreId ID of the core that the macro is executed on
199  *  - \param Isr1Id ID of the stopping CAT1 ISR
200  */
201 #define ARTI_TRACE_AR_CP_ARTI_CAT1ISR_OsCat1Isr_Start(CoreId, Isr1Id) \
202     do{ \
203         arti_os_trace = ((Isr1Id)<<16) | (ARTI_VALID_OS_SIGNALING<<8) | (
               ARTI_OSARTICAT1ISR_START<<8) | (CoreId); \
204     }while(0)
205
206 /**
207  *  \brief Expanded ARTI macro for ISR1 stop event.
208  *
209  *  State transition: Running -> Inactive
210  *
211  *  ARTI_TRACE() arguments:
212  *       - _contextName       = Unused
213  *       - _className         = AR_CP_ARTI_CAT1ISR
214  *       - _instanceName      = Unused
215  *       - instanceParameter = CoreId
216  *       - _eventName         = OsCat1Isr_Stop
217  *       - eventParameter     = Isr1Id
218  *
219  *  - \param CoreId ID of the core that the macro is executed on
220  *  - \param Isr1Id ID of the stopping CAT1 ISR
221  */
222 #define ARTI_TRACE_AR_CP_ARTI_CAT1ISR_OsCat1Isr_Stop(CoreId, Isr1Id) \
223     do{ \
224         arti_os_trace = ((Isr1Id)<<16) | (ARTI_VALID_OS_SIGNALING<<8) | (
               ARTI_OSARTICAT1ISR_STOP<<8) | (CoreId); \
225     }while(0)
226
```

```
227 #endif /* OS_ARTI_H */
```

Listing 7.2: ARTI Macros Example OS Header for Hardware Trace Instrumentation

The actual implementation C-File in listing 7.3 is therefore only instantiating the data trace variable. It is important to mention that the variable should be volatile to ensure that it is not going to be removed during compiler optimization since there is no consumer actually reading the variable. Additionally the compiler pragma locating the variable at the project dependent trace memory shall illustrate this step to be considered for the project's linking phase.

```
1  /**********************************************************************
2   *
3   * AUTOSAR example implementation of ARTI Hooks for H/W trace.
4   * Os_Arti.c
5   *
6   **********************************************************************/
7
8  #include "Platform_Types.h"
9
10 volatile uint32 arti_os_trace __at(TRACE_MEMORY_ADDRESS);
```

Listing 7.3: ARTI Macros Example OS Implementation for Hardware Trace Instrumentation

7.1.3 RTE Instrumentation

As mentioned above in Chapter 4, ARTI uses the VFB Tracing Hooks as an own trace client. Listing 7.4 shows an example ARTI header file to be included or implemented by the Rte. It lists a subset of all necessary and available hooks to illustrate the instrumentation concept.

The `arti_rte_trace` variable (line 16) layout encodes the IDs of the Runnable Entity as well as of the affected processing core. As depicted in line 18 – 28, the lower Byte can be used for the CoreId and the higher Bytes for the identification of the Runnable Entity.

```
1  /**********************************************************************
2   *
3   * AUTOSAR example implementation of ARTI Hooks for H/W trace.
4   * Rte_Arti.h
5   *
6   **********************************************************************/
7
8  #ifndef RTE_ARTI_H
9  #define RTE_ARTI_H
10
11 #include "Platform_Types.h"
12 #include "Arti.h"
13
14 /**
15  *  \brief ARTI trace macro for class 'AR_CP_RTE_RUNNABLE'
```

