

# BSW Coding Guidelines

www.bosch.com



**2014-02-21** This document contains AUTOSAR specific information. It may be used according to AUTOSAR policy.  
Disclosure is prohibited without the written consent of Robert Bosch GmbH.  
© Robert Bosch GmbH reserves all rights even in the event of industrial property rights.  
We reserve all rights of disposal such as copying and passing on to third parties.

## Abstract

The domain of these guidelines is the BSW development in CDG-SMT departments, they might also be used in other departments for BSW development. Guidelines are driven from requirements from AUTOSAR, MISRA Coding standard and Bosch internal issues. Main focus is the development of portable software which is mostly independent from hardware, compilers and in general from any development environment.

Consideration of BSW Coding Guideline:

- ▶ For a **functional update** which will be provided with the next official release baseline the code of a software component will be adapted to the newly defined BSW Coding Guideline version.
- ▶ A **bugfix** is only a code correction in a small area. A bugfix contains no functional or non-functional extensions. Therefore there is no consideration of a new BSW Coding Guideline version for a bugfix.
- ▶ **Released software components** should be adapted to a new BSW Coding Guideline version only if there is an explicit demand from a UBK product line.

# Table of Contents

<b>a</b>	<b>Scope .....</b>	<b>12</b>
<b>a.1</b>	<b>Used Conventions .....</b>	<b>12</b>
a.1.1	Used Keywords .....	12
a.1.2	Rule Structure .....	13
a.1.3	Rule Set Structure .....	14
<b>a.2</b>	<b>Used Guideline Artifacts .....</b>	<b>14</b>
<b>1</b>	<b>Introduction .....</b>	<b>15</b>
<b>2</b>	<b>Glossary .....</b>	<b>16</b>
<b>3</b>	<b>Rule Set: C Coding Guidelines .....</b>	<b>17</b>
<b>3.1</b>	<b>Rule Set: C Language Rules .....</b>	<b>17</b>
3.1.1	General C90 Language Topics .....	17
3.1.2	MISRA and HIS Metrics Conformity .....	20
3.1.3	Initializations .....	22
3.1.4	Expressions .....	24
3.1.5	Control Statement Expressions .....	30
3.1.6	Control Flow .....	33
3.1.7	Switch Statements .....	34
3.1.8	Structures and Unions .....	36
3.1.9	Preprocessing Directives .....	38
3.1.10	Macros .....	41
<b>3.2</b>	<b>Rule Set: Compiler Abstraction / Portability .....</b>	<b>44</b>
3.2.1	Main Rule .....	44
3.2.2	Prohibition of Compiler Specifics .....	44
3.2.3	Abstraction of Addressing Keywords .....	46
3.2.4	Abstraction of Memory Mapping .....	50
3.2.5	Abstraction of Inline Functions .....	55
3.2.6	Handling of Assembler Instructions .....	57
3.2.7	Prohibition of Dynamic Memory .....	57
3.2.8	Pointer Type Conversions .....	58
3.2.9	Pointer Arithmetic and Arrays .....	59

3.2.10	Ensure Usability of BSW Modules in C++ Environments .....	61
3.2.11	Prohibition of Open Source Software .....	65
<b>3.3</b>	<b>Rule Set: Types and Symbols .....</b>	<b>65</b>
3.3.1	Base Integer and Float Data Types .....	65
3.3.2	Optimized Integer Data Types .....	67
3.3.3	Standard Symbols .....	68
3.3.4	Specific Types and Symbols for Communication Software .....	70
3.3.5	Module Specific Add Ons .....	77
<b>3.4</b>	<b>Rule Set: Naming Convention .....</b>	<b>77</b>
<b>3.5</b>	<b>Rule Set: Module Design and Implementation .....</b>	<b>88</b>
3.5.1	Basis Set of Module Files .....	88
3.5.2	Header Include Concept .....	93
3.5.3	BSW Service Module with and without RTE .....	99
3.5.4	Design of APIs (Application Programming Interfaces) .....	102
3.5.5	Design of Processes and Interrupt Service Routines .....	107
3.5.6	Definition and Declaration of Functions and Objects .....	108
3.5.7	Code Re-entrancy .....	112
3.5.8	Providing of Version Information .....	113
3.5.9	BSW Scheduler and Exclusive Areas .....	116
<b>3.6</b>	<b>Rule Set: Style Guide .....</b>	<b>117</b>
3.6.1	Common File Style .....	117
3.6.2	Block Style .....	121
3.6.3	Comments .....	122
<b>4</b>	<b>Rule Set: ECU Configuration .....</b>	<b>125</b>
<b>4.1</b>	<b>Introduction to ECUConfig .....</b>	<b>125</b>
4.1.1	Authoring of ECUC artifacts .....	126
4.1.2	How ECUC Values Are Processed .....	126
<b>4.2</b>	<b>ECUC Values .....</b>	<b>127</b>
4.2.1	ECUC Values: Content Responsibility .....	127
4.2.2	ECUC Values: Editing, Checking, and Reviewing .....	129
4.2.3	ECUC Values: Testing .....	129
4.2.4	ECUC Values: Content-related Procedures .....	130

<b>4.3</b>	<b>ECUC Parameter Definitions</b>	<b>130</b>
4.3.1	ECUC ParamDefs: Content Responsibility	130
4.3.2	ECUC ParamDefs: Editing, Checking, and Reviewing	130
4.3.3	ECUC ParamDefs: Testing	131
4.3.4	ECUC ParamDefs: Content-related Procedures	131
<b>4.4</b>	<b>ECUC Processors</b>	<b>135</b>
4.4.1	ECUC Processors: Content Responsibility and Language	135
4.4.2	ECUC Processors: Editing, Checking, and Reviewing	135
4.4.3	ECUC Processors: Testing	135
4.4.4	ECUC Processors: Documentation	136
4.4.5	ECUC Processors: General Content-related Procedures	136
4.4.6	ECUC Processors: oAW-specific Procedures	139
4.4.7	ECUC Processors: Perl-specific Procedures	141
<b>4.5</b>	<b>Build Action Manifests</b>	<b>149</b>
4.5.1	BuildAction Declarations	149
4.5.2	Input/Output Declarations	151
4.5.2.1	Artefact Declarations	151
4.5.2.2	Model Reference Declarations	153
4.5.3	Action Specific Declarations	155
<b>5</b>	<b>Rule Set: Perl Coding Rules</b>	<b>158</b>
<b>5.1</b>	<b>Code Layout and Comments</b>	<b>158</b>
<b>5.2</b>	<b>Naming Conventions</b>	<b>160</b>
<b>5.3</b>	<b>Coding Conventions</b>	<b>164</b>
<b>5.4</b>	<b>Programming Tips</b>	<b>165</b>
<b>5.5</b>	<b>Templates</b>	<b>169</b>
<b>5.6</b>	<b>Usage of perltidy</b>	<b>169</b>
<b>6</b>	<b>Rule Set: oAW Rules</b>	<b>170</b>
<b>6.1</b>	<b>General oAW rules</b>	<b>170</b>
<b>6.2</b>	<b>Model Workflow Environment</b>	<b>170</b>
<b>6.3</b>	<b>Configuration Validator files (*.chk)</b>	<b>171</b>
<b>6.4</b>	<b>Extensions (Xtend files, *.ext)</b>	<b>172</b>

05	<b>6.5</b>	<b>Generator templates (Xpand, *.xpt).....</b>	<b>174</b>
	<b>6.6</b>	<b>ID Generator templates .....</b>	<b>177</b>
10	<b>6.7</b>	<b>Auxilliary rules.....</b>	<b>177</b>
	<b>7</b>	<b>Rule Set: Data Description .....</b>	<b>179</b>
	<b>7.1</b>	<b>Data Types .....</b>	<b>179</b>
15	7.1.1	ApplicationDataTypes .....	180
	7.1.1.1	Primitive ApplicationDataTypes .....	181
	7.1.1.1.1	ApplicationDataType_Value .....	181
20	7.1.1.1.2	ApplicationDataType_Boolean .....	183
	7.1.1.1.3	ApplicationDataType_String .....	183
	7.1.1.2	Complex Application DataType .....	184
25	7.1.1.2.1	ApplicationDataType_Array .....	184
	7.1.1.2.2	ApplicationDataType_Structure .....	185
	7.1.1.2.3	Rules .....	186
	7.1.2	ImplementationDataType .....	187
30	7.1.2.1	ImplementationDataType_Value .....	188
	7.1.2.1.1	Rules .....	188
	7.1.2.2	ImplementationDataType_Array .....	189
35	7.1.2.2.1	Rules .....	190
	7.1.2.3	ImplementationDataType_Structure .....	190
	7.1.2.3.1	Rules .....	190
40	7.1.3	Data Type Mapping .....	190
	7.1.3.1	Rules .....	191
	<b>7.2</b>	<b>DataPrototypes .....</b>	<b>192</b>
45	7.2.1	ParameterDataPrototype (Calibration Parameters).....	192
	7.2.1.1	ParameterDataPrototype of Category Value .....	193
	7.2.1.1.1	Rules .....	193
50	7.2.1.2	ParameterDataPrototype of Category Array.....	196
	7.2.1.2.1	Rules .....	196
	7.2.1.3	ParameterDataPrototype of Category Structure .....	196
55	7.2.1.3.1	Rules .....	196
	7.2.1.4	ApplicationSwComponentType Vs ParameterSwComponentType .....	196
	7.2.1.5	Usage of SW-CALIBRATION-ACCESS .....	197
60	7.2.1.5.1	Rules .....	197
	7.2.1.6	Usage of SW-IMPL-POLICY .....	198
	7.2.1.6.1	Rules .....	198
65	7.2.2	VariableDataPrototype(Measurement Points) .....	198

7.2.2.1	VariableDataPrototype of Category Value .....	199
7.2.2.1.1	Rules .....	199
7.2.2.2	VariableDataPrototype of Category Boolean .....	199
7.2.2.2.1	Rules .....	200
7.2.2.3	VariableDataPrototype of Category Array .....	200
7.2.2.3.1	Rules .....	200
7.2.2.4	VariableDataPrototype of Category Structure .....	200
7.2.2.4.1	Rules .....	201
7.2.2.5	Compatibility of Variable Data Prototypes and Parameter Data Protoptypes .....	201
7.2.2.6	Rules .....	201
<b>7.3</b>	<b>Usage of Attributes of SwDataDefProps .....</b>	<b>201</b>
7.3.1	Rules .....	201
<b>7.4</b>	<b>Initial Values .....</b>	<b>203</b>
7.4.1	Description .....	203
7.4.1.1	Initial Values for Calibration Parameter Overview .....	204
7.4.1.2	Assigning Initial Values for Calibration Parameter .....	204
7.4.1.2.1	Initial Value assigned to individual Calibration Parameter .....	204
7.4.2	Rules .....	205
<b>7.5</b>	<b>Elements referred by Data Def Properties .....</b>	<b>205</b>
7.5.1	Data Constraints .....	205
7.5.1.1	Rules .....	207
7.5.2	Computation Methods .....	207
7.5.2.1	IDENTICAL .....	208
7.5.2.2	LINEAR .....	208
7.5.2.3	RAT_FUNC.....	209
7.5.2.4	TEXTTABLE.....	210
7.5.2.5	Rules .....	211
7.5.3	Units and UnitGroup .....	214
7.5.3.1	Units.....	214
7.5.3.2	UnitGroup.....	214
7.5.3.3	Rules .....	216
<b>8</b>	<b>Rule Set: Basis Software Module Description (BSWMD) .....</b>	<b>218</b>
<b>8.1</b>	<b>Common Aspects of BSWMD .....</b>	<b>218</b>
<b>8.2</b>	<b>BSWMD Model View .....</b>	<b>222</b>

8.2.1	Top Layers of BSWMD .....	222
8.2.1.1	BswModuleDescription .....	223
8.2.1.2	BswInternalBehavior .....	226
8.2.1.3	BswImplementation .....	228
8.2.2	Splitting of BSWMD Files .....	232
8.2.3	Differences between BSWMD and SWCD .....	236
<b>8.3</b>	<b>BSWMD Use Cases .....</b>	<b>237</b>
8.3.1	BSW Measurement and Calibration Support.....	238
8.3.1.1	Definition of Measurement Variables .....	239
8.3.1.2	Definition of Calibration Parameters .....	240
8.3.2	Scheduling of BSW via RTE .....	244
8.3.2.1	BswModuleEntitys, BswModuleEntrys and BswEvents .....	244
8.3.2.1.1	BswCalledEntity .....	245
8.3.2.1.2	BswInterruptEntity .....	247
8.3.2.1.3	BswSchedulableEntity .....	248
8.3.2.2	BSW Modes .....	253
8.3.2.2.1	BSW Mode Management on the Mode Manager side .....	254
8.3.2.2.2	BSW Mode Management on the Mode User side .....	256
8.3.2.2.3	BSW Mode Access .....	261
8.3.2.3	BswTriggers .....	263
8.3.2.3.1	BswExternalTriggers on the Sender side .....	263
8.3.2.3.2	BswExternalTriggers on the Receiver side .....	265
8.3.2.3.3	BswInternalTriggers.....	267
8.3.2.4	Mapping between BSW and a corresponding SWC .....	269
8.3.3	BSW Documentation .....	273
8.3.4	BSW Module API Description (BswModuleEntry) .....	274
8.3.4.1	Common Attributes of a BswModuleEntry .....	274
8.3.4.2	Definition of a Return Value and Arguments of a BswModuleEntry .....	278
8.3.5	BSW Service Needs .....	289
8.3.6	BSW Exclusive Areas .....	289
<b>9</b>	<b>Rule Set: AUTOSAR Package Structure .....</b>	<b>290</b>
<b>9.1</b>	<b>Introduction to Use of ARPackage .....</b>	<b>290</b>
9.1.1	Splitting ARPackages into different files.....	291
9.1.2	Referencing ARElements .....	291
9.1.3	ReferenceBases .....	292