```
16   *
17   * This macro takes the parameter \p _eventName to create an event
       specific macro for
18   * VFB tracing.
19   *
20   * Macro parameters:
21   * - \param[in] _instanceName      Discarded
22   * - \param[in] instanceParameter  [const_Rte_CDS_<cts>_ptr]|0 ->
       Discarded
23   * - \param[in] _eventName         Used as token
24   * - \param[in] eventParameter     Passed to new macro
25   *
26   */
27  #define ARTI_TRACE_AR_CP_RTE_RUNNABLE( \
28          _instanceName, \
29          instanceParameter, \
30          _eventName, \
31          eventParameter \
32      ) \
33      ARTI_TRACE_AR_CP_RTE_RUNNABLE_ ## _eventName( \
34          (eventParameter) \
35      )
36
37  extern volatile uint32 arti_rte_trace;
38
39  /** arti_rte_trace encoding:
40      [0:7] : Core ID
41      [8:31] : Runnable ID
42
43      000000 00
44      ------ --
45           | |
46           | \ CoreId
47           |
48           \ RunnableId
49   **/
50
51  /** ARTI RTE runnable transitions **/
52  #define ARTI_COMMON_RUNNABLE_RETURN_ID (0x0 << 8)
53
54
55  /** ARTI Rte Hooks example implementations **/
56
57  /**
58   * \brief Expanded ARTI macro for RTE VFB tracing Runnable started event
59   *
60   * \param RunnableId ID of the Runnable to be written to `arti_rte_trace
       `
61   */
62  #define ARTI_TRACE_AR_CP_RTE_RUNNABLE_RteRunnable_Start(RunnableId) \
63      do{ \
64          uint32 const CoreId = (uint32)GetCoreId( ); \
65          arti_rte_trace = ((RunnableId) << 8) | CoreId; \
66      }while(0)
67
68  /**
```

```
69  * \brief Expanded ARTI macro for RTE VFB tracing Runnable returned
       event
70  *
71  * - \param RunnableId Not used
72  */
73 #define ARTI_TRACE_AR_CP_RTE_RUNNABLE_RteRunnable_Return(RunnableId) \
74     do{ \
75         uint32 const CoreId = (uint32)GetCoreId( ); \
76         (void)RunnableId; \
77         arti_rte_trace = ARTI_COMMON_RUNNABLE_RETURN_ID | CoreId; \
78     }while(0)
79
80 #endif /* RTE_ARTI_H */
```

Listing 7.4: ARTI Macros Example Rte Header for Hardware Trace Instrumentation

The actual implementation C-File in Listing 7.5 is only instantiating the data trace variable (line 8). The variable must be volatile to ensure that it is not going to be removed during compiler optimization since there is no consumer actually reading it. Additionally, the compiler pragma locating the variable at the project dependent trace memory shall illustrate this step to be considered for the project's linking phase.

```
1  /*********************************************************************
2   *
3   * AUTOSAR example implementation of ARTI Hooks for H/W trace.
4   * Rte_Arti.c
5   *
6   *********************************************************************/
7
8  #include "Platform_Types.h"
9
10 volatile uint32 arti_rte_trace __at(TRACE_MEMORY_ADDRESS);
```

Listing 7.5: ARTI Macros Example RTE Implementation for Hardware Trace Instrumentation

7.2 Example for User Provided CAT1 Interrupt

Interrupts of category 1 can be modeled and implemented using the standard AUTOSAR tool chain. But it is also possible and explicitly allowed to implement them directly in the application, without modeling them in the system description or EcuC. E.g., timer interrupts are often handled this way. In this case, an ARTI consuming tool would not be able to detect the run times of this CAT1 interrupt. To still allow measuring these interrupts, ARTI allows to manually add the required hooks and EcuC items, so that tracing tools can recognize the flow of CAT1 interrupts.

7.2.1 Instrumenting the CAT1 interrupt with ARTI Hooks

To trace CAT1 interrupts, the entry and the exit of the interrupt routine should be instrumented with ARTI hooks of class *AR_CP_ARTI_CAT1ISR*. An example implementation could be:

```
#define OS_USE_ARTI

#include "Os_Arti.h"

void __interrupt( 7 ) __vector_table( 0 ) Cat1_TimerIsr (void)
{
    ARTI_TRACE(
        NOSUSP, /* Indicates that interrupts are locked at this point */
        AR_CP_ARTI_CAT1ISR, /* ARTI trace class name for CAT1 ISRs */
        OsOs, /* OS Short Name */
        0uL, /* Index of the core this code runs on */
        OsCat1Isr_Start, /* Event name */
        7uL /* Trace ID of this ISR */
    );

    /* Enable interrupts */
    __enable( );

    incrementTimers( );
    kickTimerRelatedCat2Isrs( );

    ARTI_TRACE(
        SPRVSR, /* Indicates that interrupts are enabled, but the
            execution context has sufficient rights to disable them */
        AR_CP_ARTI_CAT1ISR, /* ARTI trace class name for CAT1 ISRs */
        OsOs, /* OS Short Name */
        0uL, /* Index of the core this code runs on */
        OsCat1Isr_Stop, /* Event name */
        7uL /* Trace ID of this ISR */
    );
}
```

Listing 7.6: Example Implementation of ARTI Hooks for CAT1 Interrupt

In this example code:

- the short name of the OS is *OsOs*,
- the core that gets the timer interrupt is core 0 (`__vector_table( 0 )`), and
- the trace index for the CAT1 ISR, which is equal to the hardware interrupt vector *7* in this example (`__interrupt( 7 )`).

7.2.2 EcuC representation of CAT1 ARTI Hooks

Manually implemented ARTI Hooks of CAT1 Interrupts need to be modeled manually and added to the EcuC description. A model within the EcuC container according to the example code shown above could look like this:

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>Arti</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Arti</DEFINITION
      -REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiOs</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Arti/ArtiOs</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>ArtiOsIsrClassCat1</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiOs/ArtiOsIsrClass</DEFINITION-REF>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>ArtiOsIsrCat1Timer</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiOs/ArtiOsIsrInstance</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-TEXTUAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiOs/ArtiOsIsrInstance/
                  ArtiOsIsrInstanceCategory</DEFINITION-REF>
              <VALUE>CATEGORY_1</VALUE>
            </ECUC-TEXTUAL-PARAM-VALUE>
            <ECUC-TEXTUAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-FUNCTION-NAME-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiOs/ArtiOsIsrInstance/
                  ArtiOsIsrInstanceFunction</DEFINITION-REF>
              <VALUE>Os_Cat1_Interrupt_TimerIsr</VALUE>
            </ECUC-TEXTUAL-PARAM-VALUE>
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                  EcucDefs/Arti/ArtiOs/ArtiOsIsrInstance/ArtiOsIsrInstanceId
                  </DEFINITION-REF>
              <VALUE>7</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
          </PARAMETER-VALUES>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

In this example configuration:

- the ISR category is "CATEGORY_1"
- the ISR function name ist "Os_Cat1_Interrupt_TimerIsr", and
- the ISR IDX is "7".

8 Outlook

With ongoing development, ARTI shall provide more and more features for users to understand their AUTOSAR project's run-time behavior. Besides OS and RTE, ARTI is aiming to support in future OS protection Hooks and other fault recognition interfaces.

Apart from that, Adaptive Platform will also be addressed.

9 Document Information

Known Limitations

- No known limitations yet.

9.1 Related documentation

9.1.1 Input documents & related standards and norms

[1] Specification of Operating System
AUTOSAR_CP_SWS_OS

[2] Specification of AUTOSAR Run-Time Interface
AUTOSAR_CP_SWS_ARTI

[3] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[4] Timing Analysis and Design
AUTOSAR_FO_TR_TimingAnalysis

9.1.2 Related specification

This document should explain the standard addressed in [2, SWS ARTI].
ARTI focuses heavily on scheduling and run-time analysis and therefore affects [1, SWS OS] and [3, SWS RTE].
An example implementation of ARTI can be found in the Timing Reference Platform, which is described in Appendix A of [4, TR TimingAnalysis].