<b>9.2</b>	<b>Main Structure of ARPackages .....</b>	<b>292</b>
<b>9.3</b>	<b>ARPackage for Basic Software .....</b>	<b>300</b>
<b>9.4</b>	<b>ARPackage for Project based ARElements .....</b>	<b>303</b>
<b>9.5</b>	<b>Referencing Elements .....</b>	<b>305</b>
9.5.1	References to other ARElements .....	305
9.5.2	Use of ReferenceBase .....	305
<b>10</b>	<b>Rule Set: Central Elements .....</b>	<b>307</b>
<b>10.1</b>	<b>Centralisation of Element Kinds .....</b>	<b>307</b>
10.1.1	Units .....	307
10.1.2	CompuMethods .....	308
10.1.3	System Constants .....	308
10.1.4	Address Methods .....	309
10.1.5	Record Layouts .....	309
10.1.6	Keywords .....	309
10.1.7	Data types .....	309
10.1.8	SwBaseType .....	310
10.1.9	Data Constraints .....	310
10.1.10	DataTypeMappingSet .....	310
<b>10.2</b>	<b>Data Types .....</b>	<b>311</b>
10.2.1	ApplicationDataTypes and ImplementationDataTypes .....	311
10.2.2	AUTOSAR Platform Types .....	312
10.2.3	AUTOSAR Standard Types .....	314
10.2.4	ComStack Types .....	314
10.2.5	Other Types .....	315
<b>10.3</b>	<b>AR Package Structure for Central Elements .....</b>	<b>315</b>
<b>10.4</b>	<b>Delivery Aspects .....</b>	<b>321</b>
<b>11</b>	<b>Rule Set: CLF (Classification File) .....</b>	<b>322</b>
<b>11.1</b>	<b>Introduction .....</b>	<b>322</b>
11.1.1	CLF projects overview .....	322
11.1.2	Tool Environment .....	322
11.1.3	References .....	322

<b>11.2</b>	<b>Guidelines</b>	<b>323</b>
11.2.1	Structural Conventions	323
11.2.2	Naming Conventions	326
<b>11.3</b>	<b>Examples for Variants Usage</b>	<b>328</b>
11.3.1	Simple enabling / disabling of folders	328
11.3.2	Using configuration variants	329
<b>A</b>	<b>Rules summary</b>	<b>334</b>
<b>B</b>	<b>Rules Derivation</b>	<b>388</b>
<b>C</b>	<b>List of Basic Software Modules</b>	<b>395</b>
<b>D</b>	<b>Physical and Logical Types</b>	<b>398</b>
<b>E</b>	<b>HIS Metrics Overview</b>	<b>400</b>
<b>F</b>	<b>References</b>	<b>402</b>
F.1	Robert Bosch GmbH AUTOSAR Guidelines	402
F.2	Robert Bosch GmbH AUTOSAR Artifacts	402
F.3	AUTOSAR Main Documents	402
F.4	AUTOSAR Basic Software Architecture and Runtime Documents	402
F.5	AUTOSAR Methodology and Templates Documents	403
F.6	Others	403
<b>G</b>	<b>Approval of Guideline</b>	<b>404</b>
<b>H</b>	<b>Documentation</b>	<b>405</b>
<b>I</b>	<b>Technical Terms</b>	<b>406</b>
	ACTIVITY	406
	AUTOSAR ITEMS	406
	AUTOSAR RULES	407
	BOSCH RULES	407
	Codes	407
	Control elements	408
	Others	408
	Products	408
	Tool	408
	Variables	408

00  
05  
10  
15  
20  
25  
30  
35  
40  
45  
50  
55  
60  
65  
70



XML/SGML-Attributes .....	408
XML/SGML-Tags .....	409
<b>J Version Information .....</b>	<b>410</b>

## a Scope

These guidelines are obligatory for all BSW software delivered by CDG-SMT but may be also used for BSW development outside CDG-SMT. Existing software which was not developed conform to these guidelines has to be adapted to them. Each SW team has to decide when this migration step will be done. Development of new software starts with these coding guidelines.

These guidelines shall be used for both AUTOSAR SW and non-AUTOSAR-SW. It shall be noticed, that even non-AUTOSAR-SW shall use data types defined in AUTOSAR, e.g. uint8. To have only one set of guidelines for AUTOSAR and non AUTOSAR modules simplifies the development and the handling of code.

If AUTOSAR defines something different to the BSW Coding Guideline the definition of AUTOSAR is more substantial for an AUTOSAR based BSW module.

A complete delivery of SW shall be conforming to a special main AUTOSAR release. A mixture of more than one main AUTOSAR release (AR3.x and AR4.x) and their different methodologies is not possible. Therefore all modules (which are defined in AUTOSAR and others who are not defined in AUTOSAR) have to follow the requirements of a special release of AUTOSAR and fulfill these coding guidelines. It is possible that features from a special (newer) revision can be implemented even if the version of the AUTOSAR release where the development depends on is an older one. The version of the AUTOSAR release where the development depends on is defined by the CUBAS FO meeting.

All rules of the coding guidelines are defined with a general focus, mainly independent from a special AUTOSAR release. Currently the guidelines are based on AR4.x. AR3.x is not explicitly supported.

## a.1 Used Conventions

### a.1.1 Used Keywords

In this document the following specific semantics are used (derived from the Internet Engineering Task Force (IETF), reference: <http://tools.ietf.org/html/rfc2119>), which is also used by AUTOSAR consortium similarly.

The keywords "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT" and "MAY" in this document are to be interpreted as:

**MUST** This word means that the definition is an absolute requirement of the specification due to legal issues.

This word **SHALL NOT** be used in RB's AUTOSAR guidelines as long as we do not define anything due to legal issues.

**MUST NOT** This phrase means that the definition is an absolute prohibition of the specification due to legal constraints.

This phrase shall not be used in RB's AUTOSAR guidelines as long as we do not define anything due to legal issues.

**SHALL** This word means that the definition is an absolute requirement of the specification.

**SHALL NOT** This phrase means that the definition is an absolute prohibition of the specification.

**SHOULD** This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

**SHOULD NOT** This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

**MAY** This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, **MUST** be prepared to interoperate with another implementation, which does include the option, though perhaps

with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

## a.1.2 Rule Structure

The main instructions, how someone should work with AUTOSAR is kept as "rules" with a detailed and precise structure.

For all rules defined in the guideline a rule summary will be created in the report rear.

First we give an example and then the formal structure of the rules.

### Rule Exa007: Driving on the right side of the road

#### Instruction

**Scope:** Germany, Austria, France

When using a car, a motor cycle or a bicycle you must use the right side of the road.

This is absolutely helpful for your health.

In some other countries it is not healthy to use the right side.

If you want to avoid confusion you should use a train.

### Rule {Rule\_Id}: Rule Title

#### Instruction

**Scope:** {Scope of the rule}. (Optional)

{Rule definition}

{Additional information}.

### Rule Structure Description

#### {Rule\_Id}

Each rule has a unique Id. It consists of an abbreviation of the document's name and a name part which has to be unique in the whole document.

The Rule\_Id has C-identifier syntax (no spaces, hyphen, ...).

Rule Title starts with the word "Rule". (This is the keyword used for creating the rules table in the report rear.)

Examples: ARPac\_003; BSWCode\_2\_3, ArCaDe\_52b.

#### {Rule Title}

The header or main keyword of the rule. The rule title should not extend beyond a single line.

#### {Scope of the rule}

Optional: the **Scope** of the rule (e.g. BU, AR-Version, ECU-domain)

If the rule is applicable under certain restrictions this is given as Scope. Scope may be an organisational unit, an ECU domain, a specific AUTOSAR Version or something else. If there are no such restrictions no scope is defined.

#### {Rule Definition}

Short and concise description of the rule.

It can consist of multiple paragraphs, cross references, and also lists.

Tables, and (sub-)notes are not possible. Pictures are not recommended (because of problems in further processing). If tables etc. are necessary, you will find them in the additional information.

The rule definition may be omitted if the complete rule is defined in the rule title.

{Additional Information}

Here you find detailed explanations, reasons, backgrounds, use cases and examples.

## a.1.3 Rule Set Structure

### Rule Set Description

The BSW coding guidelines are structured with theme specific rule sets. Each rule set has its own sub chapter in the main guidelines chapter [Chapter 3](#). The rules of each rule set has at least one name space, but it is possible that more than one subdivided sets with particular name spaces are summarized to one rule set. For example the rule Set "Types and Symbols" contains sub rules sets with the IDs "CCode\_Types", "CCode\_Symbols", "CCode\_ComStackTypes" and "SpecialTypes". Each rule set with its name space is followed by a number (001, 002, 003, ...) to separate different rules of one rule set.

In the appendix a short summary over all rule sets and rules is listed (see ).

## a.2 Used Guideline Artifacts

This Guideline includes contents of the following AUTOSAR guideline artifacts:

- ▶ AR Packages Guideline (V1.4), [Chapter 9 "Rule Set: AUTOSAR Package Structure"](#), provided by Klaus Hünfeld (CDG--SMT/EMT5)
- ▶ CEL Central Elements (V1.2), [Chapter 10 "Rule Set: Central Elements"](#), provided by Volker Kairies (CDG-SMT/ESA2)
- ▶ ECU Configuration Guideline (V1.2), [Chapter 4 "Rule Set: ECU Configuration"](#), provided by Frank Neukam (CDG--SMT/ESA1)
- ▶ Pragma Concept (V1.2.5), [Chapter 3.2.3 "Abstraction of Addressing Keywords"](#) and [Chapter 3.2.4 "Abstraction of Memory Mapping"](#), provided by Gerd Grass (CDG-SMT/EMT2)
- ▶ DataDescription (V1.4) [Chapter 7 "Rule Set: Data Description"](#), provided from Shwetha Rangaswamy (RBEI/EMT5)
- ▶ DocumentReferences (V2.0) [Chapter F "References"](#), provided by Birgit Boss (DGS-EC/ESB2)

Following chapters are provided from listed colleagues but are no own guideline artifacts:

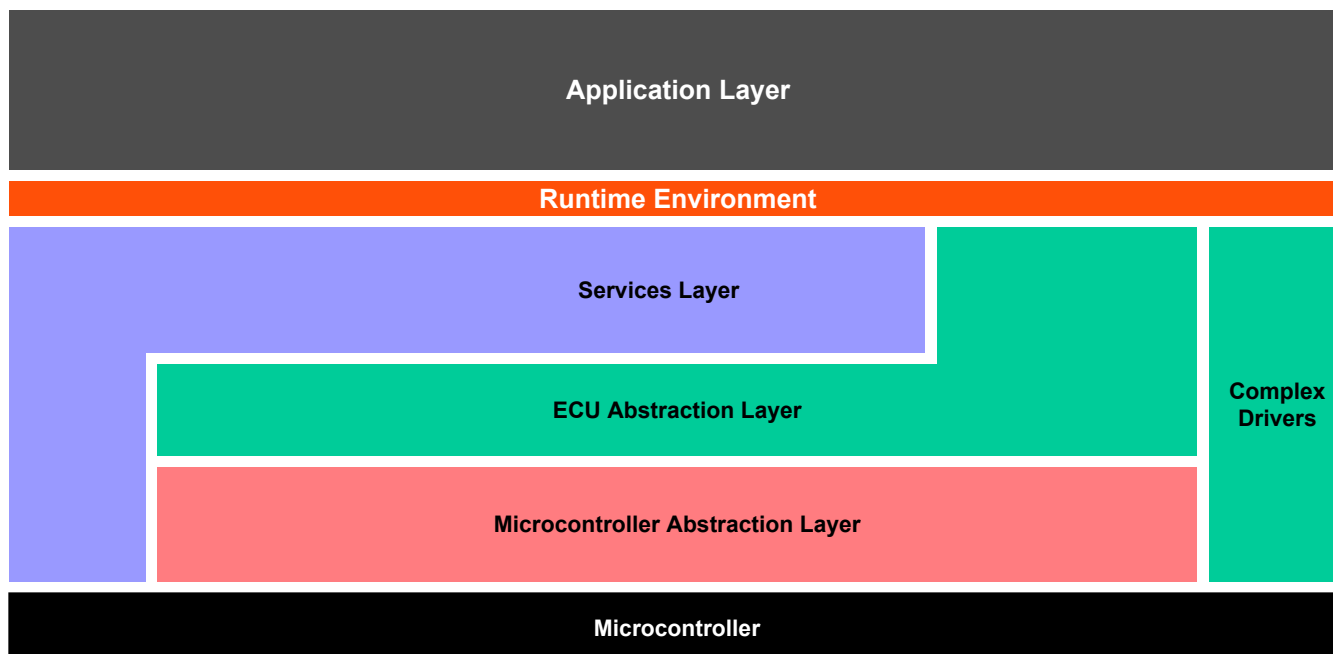
- ▶ [Chapter 3.4 "Rule Set: Naming Convention"](#), provided by Frank Böhland (CDG-SMT/ESB1)
- ▶ [Chapter 6 "Rule Set: oAW Rules"](#), provided by Marc Schreier (CDG-SMT/ESA2)
- ▶ [Chapter 3.5.7 "Code Re-entrancy"](#) and [Chapter 3.5.9 "BSW Scheduler and Exclusive Areas"](#), provided by Jörg Häcker (CDG-SMT/ESA1)
- ▶ [Chapter 3.2.10 "Ensure Usability of BSW Modules in C++ Environments"](#), provided by Jost Brachert (CDG-SMT/ESB1)
- ▶ [Chapter 8.3.2 "Scheduling of BSW via RTE"](#), provided by Peter Bolz, Christian Lemaitre (CDG-EMT/ESB2)

All other chapters and rule sets are provided from Volker Kairies (CDG-SMT/EMT5).

# 1 Introduction

Important for the kind of SW implementation is the AUTOSAR software layer model shown in [Figure 1](#). Each layer has special tasks and constraints which have to be considered in general.

Figure 1 AUTOSAR Software Layer Model



The **Microcontroller Abstraction Layer (MCAL)** provides drivers with direct access to the microcontroller and its internal peripherals. MCAL makes the higher software layers independent from the  $\mu$ C. Implementation is  $\mu$ C dependent, interfaces are standardized and  $\mu$ C independent.

APIs of the **ECU Abstraction Layer** offers access to peripherals and devices regardless of their location ( $\mu$ C internal-/external) and their connection to the  $\mu$ C (port pins, type of interface). This layer makes the higher software layers independent from the *ECU* hardware layout. Implementation is  $\mu$ C independent and interfaces are independent from the  $\mu$ C and the *ECU* hardware.

**Services Layer** provides basic services for applications and basic software modules (e.g. OS, COM, NVRAM Manager, Diagnostic services, *ECU* state manager, Watchdog, but exclusively I/O, they are located in ECU Abstraction Layer). Implementation is mostly *ECU* and  $\mu$ C independent, interfaces are completely independent.

**Complex Drivers Layer** contains special functionalities and device drivers which were not defined in AUTOSAR and are not located in one of the other software layers. Implementation and interfaces might be application,  $\mu$ C and *ECU* hardware dependent.

The **Runtime Environment (RTE)** makes the AUTOSAR SW Components independent from the mapping to a specific *ECU*. Communication mechanisms are provided (inter and/or intra *ECU*) for the application software. Implementation is generated individually for each *ECU*, but the interfaces to Application software are generally *ECU* independent.

All four SW layers below the RTE layer are named as **Base Software (BSW)** which is the main focus of these coding guidelines.

It shall be possible to create an AUTOSAR *ECU* out of modules provided as source code and modules provided as object code, even mixed. A use case of this is that some simple drivers could be provided as object code. More complex and configurable modules could be provided as source code or even generated code.

## 2 Glossary

Table 1 Glossary

Terms	Definitions
API	Application Programming Interface
AR	AUTOSAR Release
ARXML	AUTOSAR XML
ASCII	American Standard Code for Information Interchange
ASW	Application Software
AUTOSAR	AUTomotive Open System ARchitecture
BAMF	Build Action Manifest File
BSW	Basic Software
BSWMD	Basis Software Module Description
BSWMDT	Basis Software Module Description Template
C90	ISO 9899:1990 C programming language, ISO 9899, amended and corrected by ISO/IEC 9899/-COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2: 1996
cdgb	CDG Build
CLF	Classification File
component	Is used synonymously to module
CUBAS	Common UBK Basic Software
DSERAP	Dynamic SERIAL Application with Programing
ECU	Electronic Control Unit
ECUC	ECU Configuration
HIS	"Herstellerinitiative Software" – a consortium of following companies: Audi, BMW, Daimler-Chrysler, Porsche, Volkswagen
ISR	Interrupt Service Routine
MCAL	MicroController Abstraction Layer
MISRA	Motor Industry Software Reliability Association
module	Is used synonymously to component
MWE	The Modeling Workflow Engine is an extensible framework for the integration and orchestration of model processing workflows
NVRAM	Non-Volatile RAM
OS	Operation System
oAW	OpenArchitectureWare, a generator framework written in Java and now integrated in the Eclipse Modeling Project
PDU	Protocol Data Units
RTE	Runtime Environment
SDU	Service Data Unit
SWC	Software Component
SWCT	Software Component Template
TP	Transport Protocol
UBK	Germ., Unternehmensbereich Kraftfahrzeugtechnik (Automotive Technology Business Sector)
Xpand	Template for code generation (ECU Configuration)
Xtend	Template for code generation extensions (ECU Configuration)
μC	Microcontroller



## 3 Rule Set: C Coding Guidelines

### 3.1 Rule Set: C Language Rules

This rule set contains rules for basic elements of the C language. Generally all software shall be written conforming to ISO C90, but not all features and possibilities of the C language may be used. The restrictions are made to rise up quality, maintainability and readability of the code. In this context MISRA C standard is a topic which will support the avoidance of programming errors too.

Other rule sets contain also rules for C language, but these rule sets have another thematic focus. The *"Rule Set: C Language Rules"* contains all basic rules, other rule sets fulfill other technical topics.

#### 3.1.1 General C90 Language Topics

##### Rule CCode\_001: Conformity to C Standard

**Instruction** All software written in C language shall be conform to ISO/IEC 9899: 1990 (ISO C90) "Programming languages – C" amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996. Additionally some selected deviations and extensions are permitted to ensure the applicability of C language for automotive applications.

The main rule for software developed from CDG-SMT departments is to be conform to ISO C90 language and the contained set of language methods and rules. But certain deviations and extensions shall be allowed because typical automotive applications require that. On the one hand the freedom of the C language is limited (e.g. length of data types), on the other hand some extensions are helpful (e.g. C++ style comments) and they are handled uniformly by all compilers known in the automotive application. The very strict language limits of C90 are raised to C99 limits DOS/Windows style– in accordance to the recommendations given by the MISRA-C:2004 guideline itself – in order to follow the progress of the hardware and compiler's abilities. Following list shows an overview of all valid deviations and extensions:

##### 1. Assumptions about the width of C-data types

- char is 8 bit
- short is 16 bit
- int is 16 bit or 32 bit
- long is 32 bit
- If available long long is 64 bit

The given names here are the basic data types defined in C90. It shows e.g. that a char or byte is exactly 8 bits long and not 7 or 9. The data types to be used in software are defined in *"Rule Set: Types and Symbols"*.

##### 2. Assumptions about the C-language implementation

- unsigned char (resp. uint8 corresponding to [CCode\_Types\_001]) is the smallest addressable memory unit
- unsigned char (resp. uint8 corresponding to [CCode\_Types\_001]) has the smallest alignment
- Negative numbers are represented by the two's complement
- Right shift on signed operands: Arithmetic shift (not logical shift) is done when applying the right shift operator on signed variables.

- When integers are divided, the result of the "/" operator is the algebraic quotient with any fractional part discarded. If the quotient  $a/b$  is representable, the expression  $(a/b)*b + a\%b$  shall equal  $a$ . (Division according C99 standard).

Examples:

- 1)  $a = -2, b=3, a/b=0, a\%b=-2$ ;
- 2)  $a = -7, b=3, a/b=-2, a\%b=-1$ ;

### 3. C90 language limits are raised up to C99 limits

- Nesting of parentheses -- from 32 to 63
- Nesting of struct or union types -- from 15 to 63
- Number of block scope identifiers in a block -- from 127 to 511
- Number of members in struct or union -- from 127 to 1023
- Number of enumeration constants -- from 127 to 1023
- Nesting of control structures (statements) -- from 15 to 127
- Number of case labels. -- from 257 to 1023
- Nesting levels of #include "%s" -- from 8 to 15
- Nesting level of #if... -- from 8 to 63
- 60 character significance for identifiers
- Number of macro definitions simultaneously defined in one preprocessing translation unit may exceed the C99 limit of 4095

### 4. Used language extensions

- If the long long data type is available then the LL suffix for 64 bit constant definition is used (see [\[CCode\\_Types\\_004\]](#))
- Inline (encapsulated in a central macro definition) (see [\[Abstr\\_Inline\\_001\]](#))
- "//" C++ style comments (see [\[CCode\\_Comments\\_002\]](#))
- Line ends are defined as CR/LF (DOS/Windows style) (see [\[CCode\\_002\]](#))
- The basic source and execution character set is extended with "\$" (0x24) and "@" (0x40). (see [\[CCode\\_002\]](#))

## Rule CCode\_002: Character Set, Escape Sequences and Trigraphs

**Instruction** A basic set of characters and escape sequences conform to ISO C90 shall be used. Usage of trigraphs is not allowed.

A character set can have two focuses: On the one hand the character set in which source files are written (this set is called "source character set") and on the other hand the character set which is interpreted in the execution environment e.g. if strings are defined and processed (this set is called "execution character set").

For both use cases resp. focuses ISO C defines one basic character set. Additionally for the execution character set some escape sequences are defined and allowed to use. From the view of C90 special trigraphs are possible but they shall not be used.

**Hint** With trigraphs special characters within strings could be defined which are not available in character sets divergent from ASCII. A trigraph would be a sequence of 2 question marks followed by a special third character (e.g.

"??" represents a "~" (tilde) character and "??)" represents a "]""). They can cause accidental confusion with other uses of two question marks. Example: "Date should be in the form ??-??-??" would be interpreted by the compiler as: "Date should be in the form ~"]".

Because of such problems and the focus to an ASCII based character set trigraphs are not allowed to use.

For the source character set there is a small exception to ISO C90. Here tabulator characters are not allowed because they cause sometimes problems in the tool chain. Instead of a tabulator whitespace shall be used (see [\[CCode\\_Style\\_005\]](#)).

The encoding of the character set and escape sequences shall be equivalent to the standard ASCII (American Standard Code for Information Interchange) coding scheme. The encoding is (as well) only relevant for the execution character set, but it has to be defined and documented to guarantee portability for code which uses character-constants and string-literals. In following overview of basic character set and escape sequences the hexadecimal ASCII encoding values are shown inside of the brackets.

Table 2 Overview Basic Source and Execution Character Set:

<b>26 uppercase letters of the Latin alphabet:</b>	
<b>A</b> (0x41)	<b>B</b> (0x42)
<b>C</b> (0x43)	<b>D</b> (0x44)
<b>E</b> (0x45)	<b>F</b> (0x46)
<b>G</b> (0x47)	<b>H</b> (0x48)
<b>I</b> (0x49)	<b>J</b> (0x4A)
<b>K</b> (0x4B)	<b>L</b> (0x4C)
<b>M</b> (0x4D)	<b>N</b> (0x4E)
<b>O</b> (0x4F)	<b>P</b> (0x50)
<b>Q</b> (0x51)	<b>R</b> (0x52)
<b>S</b> (0x53)	<b>T</b> (0x54)
<b>U</b> (0x55)	<b>V</b> (0x56)
<b>W</b> (0x57)	<b>X</b> (0x58)
<b>Y</b> (0x59)	<b>Z</b> (0x5A)
<b>26 lowercase letters of the Latin alphabet:</b>	
<b>a</b> (0x61)	<b>b</b> (0x62)
<b>c</b> (0x63)	<b>d</b> (0x64)
<b>e</b> (0x65)	<b>f</b> (0x66)
<b>g</b> (0x67)	<b>h</b> (0x68)
<b>i</b> (0x69)	<b>j</b> (0x6A)
<b>k</b> (0x6B)	<b>l</b> (0x6C)
<b>m</b> (0x6D)	<b>n</b> (0x6E)
<b>o</b> (0x6F)	<b>p</b> (0x70)
<b>q</b> (0x71)	<b>r</b> (0x72)
<b>s</b> (0x73)	<b>t</b> (0x74)
<b>u</b> (0x75)	<b>v</b> (0x76)
<b>w</b> (0x77)	<b>x</b> (0x78)
<b>y</b> (0x79)	<b>z</b> (0x7A)
<b>10 decimal digits:</b>	
<b>0</b> (0x30)	<b>1</b> (0x31)
<b>2</b> (0x32)	<b>3</b> (0x33)
<b>4</b> (0x34)	<b>5</b> (0x35)
<b>6</b> (0x36)	<b>7</b> (0x37)
<b>8</b> (0x38)	<b>9</b> (0x39)
<b>30 graphic characters:</b>	
<b>!</b> (0x21)	<b>"</b> (0x22)
<b>#</b> (0x23)	<b>%</b> (0x25)
<b>&amp;</b> (0x26)	<b>'</b> (0x27)
<b>(</b> (0x28)	<b>)</b> (0x29)
<b>*</b> (0x2A)	<b>+</b> (0x2B)
<b>,</b> (0x2C)	<b>-</b> (0x2D)
<b>.</b> (0x2E)	<b>/</b> (0x2F)
<b>:</b> (0x3A)	<b>;</b> (0x3B)
<b>&lt;</b> (0x3C)	<b>=</b> (0x3D)
<b>&gt;</b> (0x3E)	<b>?</b> (0x3F)
<b>[</b> (0x5B)	<b>\</b> (0x5C)
<b>]</b> (0x5D)	<b>^</b> (0x5E)
<b>_</b> (0x5F)	<b>{</b> (0x7B)
<b> </b> (0x7C)	<b>}</b> (0x7D)
<b>~</b> (0x7E)	<b>Space</b> (0x20)
<b>Additional graphic characters (not defined in ISO C90 but in ASCII):</b>	
<b>\$</b> (0x24)	<b>@</b> (0x40)
<b>Escape sequences (to be used within execution character set):</b>	
<b>\a</b> (0x07)	alert
<b>\b</b> (0x08)	backspace
<b>\f</b> (0x0C)	form feed
<b>\n</b> (0x0A)	new line
<b>\r</b> (0x0D)	carriage return
<b>\t</b> (0x09)	horizontal tab
<b>\v</b> (0x0B)	vertical tab

Furthermore the editor in the development environment shall be set in such a way that DOS EOL <CR><LF> (0x0D 0x0A) is used as end of line character.

## Rule CCode\_003: Octal Constants and Escape Sequences

**Instruction** Octal constants (other than zero) and octal escape sequences shall not be used.

Any integer constant beginning with a "0" (zero) is treated as octal. So there is a danger in misinterpretation of the value of the constants. Following initializations will show few examples.

```
code[1] = 109;      /* equivalent to decimal 109 */
code[2] = 100;      /* equivalent to decimal 100 */
code[3] = 052;      /* equivalent to decimal 42  */
code[4] = 071;      /* equivalent to decimal 57  */
```

Octal escape sequences can be problematic because the inadvertent introduction of a decimal digit ends the octal escape and introduces another character.

The integer constant zero (written as a single numeric digit), is strictly speaking an octal constant, but is a permitted exception to this rule.

## Rule CCode\_004: Usage and Handling of Bit Fields

**Instruction** Bit fields are allowed but shall only be defined based on "unsigned int" or "signed int" data types. Additionally a bit field based on signed type shall be at least 2 bits long. Bit fields shall not be used in APIs and inside of unions. The size of bit fields is limited to 16 bits. Finally no certain order of the bits inside the bit field shall be assumed.

Usage of other data types than "unsigned int" or "signed int" as base for bit fields is implementation-defined and is not allowed. Order of bits within a bit field in memory is not guaranteed. Therefore no certain order of the bits shall be assumed. The size of a bit fields is limited to 16 bits to be portable.

```
struct Bitfield
{
    unsigned int    firstbit: 1;
    unsigned int    secondbit: 1;
    unsigned int    bitgroup: 3;
} Bitfield_Hugo;
```

## Rule CCode\_005: Packing of Bit Fields

**Instruction** If it is relied on, the implementation-defined behaviour and packing of bit fields shall be documented in the chapter for integration in the product documentation.

It shall be noticed, that [\[CCode\\_004\]](#) is violated if it is relied on the implementation-defined behaviour and packing of bit fields. But in this case the behaviour shall be documented in the chapter for integration in the product documentation.

## 3.1.2 MISRA and HIS Metrics Conformity

### Rule CCode\_MisraHIS\_001: Conformity to MISRA Standard and HIS Metrics

**Instruction** All software modules written in C language shall conform to the accepted set of rules of the MISRA C Standard and shall be conform to the accepted set of HIS metrics.

AUTOSAR defines that all software shall be conform to the HIS subset of the MISRA C Standard. HIS has defined that all rules of the MISRA-C:2004 has to be followed. Derivations are allowed only in exceptional cases. Each derivation has to be clarified and has to be documented in a written form. This clarification and documentation is done by a working group. Within these coding guidelines all accepted MISRA rules are implemented. The derivations from MISRA are documented in a special derivation documentation outside of this guidelines. The accepted rules and the defined derivations are communicated to the customers and OEMs. Therefore it is necessary to fulfill the accepted rules as good as possible. This will avoid discussions with our customers and OEMs if they check the SW regarding the accepted MISRA rule set.

The MISRA rules and the complete coding guidelines of this document will help to provide software with a high quality. Especially the MISRA rules support the portability of software, the maintainability, the error avoidance and the safety.

Not all MISRA rules can be checked by a tool or the check is not fully implemented. In a code review all rules have to be checked and considered.

Which rules of these coding guidelines are derived from the MISRA C Standard (and others are derived from requests from AUTOSAR) can be found in the *"Rules Derivation"* list in the appendix.

The HIS Software Metrics are the basis for efficient project and quality management. With HIS software metrics statements can be made about the quality of the software product and the software development process. The accepted set of HIS Metrics are listed in the "*HIS Metrics Overview*" list in the appendix. Derivations of the HIS metrics are allowed only in exceptional cases and has to be documented.

## Rule CCode\_MisraHIS\_002: Commendation of MISRA Violations in C Code

**Instruction** In technically reasonable, exceptional cases MISRA violations are permissible and shall be documented with a comment within the source file.

Only in technically reasonable and exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the source code. It is important to document the rationale why the MISRA rule is violated and not that a MISRA rule is violated.

The comment shall be placed right above the line of code which causes the violation and have the following syntax:

```
/* MISRA RULE X.Y VIOLATION: Reason why the MISRA rule could not be followed in this special case */  
X.Y represents the MISRA rule number.
```

It is clear that only those MISRA rules can be commented which are generally accepted and active. In the technical working group for coding guidelines some MISRA rules have been rejected. Such rules are not visible within this coding guidelines and they have not to be commented out by default.

The MISRA rule number is different to the rule number of this coding guidelines. The reason for this behaviour is that all rules within this guideline are located to the existing rule sets which are independent from the derivation of the rules. To get an overview of the derivation of the rules and to find which coding rule is derived from which MISRA rule the list "*Rules Derivation*" is given in the appendix. But no panic, in most cases an exception of a MISRA rule is found by a static check (QAC Tool). Here the corresponding MISRA rule is stated.

## Rule CCode\_MisraHIS\_003: Commendation of MISRA Violations for a Complete Module

**Instruction** In technically reasonable, exceptional cases component wide MISRA rules violations are permissible and shall be documented in the chapter for integration in the product documentation.

Only in technically reasonable and exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented in the chapter for integration in the product documentation. It is important to document the rationale why the MISRA rule is violated and not that a MISRA rule is violated.

It is important, that any restriction in the usage of the component is clearly documented. E.g. bit field members are used, which has a size greater than 16 bits which is violation of *[CCode\_004]*.

## Rule CCode\_MisraHIS\_004: Commendation HIS Metric Violations in C Code

**Instruction** In technically reasonable, exceptional cases HIS Metrics limit violations are permissible and shall be documented with a comment within the source file.

Only in technically reasonable and exceptional cases HIS Metrics limit violations are permissible. Such violations shall be clearly identified and documented within comments in the source code. It is important to document the reason why the HIS Metric cannot be fulfilled and not that the HIS Metric is not fulfilled.

The comment shall be placed right above the line of code which causes the violation and have the following syntax:

```
/* HIS METRIC XY VIOLATION: Reason why the HIS Metric limit is exceeded in this special case */  
XY represents the name of the HIS Metric. The name can be found in the "HIS Metrics Overview" list in the appendix.
```

Modifications in code which are done to fulfill the HIS Metrics can sometimes cause a higher resource consumption. This could be a reason to comment a HIS Metric instead of having enlarged code. But it should be concerned that no quality problem of the software could occur when a HIS Metric limit is not kept. A violation could happen if the code is worsen

(quality, readability, maintainability). But finally it is necessary to keep the HIS Metrics as best as possible because some OEMs take a look to that metrics and want to have statements why a HIS Metric is not fulfilled.

### 3.1.3 Initializations

#### Rule CCode\_Inits\_001: Initialization of Variables

**Instruction** All variables shall have been assigned a value before they are being used.

The intent of this rule is that all variables shall have been assigned with explicit values before they are read. Otherwise the variable can contain a unspecified value. An initialization has to be done but this does not necessarily require that the initialization has to be done at declaration time. There can be made a distinction between following types of variables:

► Static and global (with external linkage) variables:

According to the ISO C90 standard, variables with static storage duration are automatically initialized to zero by default, unless explicitly initialized. But this behaviour is not implemented in embedded environments without additional mechanisms. A static or global variable has to be located explicitly to a memory section where an initialization to zero is made. To do that the mechanism described in "*Abstraction of Memory Mapping*" has to be used. If a static or global variable is not located to an automatic initialized memory section an explicit initialization of that variable has to be done by the developer (e.g. using an initialization function or initialization process).

► Variables defined inside a function or a block:

Such variables have an automatic storage duration and according to the ISO C90 standard such variables are not initialized automatically and can contain any unpredictable value if not initialized. An initialization has to be done explicitly at all times before such variables are used within the function. Such variables are normally located to registers of the processor.

Examples:

```
#define MYMODULE_START_SEC_VAR_INIT_32
#include "MyModule_MemMap.h"
static uint32 staticvar1_u32 = 10L; /* Explicitly initialized variable */
...
#define MYMODULE_STOP_SEC_VAR_INIT_32
#include "MyModule_MemMap.h"

#define MYMODULE_START_SEC_VAR_CLEARED_32
#include "MyModule_MemMap.h"
static uint32 staticvar2_u32; /* Variable will be initialized with 0 */
...
#define MYMODULE_STOP_SEC_VAR_CLEARED_32
#include "MyModule_MemMap.h"

void MyModule_Func(void)
{
    uint32 localvar1_u32; /* This variable has to be initialized */
    uint32 localvar2_u32; /* For function local variable no memory */
    uint32 localvar3_u32; /* mapping concept has to be used. */
    ...

    localvar2_u32 = localvar1_u32; /* Not OK: Uninitialized variable is used */
    ExtModule_Func(localvar1_u32); /* Not OK: External function called with */
    ... /* uninitialized variable */

    localvar1_u32 = 0x12345L; /* Now myvar1_u32 is initialized */
    ExtModule_Func(localvar1_u32); /* OK, local variable is initialized before */
    ...

    localvar3_u32 = staticvar1_u32; /* OK, staticvar1_u32 is initialized with 10 */
}
```

```

05      ...

      localvar3_u32 = staticvar2_u32; /* OK, staticvar2_u32 is at least set to 0 */
      ...
    }

```

All variables with external linkage and all variables with the storage class specifier static, which are not initialized at startup, shall be clearly documented in the chapter for integration in the product documentation. According to the ISO C90 standard such variables are initialized at startup to zero if there is no explicit initialization. But nevertheless, some software relies on, that in exceptional cases these variables will not be initialized, e.g. to retain information after a SW-reset. This exceptional requirement has to be documented in the chapter for integration in the product documentation. It shall be noticed, that this is in this case a violation of MISRA rule 1.1.

## Rule CCode\_Inits\_002: Initialization of Enumerators

**Instruction** In an enumerator list, the '=' construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.

If an enumerator list is given with no explicit initialisation of members, then C allocates a sequence of integers starting at 0 for the first element and increasing by 1 for each subsequent element. An explicit initialisation of the first element, as permitted by the above rule, forces the allocation of integers to start at the given value. When adopting this approach it is essential to ensure that the initialisation value used is small enough that no subsequent value in the list will exceed the *int* storage used by enumeration constants. Explicit initialisation of all items in the list, which is also allowed, prevents the mixing of automatic and manual allocation, which is error prone. However it is then the responsibility of the programmer to ensure that all values are in the required and allowed range, and that values are not unintentionally duplicated.

To be portable enumerator values shall only be defined in a range of sint16. sint16 is the default defined in C90. With enumerators negative values could also be used even if a developer usually expects only positive values.

Examples:

<pre> 40 typedef enum     {         MYCODE_OK_E,         MYCODE_NOT_OK_E,         MYCODE_BUSY_E     } MyModule_Enum_ten; /* OK because automatic initialization */ /* is used. */ </pre>	<pre> typedef enum {     MYCODE_OK_E = 3,     MYCODE_NOT_OK_E = 4,     MYCODE_BUSY_E = 5 } MyModule_Enum_ten; /* OK because all elements are */ /* initialized */ </pre>
<pre> 50 typedef enum     {         MYCODE_OK_E = 5,         MYCODE_NOT_OK_E,         MYCODE_BUSY_E     } MyModule_Enum_ten; /* OK because only first element is */ /* initialized with a value != 0 */ </pre>	<pre> typedef enum {     MYCODE_OK_E,     MYCODE_NOT_OK_E = 2,     MYCODE_BUSY_E } MyModule_Enum_ten; /* Not OK because second element is */ /* initialized, but not first one */ </pre>
<pre> 55 typedef enum     {         MYCODE_OK_E = -1,         MYCODE_NOT_OK_E = -2,         MYCODE_BUSY_E = -3     } MyModule_Enum_ten; /* OK because all elements are */ /* initialized */ </pre>	<pre> typedef enum {     MYCODE_OK_E = -1,     MYCODE_NOT_OK_E,     MYCODE_BUSY_E } MyModule_Enum_ten; /* OK because only first element is */ /* initialized with a value != 0 */ /* Hint: MYCODE_NOT_OK_E will get 0, */ /* MYCODE_BUSY_E will get 1 */ </pre>

Additional information about the location of enumerators:

The size of an enumeration variable *depends on the used compiler settings/environment*. For example:



```

05 typedef enum
{
    RBA_GTM_SYNC_PWM_MODE_DSBL_E,
    RBA_GTM_SYNC_PWM_MODE_ASYNC_E,
    RBA_GTM_SYNC_PWM_MODE_SYNC_E
10 } rba_gtm_sync_pwm_mode_ten;

rba_gtm_sync_pwm_mode_ten test_cd_sync_pwm_mode_en = RBA_GTM_SYNC_PWM_MODE_DSBL_E;

```

Compiled for two different machine types the size varies between one byte and four bytes (extraction from \*.map file)

```

15 IFX: 0x7000ef0b 0x7000ef0b    1 g test_cd_sync_pwm_mode_en
JDP: 0x40007890 0x40007893    4 g test_cd_sync_pwm_mode_en

```

When changing the enumeration definition to an init value of 300

```

20 typedef enum
{
    RBA_GTM_SYNC_PWM_MODE_DSBL_E = 300,
    RBA_GTM_SYNC_PWM_MODE_ASYNC_E,
    RBA_GTM_SYNC_PWM_MODE_SYNC_E
} rba_gtm_sync_pwm_mode_ten;

```

The size of the variable changes to two bytes (extraction from \*.map file):

```

IFX: 0x70002bce 0x70002bcf    2 g test_cd_sync_pwm_mode_en

```

This should be kept in mind when placing enumeration variables within structures.

## 3.1.4 Expressions

### Rule CCode\_Expr\_001: Limited Dependence

**Instruction** Limited dependence should be placed on C's operator precedence rules in expressions.

In addition to the use of parentheses to override default operator precedence, parentheses should also be used to emphasise it. It is easy to make a mistake with the rather complicated precedence rules of C, and this approach helps to avoid such errors, and helps to make the code easier to read. However, do not add too many parentheses so as to clutter the code and make it unreadable.

Examples:

```

45 /* Here no parentheses are needed: */
x = a + b;
x = a * -1;

/* Here parentheses are needed to clarify context: */
50 x = a * b - c + d;
/* Some possible variants: */
x = (a * b) - (c + d);
x = a * (b - (c + d));
55 x = ((a * b) - c) + d;

```

### Rule CCode\_Expr\_002: Value of Expressions

**Instruction** The value of an expression shall be the same under any order of evaluation that the standard permits.

Apart from a few operators (notably the function call operator (), &&, ||, ?: and , (comma)) the order in which subexpressions are evaluated is unspecified and can vary. This means that no reliance can be placed on the order of evaluation of sub-expressions, and in particular no reliance can be placed on the order in which side effects occur. Those points in the evaluation of an expression at which all previous side effects can be guaranteed to have taken place are called 'sequence points'.



Note that the order of evaluation problem is not solved by the use of parentheses, as this is not a precedence issue.

Examples:

► Increment and decrement operators

```
x = b[i] + i++;
```

This will give different results depending on whether `b[i]` is evaluated before `i++` or vice versa. The problem could be avoided by putting the increment operation in a separate statement.

```
x = b[i] + i;
i++;
```

► Functions

The order of evaluation of function arguments is unspecified

```
x = func(i++, i);
```

This will give different results depending on which of the function's two parameters is evaluated first. The problem could be avoided by using a temporary variable.

```
j = i++;
x = func(j, i);
```

Functions may have additional effects when they are called (e.g. modifying some global data). Dependence on order of evaluation could be avoided by invoking the function prior to the expression that uses it, making use of a temporary variable for the value.

```
x = f(a) + g(a);
```

Could be written as:

```
x = f(a);
x += g(a);
```

► Nested assignment statements

Assignments nested within expressions cause additional side effects. The best way to avoid any change of this leading to a dependence on order of evaluation is to not embed assignments within expressions. Following examples are not recommended:

```
x = y = y = z / 3;
x = y = y++;
```

## Rule CCode\_Expr\_003: Usage of Sizeof Operator

**Instruction** The `sizeof` operator shall not be used on expressions that contain side effects.

A possible programming error in C is to apply the `sizeof` operator to an expression and expect the expression to be evaluated. The operations that cause side-effects are modifying an object, accessing a volatile object and calling a function that does any of those operations, which cause changes in the state of the execution environment of the calling function. However the expression is not evaluated: `sizeof` only acts on the type of the expression. To avoid this error, `sizeof` shall not be used on expressions that contain side effects, as the side effects will not occur. `sizeof()` shall only be applied to an operand which is a type or an object.

An expression which reads a volatile object has a side effect by default. The `sizeof` operator applied on a volatile object is explicitly allowed because the `sizeof` operator acts here again on the type of the volatile object. The implicit side effect of the volatile object has no negative effect on the execution of the `sizeof` operator.

Examples:

```
uint16 Var1_u16;
uint16 Var2_u16;
uint16 Var3_u16;
```

```
Var1_u16 = sizeof(Var2_u16);    /* OK. The size of the uint16 variable is returned */
```

```

05 Var2_u16 = sizeof(Var3_u16++);    /* Not OK. sizeof over an expression    */
   Var3_u16 = sizeof(MyFunc());    /* Not OK. sizeof over a function call */

```

## Rule CCode\_Expr\_004: Logical and Conditional Operators without Side Effects

**Instruction** The right hand operand of a logical '&&' or '||' operator and of a conditional operator '? : ' shall not contain side effects.

There are some situations in C code where certain parts of expressions may not be evaluated. If these sub-expressions contain side effects than those side effects may or may not occur, depending on the values of other sub expressions. The operators which can lead to this problem are &&, || and ? :.

### ► Logical operators '&&' and '||':

The evaluation of the right-hand operand is conditional on the value of the left-hand operand. The conditional evaluation of the right hand operand of one of the logical operators can easily cause problems if the programmer relies on a side effect occurring.

Examples:

```

25 if (ishigh && (x == I++))    /* Not OK because I is modified    */
   if (ishigh && (x == f(x)))    /* OK but only if f(x) has no side effects */

```

### ► Conditional operator '? : ':

The ? : operator is specifically provided to choose between two sub-expressions, and is therefore less likely to lead to mistakes. Either the second or third operands are evaluated but not both. Therefore the conditional operator may only be used in the following way:

```
Conditional expression = (comparison) ? value1: value2;
```

value1 and value2 must be simple expressions like a variable name or an explicit value. Function calls or function like macros are not allowed. For such cases if/else expressions shall be used.

## Rule CCode\_Expr\_005: Logical Operator Operands as Primary Expressions

**Instruction** The operands of a logical '&&' or '||' operators shall be primary-expressions or extra parentheses are recommended.

'Primary expressions' are essentially either a single identifier, a constant, a literal string or a parenthesised expression. The effect of this rule is to require that if an operand is other than a single identifier or constant then it shall be parenthesised. Parentheses are important in this situation both for readability of code and for ensuring that the behaviour is as the programmer intended. Where an expression consists of either a sequence of only logical && or a sequence of only logical ||, extra parentheses are not required.

Examples:

```

55 if ((x > c1) && (y > c2) && (z > c3))    /* OK because only && is used    */
   if ((x > c1) && (y > c2) || (z > c3))    /* Not OK because of a mix of && and || */
   if ((x > c1) && ((y > c2) || (z > c3)))    /* OK because of extra parentheses */

```

## Rule CCode\_Expr\_006: Usage of Logical Operators

**Instruction** The operands of logical operators ('&&', '||' and '!') should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than '&&', '||', '!', '=', '==', '!=' and '? : '.

The logical operators '&&', '||' and '!' can be easily confused with the bitwise operators '&', '|' and '~'. It is only allowed to use boolean values operator in logic expressions but never in arithmetical expressions. For boolean variables following arithmetic or bitwise operators shall not be used: +, ++, -, --, \*, /, <<, >>, ~.

Examples:

```

res_s16 = (a_s16 < b_s16) & (c_s16 > d_s16);
/* Not OK: (a_s16 < b_s16) results in a boolean type; '&' is a bitwise operator */
/* A correct logical expression is: res_b = (a_s16 < b_s16) && (c_s16 > d_s16); */

res_s16 = (a_s16 < b_s16) + (c_s16 > d_s16);
/* Not OK: (a_s16 < b_s16) results in a boolean type; '+' is an arithmetic operator */

res_s16 = (a_s16 + b_s16) || (c_s16 + d_s16);
/* Not OK: Result of '+' is not a boolean type; '||' is a logical operator */

res_s16 = !(a_s16 | b_s16);
/* Not OK: (a_s16 | b_s16) is a bitwise expression; '!' is a logical operator */

```

## Rule CCode\_Expr\_007: Type of Bitwise Operators

**Instruction** Bitwise operators shall not be applied to operands whose type is a signed integer.

Bitwise operations (~, <<, <=, >>, >=, &, &=, ^, ^=, | and |=) are not normally meaningful on signed integers. Problems can arise if, for example, a right shift moves the sign bit into the number, or a left shift moves a numeric bit into the sign bit.

There are as well exceptions to this rule, e.g. if a signed variable is used in an expression with a mask and the mask does not change the sign bit.

## Rule CCode\_Expr\_008: Usage of Shift Operator

**Instruction** The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.

If, for example, the left hand operand of a left-shift or right-shift is an unsigned 16-bit integer, then it is important to ensure that this is shifted only by a number between 0 and 15 inclusive.

There are various ways of ensuring this rule is followed. The simplest is for the right hand operand to be a constant (whose value can then be statically checked). Use of an unsigned integer type will ensure that the operand is non-negative, so then only the upper limit needs to be checked (dynamically at run time or by review). Otherwise both limits will need to be checked.

Examples:

```

var_u8 <= 7; /* OK. Shift is inside borders of uint8 */
var_u8 = (uint8)(var_u8 << 9); /* Not OK. Shift is outside of uint8 */
var_u16 = (uint16)((uint16)var_u8 << 9); /* OK. var_u8 is converted to uint16 */

```

## Rule CCode\_Expr\_009: Usage of Unary Minus Operator

**Instruction** The unary minus operator shall not be applied to an expression whose type is an unsigned integer.

Applying the unary minus operator to an expression of an unsigned integer type generates a result of an unsigned integer type respectively and is not a meaningful operation. Applying unary minus to an operand of smaller unsigned integer type may generate a meaningful signed result due to integral promotion, but this is not good practice.

Examples:

```

var_u16 = -(var1_u8 + var2_u8); /* Not OK */
var_u8 = -var_u8; /* Not OK */

```

## Rule CCode\_Expr\_010: Usage of Comma Operator

**Instruction** The comma operator shall not be used, except in the control expression of a "for" loop and within macros.

Use of the comma operator is generally detrimental to the readability and maintainability of code, and the same effect can be achieved by other means. For example creation of two statements avoids at all times the use of a comma operator. Within macros or for statements a comma operator is sometime needed because of technical reasons. Readability and maintainability is here a smaller problem.

Examples:

```
var1_u8 = (var2_u8++, var3_u8++);           /* Not OK because code is abstruse */

var2_u8++;                                /* OK because separate statements are clearer */
var1_u8 = var3_u8++;

#define MYMODULE_MACRO(Val, Struct) ((Struct)->X = 0, (State)->Y = (Val)) /* OK */

for (I = 0, j = 0; I++; I < 100)           /* OK, restriction within for statement */
...
```

## Rule CCode\_Expr\_011: Usage of Constant Unsigned Integer Expression

**Instruction** Evaluation of constant unsigned integer expressions should not lead to wrap-around.

Unsigned integer expressions do not strictly overflow. A wraparound will occur if the result of the unsigned integer expression is greater than the maximum value an unsigned integer variable can hold. As well subtracting unsigned integer type constants will never produce a negative result. Here a wrap-around will occur too, to produce another positive value. Therefore any instance of an unsigned integer constant expression wrapping around is likely to indicate a programming error.

Examples:

```
var_u32 = 0x1 - 0x2;                      /* Not OK: Result is implicitly negative -> wraparound */

var_u32 = 0xFFFFFFFF + 0x2;              /* Not OK: Results in a wraparound because result of
/* expression is greater than maximum of uint32 */

var_u32 = 0xFFFF + 0x2;                  /* Not OK: The difference to the example before is that
/* both constants represent 16 bit values.
/* Here the calculation of the "+" expression is done in
/* in an "unsigned int" context. With a compiler where
/* "int" is 16 bit the result wraps around to value 0x1.
/* If "int" has 32 bit the result is 0x10001.
/* If 0x10001 is expected and the calculation shall be
/* independent from the representation of the "int" type
/* the expression should be written in following form:
/* var_u32 = (uint32)0xFFFF + 0x2;
```

## Rule CCode\_Expr\_012: Usage of Increment and Decrement Operator

**Instruction** The increment '++' and decrement '--' operators should not be mixed with other operators in an expression.

The use of increment and decrement operators in combination with other arithmetic operators is not recommended because:

- ▶ It can significantly impair the readability of the code
- ▶ It introduces additional side effects into a statement with the potential for undefined behaviour

It is safer to use these operations in isolation from any other arithmetic operators. It is the intention of the rule that when the increment or decrement operator is used, it should be the only side-effect in the statement.

For example a statement such as the following is not compliant:

```
var1_u8 = ++var2_u8 + var3_u8--;
```

The following sequence is clearer and therefore safer:

```
++var2_u8;  
var1_u8 = var2_u8 + var3_u8;  
var3_u8--;
```

Together with pointer increment and access to arrays the usage of increment and decrement operators may result in more efficient code. In that case an upcoming MISRA violation can be commented out.

Examples:

```
var1_u8 = array_au8[index_u8++];    /* OK */  
array_au8[index_u8]++;             /* OK */  
(*ptr_u8)++;                      /* OK */
```

## Rule CCode\_Expr\_013: Value of Floating Complex Expression

**Instruction** The value of a complex expression of floating type shall only be cast to a floating type which is narrower or of the same size.

If a cast is to be used on any complex expression, the type of cast that may be applied is severely restricted. Conversions on complex expressions are often a source of confusion and it is therefore wise to be cautious. In order to comply with these rules, it may be necessary to use a temporary variable and introduce an extra statement.

Examples:

```
float32 Var1_f32;  
float32 Var2_f32;  
float64 Var1_f64;  
float64 Var2_f64;  
...
```

```
Var1_f32 = (float32)(Var1_f64 + Var2_f64);    /* OK because of cast to narrower type */  
Var1_f64 = (float64)(Var1_f32 + Var2_f32);    /* Not OK because of cast to wider type */
```

## Rule CCode\_Expr\_014: Implicit Integer Type Conversions

**Instruction** Implicit type conversions without a cast of integer types shall only be done from smaller to bigger types where no information is lost.

Generally, if only the following listed implicit type conversions are used, it is guaranteed that there is no loss of information:

- ▶ uint8 to uint16/sint16/uint32/ sint32/uint64/sint64
- ▶ sint8 to sint16/sint32/sint64
- ▶ uint16 to uint32/sint32/uint64/sint64
- ▶ sint16 to sint32/sint64
- ▶ uint32 to uint64/sint64
- ▶ sint32 to sint64

The value of the smaller type always fits into the larger type by keeping also its signedness. Only for this cases implicit conversion is allowed.

Examples:

```
uint8 Var_u8;  
uint16 Var_u16;  
uint32 Var_u32;  
sint32 Var_s32;
```

```

05 Var_u16 = Var_u8;      /* OK: Value range of uint16 variable involves */
                          /* the range of an uint8 variable */
Var_s32 = Var_u16;      /* OK: Value range of sint32 variable involves */
                          /* the range of an uint16 variable */
10 Var_u32 = Var_s32;    /* Not OK: uint32 contains only positive values, */
                          /* sint32 cannot converted to an uint32 */

```

## Rule CCode\_Expr\_015: Explicit Integer Type Casts

**Instruction** Explicit type casts of integer types shall only be done if an explicit conversion to a narrower type is intended.

The developer is responsible to ensure that no information is lost unintended.

Examples:

```

20 uint8 Var_u8;
   uint16 Var_u16;

Var_u16 = 100;
25 Var_u8 = (uint8)Var_u16;    /* OK: Value of Var_u16 is in range of uint8 */

Var_u16 = 1000;
Var_u8 = (uint8)Var_u16;    /* Not OK: Value of Var_u16 is outside of range of uint8 */

```

## 3.1.5 Control Statement Expressions

### Rule CCode\_Control\_001: Boolean Expressions

**Instruction** Assignment operators shall not be used in expressions that yield a Boolean value.

No assignments are permitted in any expression which is considered to have a Boolean value. This precludes the use of both simple and compound assignment operators in the operands of a Boolean-valued expression. However, it does not preclude assigning a Boolean value to a variable.

If assignments are required in the operands of a Boolean-valued expression then they shall be performed separately outside of those operands. This helps to avoid getting '=' and '==' confused, and assists the static detection of mistakes.

Examples:

```

if ((x_u16 = y_u16) == 0)    /* Not OK because (x_u16 = y_u16) results in a */
...                          /* boolean context */

```

*It is better and conforms to the rule to write it in following way:*

```

x_u16 = y_u16;              /* OK because assignment is separated and if */
if (x_u16 == 0)             /* condition has no boolean context any more */
...

```

**Hint** It is a typical pitfall in C that within an if clause a "=" can be written instead of a "==". To have a check by the compiler it is a proposal to write an if-clause in the following form:

```
if (0 == x_u16)
```

The compiler will create an error message if a single "=" is written instead of a "==".

### Rule CCode\_Control\_002: Value Test Expression

**Instruction** Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.

Where a data value is to be tested against zero then the test should be made explicit. The exception to this rule is when data represents a Boolean value, even though in C this will in practice be an integer. This rule is in the interests of clarity, and makes clear the distinction between integers and logical values.

Keep in mind that the base type boolean has only two states, TRUE and FALSE, while TRUE expands to the integer constant 1, FALSE expands to the integer constant 0 (see [\[CCode\\_Symbols\\_001\]](#)). Take care of correct usage of type boolean, especially when testing for TRUE.

Examples for non-boolean context:

```
if (x_u32 != 0)      /* OK, test if x_u32 is non zero          */
if (x_u32)          /* Not OK, explicit test value is missing */
```

Examples for boolean context:

```
boolean var_b;

/* Query for "TRUE": */
if (var_b)      or      if (var_b != FALSE)  /* OK          */
                  if (var_b == TRUE)       /* Not OK       */

/* Query for "FALSE": */
if (!var_b)     or      if (var_b == FALSE)  /* OK          */
```

### Rule CCode\_Control\_003: Test of Floating-Point Expressions

**Instruction** Floating-point expressions shall not be tested for equality or inequality. A check for equality or inequality to zero is allowed.

The inherent nature of floating-point types is such that comparisons of equality will often not evaluate to true even when they are expected to. In addition the behaviour of such a comparison cannot be predicted before execution, and may well vary from one implementation to another.

The recommended method for achieving deterministic floating-point comparisons is to write a library that implements the comparison operations. But up to now such a library is not available. Clearly this restricts the usage of the type float, but currently it is rarely used in BSW.

Examples:

```
float32 x_f32, y_f32;

if (x_f32 == y_f32)      /* Not OK because of float comparison */
...
```

*An indirect test is equally problematic and is also forbidden by this rule:*

```
if ((x_f32 <= y_f32) && (x_f32 >= y_f32))  /* Not OK too */

if (x_f32 == 0.0f)      /* OK, comparison to zero is allowed */
```

**Hint** It is planned that a service interface for the comparison of float numbers is provided. However, the discussion about such an interface is still ongoing in AUTOSAR working groups and thus the interface will be earliest provided with AR4.1.1. In the meanwhile the checks have to be done without calling such an interface and the corresponding warning in the check tool has to be commented. A verification that the float check is working correctly is required. Since the float comparison is implementation-dependent and may show different behaviour on different machines, the code has to be cross-checked for different compilers as well as controllers.

## Rule CCode\_Control\_004: For Loop Expression

**Instruction** The three expressions of a for statement shall be concerned only with loop control and shall not contain any objects of floating type.

The three expressions of a for statement shall be used only for these purposes:

- ▶ First expression – Initialising the loop counter
- ▶ Second expression – Shall include testing the loop counter, and optionally other loop control variables
- ▶ Third expression – Increment or decrement of the loop counter

Following options are allowed:

- ▶ all three expressions shall be present
- ▶ the second and the third expression shall be present with prior initialization of the loop counter
- ▶ all three expressions shall be empty for a deliberate infinite loop

Floating-point variables shall not be used for this purpose. Rounding and truncation errors can be propagated through the iterations of the loop as in context of [\[CCode\\_Control\\_003\]](#). For example the number of times the loop is performed may vary from one implementation to another, and may be unpredictable.

In addition it is allowed that an integer pointer is used as loop counter instead of an integer counter. An use case of this form is e.g. when a for statement loops over a data base.

Examples:

```

for (I = 0; I < 10; I++)           /* OK, classic form of a for loop          */
...
for (ptr=start; ptr != NULL; ptr=ptr->next) /* OK, all three expressions are used */
...
for (;;)                          /* OK, infinite loop                    */
...
for (x_f32 = 0.0F; x_f32 < 10.0f; x_f32++) /* Not OK because of floating types */
...
for (i_u8 = 0; ++i_u8 < 10; ++i_u8)      /* Not OK, loop counter is incremented twice */
...
for (x_u8 = 0; i_u8 < 10; i_u8++)       /* Not OK, if i_u8 isn't initialized before the loop */

```

## Rule CCode\_Control\_005: For Loop Iteration

**Instruction** Numeric variables being used within a "for" loop for iteration counting shall not be modified in the body of the loop.

Loop counters shall not be modified in the body of the loop. However other loop control variables representing logical values may be modified in the loop, for example a flag to indicate that something has been completed, which is then tested in the for statement.

Example:

```

boolean flag_b = TRUE;
uint8 i_u8;

for (i_u8 = 0; (i_u8 < 5) && (flag_b != FALSE); i_u8++)
{
    ...
    flag_b = FALSE;           /* OK, additional logical value for early termination */
    /* of the loop -> checked in the loop body                    */
    i_u8 = i_u8 + 3;          /* Not OK, because of altering of the loop counter */
}

```



## 3.1.6 Control Flow

### Rule CCode\_CntrFlow\_001: Unreachable Code

**Instruction** There shall be no unreachable code.

This rule refers to code which cannot be reached under any circumstances, and which can be identified as such at compile time. Code that can be reached but may never be executed is excluded from the rule (e.g. defensive programming code). A portion of code is unreachable if there is no control flow path from the relevant entry point to that code. For example there is code behind a break statement in a case block of a switch-case. Or default state in a switch-case is sometimes not executed but needed for defensive programming.

Code which is excluded by pre-processor directives is not present following pre-processing, and is therefore not subject of this rule.

Finally uncalled services of a library or module are not considered by this rule.

### Rule CCode\_CntrFlow\_002: Duplication of Code

**Instruction** The duplication of code should be avoided.

This rule will help to avoid bugs during maintenance.

Example: A module contains 4 code segments which are equal. During maintenance of the module 3 of them have been updated, 1 has been forgotten -> which is an obvious bug.. The use of sub functions avoids this problem and fulfills this rule.

### Rule CCode\_CntrFlow\_003: Non-Null Statements

**Instruction** All non-null statements shall either a) have at least one side-effect however executed, or b) cause control flow to change.

Any statement (other than a null statement) which has no side-effect and does not result in a change of control flow will normally indicate a programming error, and therefore a static check for such statements shall be performed. For example, the following statements do not necessarily have side-effects when executed:

```
x >= 3u;          /* Not OK, x is compared to 3, the answer is discarded */
```

### Rule CCode\_CntrFlow\_004: Null Statement

**Instruction** Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.

Null statements should not normally be deliberately included, but where they are used they shall appear on a line by themselves. White-space characters may precede the null statement to preserve indentation. If a comment follows the null statement then at least one white-space character shall separate the null statement from the comment. The use of a white-space character to separate the null statement from any following comment is required because it provides an important visual cue to reviewers. Following this rule enables a static checking tool to warn about null statements appearing on a line with other text, which would normally indicate a programming error.

Example:

```
while ((port & 0x80) == 0)    /* port has to be volatile          */
{
    ;                        /* OK: wait for pin          */
    /* wait for pin */ ;      /* Not OK, comment before ; */
    ;/* wait for pin */      /* Not OK, no white-space char after ; */
}
```

```
05 ;I++;          /* Not OK, first ; is not in a single line */  
I++;          /* Not OK, because of a typing error? */  
if (n == 1);  /* Not OK, line makes no sense */
```

## Rule CCode\_CntrFlow\_005: Goto and Continue

**Instruction** The 'goto' and the 'continue' statements shall not be used. Additionally goto labels shall not be defined.

Any program that uses a 'goto' or 'continue' statement can be rewritten to give the same effect without those statements.

## Rule CCode\_CntrFlow\_006: Iteration Statement

**Instruction** For any iteration statement there shall be at most one break statement used for loop termination.

This rule is in the interests of good structured programming. One break statement is allowed in a loop but it need not to be used. A break within a loop can be used for optimal coding. More than one break within a loop is not allowed.

## Rule CCode\_CntrFlow\_007: If ... Else

**Instruction** All if ... else if constructs shall be terminated with an else clause.

This rule applies whenever an if statement is followed by one or more else if statements; the final else if shall be followed by an else statement. In the case of a simple if statement then the else statement need not be included. The requirement for a final else statement is defensive programming. The else statement shall either take appropriate action or contain a suitable comment as to why no action is taken.

```
35 if (x_u8 < 0)  
{  
    x_u8 = 0;  
}  
else if (x_u8 > 3)  
{  
    x_u8 = 3;  
}  
else    <-- This else clause is required, even if the programmer expects this will never be reached  
{  
    /* no change in value of x */  
45 }
```

## 3.1.7 Switch Statements

### Rule CCode\_Switch\_001: Prohibition of Declarations or Definitions between Case and Default Clauses

**Instruction** The case and default clauses in the body of a switch statement shall not be preceded by declarations or definitions.

The syntax for the switch in C is weak, allowing complex, unstructured behaviour.

Example:

```
60 switch(var_u8)  
{  
    static uint8 value_u8;    /* Not OK: definition has to be done outside */  
  
    case 1:  
65 {  
        value_u8 = var_u8;
```

```
    }  
    break;  
    ...
```

## Rule CCode\_Switch\_002: Position of Switch Labels

**Instruction** A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.

The scope of a case or default label shall be the compound statement, which is the body of a switch statement. All case clauses and the default clause shall be at the same scope.

Example:

```
switch(var_u8)  
{  
    case 0:  
    {  
        if (var_s16 > 0)  
        {  
            case 1:          /* Not OK: case is not on the same level */  
            {  
                ...  
            }  
        }  
    }  
    ...
```

## Rule CCode\_Switch\_003: Usage of Break Statement

**Instruction** An unconditional break statement shall terminate every non-empty switch clause even it is not used because of optimized programming.

The last statement in every switch clause shall be a break statement, or if the switch clause is a compound statement, then the last statement in the compound statement shall be a break statement. The break statement can be omitted because of resource optimizations. An upcoming MISRA warning from the check tool has to be commented. Generally it has to be avoided that a break is not written by intention.

Example:

```
switch(var_u8)  
{  
    case 0:          /* OK because of empty case clause */  
    case 1:  
    {  
        ...  
    }  
    break;  
    case 2:  
    {  
        var_s16 = 0;          /* Not OK: break is missing. */  
        /* Is this done by intention? */  
    }  
    default:  
    {  
        ...  
    }  
    break;          /* OK regular break of default path */  
}
```

## Rule CCode\_Switch\_004: Usage of Default Clauses

**Instruction** The final clause of a switch statement shall be the default clause.

The requirement for a final default clause is defensive programming. This clause shall either take appropriate action or contain a suitable comment as to why no action is taken. The default clause shall exist and shall not be the first clause and shall not be in the middle of some other clauses, it shall be exactly the last clause.

## Rule CCode\_Switch\_005: Condition for Switch Expression Values

**Instruction** A switch expression shall not represent a value that is effectively Boolean.

A switch expression shall not evaluate a boolean value, or a switch shall not be used for a boolean variable. For a boolean context still an if-else construct has to be used.

Examples:

```
switch (var1_u8 == var2_u8)    /* Not OK because boolean expression */
...
switch (var_b)                /* Not OK because boolean variable  */
...
```

## Rule CCode\_Switch\_006: Usage of Switch Statements

**Instruction** Every switch statement shall have at least one case clause.

If the code is generated, the switch case might be empty. In this case the generator has to generate a MISRA comment to disable an upcoming MISRA warning from the check tool.

## 3.1.8 Structures and Unions

### Rule CCode\_Struct\_001: Usage of Unions

**Instruction** Unions are allowed but they shall not be used to access to sub-parts of objects or to pack or unpack objects. An object shall not be assigned to an overlapping object.

Unions are allowed only in the way C90 guarantees the implementation. In practice this means, if an union member is written, only a read on the same member is allowed. Otherwise the behaviour of a data packing or unpacking (write on the one union member and read from another union member) is implementation defined and shall not be used.

Following example will show the allowed and forbidden behaviour:

```
typedef union
{
    uint32 Full_u32;          /* For bytes in one uint32 integer */
    uint8  Part_u8[4];        /* For single bytes in an array   */
} MyModule_MyUnion_tun;

void MyModule_Func1(void)
{
    MyModule_MyUnion_tun TstUnion_un;
    uint32                Tst_u32;

    /* OK: because same element was written before it was read */
    TstUnion_un.Full_u32 = 0;
    ...
    Tst_u32 = TstUnion_un.Full_u32;
}

uint8 MyModule_Func2(void)
{
    MyModule_MyUnion_tun TstUnion_un;
    uint8                Tst_u8;
```

```

TstUnion_un.Full_u32 = 0;
...
/* Not OK: It is not guaranteed that the sub elements has */
/* the same content, Part_u8[...] may not be cleared.      */
Tst_u8 = TstUnion_un.Part_u8[2];

return (Tst_u8);

/* The optimizer of the compiler could remove the line */
/* TstUnion_un.Full_u32 = 0; because from that element is */
/* not read within the function. In that case undefined */
/* values might be read and returned from that function. */
}

```

## Rule CCode\_Struct\_002: Typedef Declarations for Structures and Unions

**Instruction** All structure and unions should base on specific typedef declarations.

This rule is a good programming style and will help to define multiple structures or unions based on the same type.

Example structure:

```

/* Typedef in a header file: */
typedef struct
{
    uint8 x;
    uint8 y;
    uint8 z;
} MyModule_Vector_tst;

/* Usage and initialization in a C file: */
MyModule_Vector_tst Vector1_st = { 0, 0, 0 };

```

Example union:

```

/* Typedef in a header file: */
typedef union
{
    uint32 Dword_u32;
    uint8 Byte_a8[4];
} dataCrc32_tun;

/* declaration of a function-local union */
dataCrc32_tun dataRamCrc32_un;

/* Usage and initialization in a C file: */
dataRamCrc32_un.Dword_u32 = Temp_pvu32[idxRam_u16];

```

## Rule CCode\_Struct\_003: Completion of Structure and Union Types

**Instruction** All structure and union types shall be complete at the end of a translation unit.

A complete declaration of the structure or union shall be included within any translation unit that refers to, that reads from or writes to that structure. A pointer to an incomplete type is itself complete and is permitted, therefore the use of opaque pointers is permitted. Every union or structure shall have a defined body. Objects may be declared of incomplete union or structure types but the value of such an object cannot subsequently be used.

Example:

```

struct tnode * pt;          /* Not OK, tnode is incomplete at this point */
...

struct tnode
{
    int count;
}

```

```

05     struct tnode *left;      /* OK, pointer to incomplete type is allowed */
        struct tnode * right; /* OK, pointer to incomplete type is allowed */
};                                     /* OK, type tnode is now complete */

struct MyStruct                      /* Not OK, no members in this struct */
{
10     };

```

## Rule CCode\_Struct\_004: Avoidance of Alignment Gaps inside Structures

**Instruction** Elements of structures shall be arranged in that way to avoid alignment gaps.

Some  $\mu$ Cs can access variables only with their natural alignment. This can affect the efficiency and memory consumption of structures. If no consideration is taken with the order of structure elements, gap bytes could appear implicitly which would waste memory resources. Therefore similar structure elements shall be arranged together to avoid such gap bytes. There is no need to sort elements e.g. stringent from small to big data types. The only request is to avoid implicit gap bytes.

Example (based on a  $\mu$ C which only supports natural alignment):

```

25 typedef struct
{
    uint8 xTestvall_u8;      /* Uses 1 byte in memory */
        <- Here a gap byte can occur
    uint16* adrData_pul6;    /* Pointer uses 4 bytes in memory */
30    uint8 xTestval2_u8;     /* Uses 1 byte in memory */
        <- Here a gap byte can occur
    uint16 stMachine1_u16;   /* Uses 2 bytes in memory */
} Example_tst;

```

### Possible alternative 1:

```

35 typedef struct
{
    uint16* adrData_pul6;
    uint16 stMachine1_u16;
40    uint8 xTestvall_u8;
    uint8 xTestval2_u8;
} Example_tst;

```

### Possible alternative 2:

```

typedef struct
{
    uint16 stMachine1_u16;
    uint8 xTestvall_u8;
    uint8 xTestval2_u8;
    uint16* adrData_pul6;
} Example_t;

```

## 3.1.9 Preprocessing Directives

### Rule CCode\_Prepro\_001: Handling of #include Statements

**Instruction** #include statements in a file should only be preceded by other preprocessor directives or comments.

All the #include statements in a particular code file should be grouped together near the head of the file. The rule states that the only items which may precede a #include in a file are other preprocessor directives or comments.

An exception is the include of memmap header files (" $\langle$ Mip $\rangle$ \_MemMap.h" for BSW modules and "<Module>\_MemMap.h" for ASW components). These headers have to be included more than once and at any position within a file. For memmap headers this rule is not violated. More information about memmap headers can be found in [Chapter "Abstraction of Memory Mapping"](#).

### Rule CCode\_Prepro\_002: Prohibition of Non-Standard Characters in #include Directives

**Instruction** Non-standard characters should not occur in header file names in #include directives.

If the ', \, ", or /\* characters are used between < and > delimiters, or the ', \, or /\* characters are used between the " delimiters in a header name preprocessing token, then the behaviour is undefined. Use of the \ character is permitted

in filenames paths without the need for a derivation if required by the host operation system of the development environment.

### Rule CCode\_Prepro\_003: #include followed by a File Name

**Instruction** The #include directive shall be followed by either a <filename> or "filename" sequence.

The '#include' directive requires the specified filename to be enclosed either within angle brackets <> (usually denoting a system header) or within double quotes "" (usually denoting an user supplied header file). Whether a filename is enclosed within <> or "" affects the locations an implementation may choose to search when attempting to find the specified file.

A preprocessing directive of the form #include <filename.h> searches a sequence of implementation-defined places for a header identified uniquely by the specified sequence between the < and > delimiters, and causes the replacement of that directive by the entire contents of the header.

A preprocessing directive of the form #include "filename.h" causes the replacement of that directive by the entire contents of the source file identified by the specified sequence between the " " delimiters. The named file is searched for in an implementation-defined manner. If the search fails the directive is reprocessed as if it read #include <filename.h> with the identical contained sequence (including > characters, if any) from the original directive.

Within the CDG development environment both kinds of include directive using a < or " delimiter are working (the implementation-defined places and search strategies are well defined within the CDG development environment.). Both delimiters can be used but it is recommended to use the " delimiters because as written above a double-staged search is executed to find the header file.

For example, the following includes are allowed:

```
#include "filename.h"
#include <filename.h>
#define HEADER_A "filename.h"
#include HEADER_A
```

Logically it is not allowed to make an include in the following form:

```
#include filename.h
#include 'filename.h'
```

**Hint** If an include of a header is encapsulated by a #define as shown in the example above no conflict with rule [\[BSW\\_HeaderInc\\_009\]](#) shall occur. The name of the #define shall not be defined in the form of "HEADERNAME\_H" which is the used convention for the header include protection specified in rule [\[BSW\\_HeaderInc\\_009\]](#). Therefore in the example the name HEADER\_A is chosen to differ from the name HEADERNAME\_H which is used by the include protection.

### Rule CCode\_Prepro\_004: Prohibition of #undef

**Instruction** #undef shall not be used.

#undef should normally not be needed. Its use can lead to confusion with respect to the existence or meaning of a macro when it is used in the code.

**Hint** An exception of this rule is the MemMap concept (for more details see [Chapter 3.2.4 "Abstraction of Memory Mapping"](#)). Here #undef is used from the concept.

### Rule CCode\_Prepro\_005: Handling of Preprocessor Operator "defined"

**Instruction** The "defined" preprocessor operator shall only be used in one of the two standard forms.

The only two permissible forms for the "defined" preprocessor operator are:

defined (identifier)

defined identifier

Any other form leads to undefined behaviour. Generation of the token defined during expansion of a #if or #elif preprocessing directive also leads to undefined behaviour and shall be avoided.

Examples:

```
#if defined (X)           /* OK, normal usage           */
15 #if defined X           /* OK, normal usage           */
    #if defined (X > Y)   /* Not OK, results in undefined behaviour */

#define DEFINED defined
    #if DEFINED(X)        /* Not OK, results in undefined behaviour */
```

**Hint** This rule handles only the "defined" preprocessor operator. If "defined" is used the rule has to be followed. Otherwise the preprocessor operators "#ifdef" and "#ifndef" are also valid as replacement for "#if defined" and "#if !defined".

## Rule CCode\_Prepro\_006: Correctness of Preprocessing Directives

**Instruction** Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.

When a section of source code is excluded by preprocessor directives, the content of each excluded statement is ignored until a #else, #elif or #endif directive is encountered (depending on the context). If one of these excluded directives is badly formed, it may be ignored without warning by a compiler with unfortunate consequences.

The requirement of this rule is that all preprocessor directives shall be syntactically valid even when they occur within an excluded block of code. In particular, ensure that #else and #endif directives are not followed by any characters other than white-space. Compilers are not always consistent in enforcing this ISO requirement.

```
40 #define AAA 2
    ...
    uint16 foo(void)
    {
        uint16 x = 0;
        ...
45 #ifndef AAA
        x = 1;
    #else                               /* Not OK, no valid syntax, line will be ignored */
        x = AAA;
    #endif
50     ...
        return x;
    }
```

## Rule CCode\_Prepro\_007: Hold Preprocessor Directives together

**Instruction** All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.

When the inclusion and exclusion of blocks of statements is controlled by a series of preprocessor directives, confusion can arise if all of the relevant directives do not occur within one file. This rule requires that all preprocessor directives in a sequence of the form #if / #ifdef ... #elif ... #else ... #endif shall reside in the same file. Observance of this rule preserves good code structure and avoids maintenance problems.

Notice that this does not preclude the possibility that such directives may exist within included files as long as all directives that relate to the same sequence are located in one file.



Example:

```
file.c
#define A
...

#ifdef A
    #include "file1.h"
#endif
...
#if 1
    #include "file2.h"
...
EOF

file1.h
#if 1
...
#endif          /* OK, self-contained preprocessor directive */
EOF

file2.h
...
#endif          /* Not OK, belongs to #ifdef A in file.c      */
```

## Rule CCode\_Prepro\_008: Macro Identifiers in Preprocessor Directives

**Instruction** All macro identifiers in preprocessor directives shall be defined before use, except in `#ifdef` and `#ifndef` preprocessor directives and the `defined()` operator.

If an attempt is made to use an identifier in a preprocessor directive, and that identifier has not been defined, the preprocessor will sometimes not give any warning but will assume the value zero. `#ifdef`, `#ifndef` and `defined()` are provided to test the existence of a macro, and are therefore excluded. Consideration should be given to the use of a `#ifdef` test before an identifier is used. Note that preprocessing identifiers may be defined either by use of `#define` directives or by options specified at compiler invocation. However the use of the `#define` directive is preferred.

Example:

*Following macros are defined to be used within a preprocessor directive:*

```
#define x 0
#define CAN_ZERO (uint8)0x0U
```

*The preprocessor directive:*

```
#if (x == CAN_ZERO)
...

```

*The preprocessor generates following result:*

```
#if 0 == (0)0x0U
```

*"uint8" is replaced by 0 because it was not explicitly defined as macro!*

*Hint: U will be recognized.*

## 3.1.10 Macros

### Rule CCode\_Macro\_001: Handling of C Macros

**Instruction** C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.

These are the only permitted uses of macros. Storage class specifiers and type qualifiers include keywords such as *extern*, *static* and *const*. Any other use of `#define` could lead to unexpected behaviour when substitution is made, or to very hard-to-read code.

In particular macros shall not be used to define statements or parts of statements except the use of the do-while construct. Nor shall macros redefine the syntax of the language. All brackets of whatever type ( ) { } [ ] in the macro replacement list shall be balanced.

The do-while-zero construct (see example below) is the only permitted mechanism for having complete statements in a macro body. The do-while-zero construct is used to wrap a series of one or more statements and ensure correct behaviour. Note: the semicolon shall be omitted from the end of the macro body.

Examples:

```
#define PI          3.14159F          /* OK, constant */
#define CAT        (PI)               /* OK, parenthesised expression */
#define XSTAL      10000000           /* OK, constant */
#define CLOCK      (XSTAL/16)         /* OK, Constant expression */
#define PLUS2(X)    ((X) + 2)         /* OK, macro expanding to expression */
#define STOR        extern            /* OK, storage class specifier */
#define INIT(value) {(value), 0, 0}   /* OK, braced initializer */
#define FILE_A      "filename.h"      /* OK, string literal */
#define READ_TIME_32() do              \
    {                                  \
        DISABLE_INTERRUPTS();          \
        time_now = (uint32)TIMER_HI << 16; \
        time_now = time_now | (uint32)TIMER_LO; \
        ENABLE_INTERRUPTS();          \
    } while (0) /* OK, example of do-while-zero */

#define int32_t    long /* Not OK, use typedef instead */
#define CAT        PI  /* Not OK, non parenthesised expression */
#define MY_IF      if( /* Not OK, unbalanced () and language redefinition */
```

## Rule CCode\_Macro\_002: Validity of C Macros

**Instruction** Macros shall not be `#define`'d or `#undef`'d within a block.

While it is legal C to place `#define` or `#undef` directives anywhere in a code file, placing them inside blocks is misleading as it implies a scope restricted to that block, which is not the case.

Normally, `#define` directives will be placed near the start of a file, before the first function definition. Normally, `#undef` directives will not be needed (see [\[CCode\\_Prepro\\_004\]](#)).

Example:

```
void MyModule_Func(void)
{
    #define MYMODULE_MACRO 10 /* Not OK, Macro is defined ... */
    ...
    #undef MYMODULE_MACRO    /* Not OK, and undefined within a block */
}
```

Example:

```
#define MYMODULE_POSTBUILD 5
...

void MyModule_Init(void)
{
    if(MYMODULE_POSTBUILD == MyModule_Cfg_ConfigVariant_u8)
    {
        #undef MYMODULE_POSTBUILD /* Not OK, undef of global #define */
        #define MYMODULE_POSTBUILD 10
    }
}
```

```

05     }
    }

```

### Rule CCode\_Macro\_003: Handling of Function-like Macros

**Instruction** In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##.

Within a definition of a function-like macro, the arguments shall be enclosed in parentheses. Following example will show what could happen if this rule is not fulfilled.

```

Simple macro to get an absolute value:
#define abs(x) ((x) >= 0) ? (x) : -(x)    /* OK, parentheses are used */
#define abs(x) (( x >= 0) ?  x: -x )    /* Not OK, parentheses are not used */

```

If this rule is not adhered to then when the preprocessor substitutes the macro into the code the operator precedence may not give the desired results.

Consider what happens if the second, incorrect, definition is substituted into the expression:  
`z = abs(a - b);`

```

Results after preprocessor:
z = ((a - b >= 0) ? a - b: -a - b);

```

The sub-expression "`-a - b`" is equivalent to "`-(a-b)`" rather than "`-(a-b)`" as intended. Putting all the parameters in parentheses in the macro definition avoids this problem.

### Rule CCode\_Macro\_004: Calling of Function-like Macros

**Instruction** A function-like macro shall not be invoked without all of its arguments.

This is a constraint error, but preprocessors have been known to ignore this problem. Each argument in a function-like macro shall consist of at least one preprocessing token otherwise the behaviour is undefined.

Example:

```

#define MAX(A, B)    (((A) > (B)) ? (A) : (B))
...

sint16 MyModule_Func(sint16 arg1_s16, arg2_s16)
{
    sint16 value_s16;

    value_s16 = MAX(arg1_s16, 10);    /* OK, both arguments are given */
    value_s16 += MAX(arg2_s16);      /* Not OK, only one argument is given */

    return value_s16;
}

```

### Rule CCode\_Macro\_005 Arguments of Function-like Macros

**Instruction** Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.

If any of the arguments act like preprocessor directives, the behaviour when macro substitution is made can be unpredictable.

Example:

```

#define MAX(A, B)    (((A) > (B)) ? (A) : (B))
...

sint16 MyModule_Func(sint16 arg1_s16, arg2_s16)
{

```

```

05  sint16 value_s16;

    value_s16 = MAX(
        #ifdef MAGIC_SWITCH
            arg1_s16,
10     #else
            arg2_s16,
        #endif
        10);          /* Not OK, because of preprocessor    */
                    /* within argument list of macro.      */

15  return value_s16;
}

```

## 3.2 Rule Set: Compiler Abstraction / Portability

The motivation for compiler abstraction is portability. Source code must be compiler-independent because software modules will be used on several  $\mu$ C platforms (UBK and foreign ECUs) and compiled via several compilers. Nowadays mostly 32 bit controllers are in use (with different compilers) but theoretically it is possible to integrate software modules to 16 bit and 8 bit controller platforms, too. Upward portability must be possible (8bit  $\rightarrow$  16bit  $\rightarrow$  32bit) whereas downward portability seems not so important (32bit  $\rightarrow$  16bit  $\rightarrow$  8bit) because it is the more difficult form.

MCAL functions are always  $\mu$ C specific. Therefore compiler independence is not so important for MCAL functions. But if possible, MCAL modules can handle and use parts of the compiler abstraction topics, too. All other SW layers above MCAL should be implemented portable.

In particular cases rules of rule set "Compiler Abstraction / Portability" may not be fulfilled if a module is explicitly written for an explicit  $\mu$ C or compiler. This has to be documented and such software modules shall not be provided for other  $\mu$ Cs or compilers than the preferred one.

Finally, it shall be ensured that the BSW modules written in C can be used and compiled within project that are using C++ compilers. Attention has to be paid for macros and inline functions to enable the code to be translated from any C++ compiler. The rules given in chapter 3.2.10 shall be kept from every BSW module also MCALs.

### 3.2.1 Main Rule

#### Rule Abstr\_Main\_001: Main Rule of Compiler Abstraction

**Instruction** The source code of software modules (at least above the  $\mu$ C Abstraction Layer (MCAL)) shall be neither processor-dependent nor compiler-dependent. No reliance shall be placed on undefined or unspecified behaviour of the compiler.

Those software modules have to be developed once and shall be compilable for all processor platforms without any changes. Any necessary processor or compiler specific instructions (e.g. memory locators, pragmas, use of atomic bit manipulations etc.) have to be exported to macros and include files. This proceeding will minimize number of variants and development effort. Undefined or unspecified behaviour of the compiler shall not be used. This principle is covered by many rules of this rule set.

### 3.2.2 Prohibition of Compiler Specifics

#### Rule Abstr\_Main\_002: Prohibition of Compiler Specific Headers and Libraries

**Instruction** Compiler specific header files, libraries and intrinsic functions shall not be used.

Compiler specific header files and libraries (provided by compiler manufacturer or 3rd party) are not portable because they are several times within a single compiler with distinct content eventually also using compiler internals. This is quite

intransparent and thus hard to understand. Concluding the usage of compiler-specific header files and functions is not allowed.

Within the MISRA standard the usage of special library functions as well as complete libraries is forbidden (e.g. error indicator "errno", macros "offsetof", "setjmp" "longjmp", "atof", "atoi", "atol", library functions "abort", "exit", "getenv", "system" and the libraries "stdio.h", "signal.h" and "time.h"). This rule is more comprehensive as the general usage of compiler libraries is not allowed.

Additionally all intrinsic functions are extensions to the C-language and therefore they are not portable. Intrinsic functions are µC specific operations which were encapsulated in macros (like accesses to special registers of the µC (Example for Tricore: "\_mfc" and "\_mtcr", encapsulation of special commands from the assembler (Example for Tricore: "\_isync", "\_dsync" or "\_nop").

ANSI-C libraries are also affected by this rule. That means that library functions specified by ANSI-C shall not be used. Currently there are discussions to allow some headers of the ANSI-C library but the discussions are business unit-specific and thus there is no general conclusion. Therefore the guideline go a conservative way in disallowing the ANSI-C library because BSW software will be delivered to different business units and it is not ensured that the needed headers of the ANSI-C library are really available there.

### Rule Abstr\_Main\_003: Prohibition of Compiler Specific Keywords

**Instruction** Compiler specific keywords and internal defines shall not be used.

Compiler specific keywords and defines are compiler specific extensions which are generally not portable and shall not be used inside the C-code.

Examples for compiler specific keywords are (exemplary for Tricore) single bit type "\_bit", attribute keywords "\_\_attribute\_\_", typeof operator or specific techniques with brackets ({...}). Inline keywords are not allowed too, but for them a solution is provided described in [\[Abstr\\_Inline\\_001\]](#).

Compilers provide sometimes defines to handle internal compiler issues (like version specific features), target specific issues (like distinction of different target / derivation types), CPU errata workaround, limits and configuration issues (like settings for FPU Floating Point Unit). Such defines are compiler specific and are contrary to portability topic.

Examples (not to be used):

Writing code for different compilers within the same source file:

```
#if defined tricore
    /* any TriCore specific code here */
#elif defined WIN32
    /* any Windows specific code here */
#else
    #error This code only works for TriCore and Win32
#endif
```

Hardware Configuration specific code:

```
#ifndef __HARD_FLOAT__
    #error This code requires a FPU (Floating Point Unit)
#else
    /* any useful code here */
#endif
```

### Rule Abstr\_Main\_004: Prohibition of Redefinition of Reserved Identifiers, Macros and Functions

**Instruction** Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined. The names of standard library macros, objects and functions shall not be reused.

It is generally bad practice to #undef a macro which is defined in the standard library. It is also bad practice to #define a macro name which is a C reserved identifier, a C keyword or the name of any macro, object or function in the standard library. For example, there are some specific reserved words and function names which are known to give rise to

undefined behaviour if they are redefined or undefined, including defined, `__LINE__`, `__FILE__`, `__DATE__`, `__TIME__`, `__STDC__`, `errno` and `assert`. Generally, all identifiers that begin with the underscore character are reserved.

Where new versions of standard library macros, objects or functions are used by the programmer (e.g. enhanced functionality or checks of input values) the modified macro, object or function shall have a new name. This is to avoid any confusion as to whether a standard macro, object or function is being used or whether a modified version of that function is being used.

### 3.2.3 Abstraction of Addressing Keywords

For 16 bit controllers the address bus is 16 bit. With 16 bit only 64 kB can be addressed. But usually there is more code/data to be addressed. Therefore it has to be distinguished whether an address within these 64 kB (**near**) or outside (**far**) shall be reached. The near/far keywords are compiler specific and therefore have to be encapsulated. This encapsulation is described in [\[Document AUTOSAR\\_SWS\\_CompilerAbstraction / URL: \\Si8256\autosar\\$\SVN3-COPY\22 - Releases\40\\_Release4.0R0003\01\\_Standard\AUTOSAR\\_SWS\\_CompilerAbstraction.pdf\]](#).

Some words before:

16 bit microcontroller abstraction is mainly important for generated code. For hand coding there are some disadvantages:

- ▶ There is more effort when 16 bit microcontrollers issues also have to be considered.
- ▶ Code is more difficult to understand.
- ▶ Code for 16 bit microcontroller is usually never tested. So hand coded 16 bit microcontroller abstraction might be released buggy (whereas generated code usually is with much less bugs).

But to be AUTOSAR compliant also for hand code the abstraction of addressing keywords will be done as described with the following rules.

#### Rule Abstr\_NearFar\_001: Prohibition of Usage of Addressing Keywords

**Instruction** Direct use of compiler and platform specific keywords for addressing of data and code like `"_near"` or `"_far"` is not allowed.

Direct use of not standardized keywords in the source code will create compiler and platform dependencies that shall strictly be avoided. If no precautions were made, portability and reusability of influenced code is deteriorated and effective release management is costly and hard to maintain.

Specific keywords shall be encapsulated via special headers.

#### Rule Abstr\_NearFar\_002: AUTOSAR 16 bit Microcontroller Abstraction

**Instruction** To encapsulate 16 bit microcontroller the AUTOSAR standardized #defines shall be used.

To encapsulate 16 bit microcontroller the AUTOSAR standardized #defines have to be used. For these #defines

- ▶ memclass is described in [Table 3 "Overview of Memory Classes"](#)
- ▶ ptrclass is defined in [Table 4 "Overview of Pointer Classes"](#)

Here the list of available #defines:

- ▶ #define **FUNC**(rettype, memclass)  
rettype is the return type of the function  
memclass is the memory class of the function

The compiler abstraction shall define the FUNC macro for the declaration and definition of **functions**, that ensures correct syntax of function declarations as required by a specific compiler.

In the parameter list of this macro no further Compiler Abstraction macros shall be nested. Instead, use a previously defined type as return type or use FUNC\_P2CONST/FUNC\_P2VAR.

► #define **FUNC\_P2CONST**(rettype, ptrclass, memclass)

rettype is the return type of the function

ptrclass is the classification of the constant where the pointer points to

memclass is the memory class of the function.

The compiler abstraction shall define the FUNC\_P2CONST macro for the declaration and definition of **functions returning a pointer to a constant**. This shall ensure the correct syntax of function declarations as required by a specific compiler.

In the parameter list of the FUNC\_P2CONST, no further Compiler Abstraction macros shall be nested.

► #define **FUNC\_P2VAR**(rettype, ptrclass, memclass)

rettype is the return type of the function

ptrclass is the classification of the variable where the pointer points to

memclass is the memory class of the function

The compiler abstraction shall define the FUNC\_P2VAR macro for the declaration and definition of **functions returning a pointer to a variable**. This shall ensure the correct syntax of function declarations as required by a specific compiler.

In the parameter list of the macro FUNC\_P2VAR, no further Compiler Abstraction macros shall be nested.

► #define **P2VAR**(ptrtype, memclass, ptrclass)

ptrtype is the type of the referenced variable

memclass is the classification of the pointer itself

ptrclass is the classification of the variable where the pointer points to

The compiler abstraction shall define the P2VAR macro for the declaration and definition of **pointers in RAM, pointing to variables**.

The pointer itself is modifiable (e.g. ExamplePtr++).

The pointer's target is modifiable (e.g. \*ExamplePtr = 5).

► #define **P2CONST**(ptrtype, memclass, ptrclass)

ptrtype is the type of the referenced const

memclass is the classification of the pointer itself

ptrclass is the classification of the const where the pointer points to

The compiler abstraction shall define the P2CONST macro for the declaration and definition of **pointers in RAM pointing to constants**.

The pointer itself is modifiable (e.g. ExamplePtr++).

The pointer's target is not modifiable (read only).

► #define **CONSTP2VAR**(ptrtype, memclass, ptrclass)

ptrtype is the type of the referenced variable

memclass is the classification of the pointer itself

ptrclass is the classification of the variable where the pointer points to

The compiler abstraction shall define the CONSTP2VAR macro for the declaration and definition of **constant pointers accessing variables**.

The pointer itself is not modifiable (fix address).

The pointer's target is modifiable (e.g. \*ExamplePtr = 18).

► #define **CONSTP2CONST**(ptrtype, memclass, ptrclass)

ptrtype is the type of the referenced const

memclass is the classification of the pointer itself

ptrclass is the classification of the const where the pointer points to

The compiler abstraction shall define the CONSTP2CONST macro for the declaration and definition of **constant pointers accessing constants**.

The pointer itself is not modifiable (fix address).

The pointer's target is not modifiable (read only).

► #define **P2FUNC**(rettype, ptrclass, fctname)

rettype is the return type of the function

ptrclass is the classification of the pointer's distance

fctname function name respectively name of the defined type

The compiler abstraction shall define the P2FUNC macro for the type definition of **pointers to functions**.

► #define **CONST**(consttype, memclass)

consttype is the type of the constant

memclass is the classification of the constant itself

The compiler abstraction shall define the CONST macro for the declaration and definition of **constants**.

► #define **VAR**(vartype, memclass)

vartype is the type of the variable

memclass classification of the variable itself

The compiler abstraction shall define the VAR macro for the declaration and definition of **variables**.

Available memclasses are shown in [Table 3 "Overview of Memory Classes"](#).

Available ptrclasses are shown in [Table 4 "Overview of Pointer Classes"](#).

In this tables **<MODULE>** is the module abbreviation:

► For BSW from the module list in UPPERCASE letters (e.g. 'EEP' or 'CAN').

► For ASW the short name of the software component in case sensitive letters.

Table 3 Overview of Memory Classes

Memory Type	Syntax of memory class (memclass) macro parameter	Comments
Variable	<Module>_VAR_<INIT_POLICY>_<SIZE> <Module>_VAR_FAST_<INIT_POLICY>_<SIZE> <Module>_VAR_SLOW_<INIT_POLICY>_<SIZE> <Module>_INTERNAL_VAR_<INIT_POLICY>_<SIZE> <Module>_VAR_SAVED_ZONE<X>_<SIZE>	memclass follows the same rules than for the memory allocation keyword for variables – see table <a href="#">Table 7 "Overview of Memory Allocation Keywords for Variables"</a>



Memory Type	Syntax of memory class (memclass) macro parameter	Comments
Constant	<Module>_CONST_<SIZE> <Module>_CONST_FAST_<SIZE> <Module>_CONST_SLOW_<SIZE> <Module>_CALIB_<SIZE> and so on	memclass follows the same rules than for the memory allocation keyword for constants – see table <a href="#">Table 8 "Overview of Memory Allocation Keywords for Constants"</a>
Code	<Module>_CODE <Module>_CODE_FAST <Module>_CODE_SLOW <Module>_CODE_LIB and so on	memclass follows the same rules than for the memory allocation keyword for code – see table <a href="#">Table 9 "Overview of Memory Allocation Keywords for Code"</a>

Table 4 Overview of Pointer Classes

Memory Type	Syntax of pointer class (ptrclass) macro parameter	Comments
Pointer to variable	<Module>_APPL_DATA	Configurable memory class for pointers to application data (expected to be in RAM or ROM) passed via API.
Pointer to constant	<Module>_APPL_CONST	Configurable memory class for pointers to application constants (expected to be certainly in ROM, for instance pointer of Init-function) passed via API.
Pointer to code	<Module>_APPL_CODE	Configurable memory class for pointers to application functions (e.g. call back function pointers).
Pointer to code	<Module>_CALLOUT_CODE	Configurable memory class for pointers to application functions (e.g. callout function pointers).

Some examples:

```

/*
  Definition of a variable test with type uint16 and being cleared
  test is defined in a FC called MyFC
*/
VAR(uint16, MyFC_VAR_CLEARED_16) test;

/* Definition of a pointer to that variable */
P2VAR(uint16, MyFC_VAR_CLEARED_16, MyFC_APPL_DATA) test;

/*
  Declaration of test in an other FC called OtherFC.
  Hint: For the memclass the FC name of the definition has to be used.
*/
extern VAR(uint16, MyFC_VAR_CLEARED_16) test;

```

### Rule Abstr\_NearFar\_003: Usage of Module Name of near/far Addressing Keywords

**Instruction** For declarations the module name of the defining module has to be used for the near/far addressing keywords.

To have a single source of information and to ensure that there is no conflicting information for the abstraction keywords always the module name of the defining module has to be used.

Example: See previous example for it.

## 3.2.4 Abstraction of Memory Mapping

To be able to map code/data to special memory areas usually `#pragma` commands are used. Since these `#pragma` commands are compiler specific they are encapsulated in memory allocation keywords.

This encapsulation is described in [\[Document AUTOSAR\\_SWS\\_MemoryMapping / URL: \\Si8256\autosar\\$\SVN3-COPY\22\\_Releases\40\\_Release4.0R0003\01\\_Standard\AUTOSAR\\_SWS\\_MemoryMapping.pdf\]](#).

Additional an AUTOSAR bug fix proposal is considered [\[Document AUTOSAR Bugzilla / URL: http://www.autosar.org/bugzilla/show\\_bug.cgi?id=48406\]](#).

For many ECUs and microcontroller platforms it is of utmost necessity to be able to map code, variables and constants module wise to specific memory sections. As opposed to [Chapter 3.2.3 "Abstraction of Addressing Keywords"](#) where the encapsulation of near/far addressing keywords was described this chapter specifies a mechanism for the mapping of code, data and constants to specific memory sections. To explain the context here are some use cases:

- ▶ **Avoidance of waste of RAM:** If different variables (8, 16 and 32 bit) are used within different modules on a 32 bit platform, the linker might leave gaps in RAM when allocating the variables in the RAM. This is because the microcontroller platform requires a specific alignment of variables and some linkers do not allow an optimization of variable allocation. This wastage of memory can be avoided if the variables are mapped to specific memory sections depending on their size. This minimizes unused space in RAM.
- ▶ **Usage of specific RAM properties:** Some variables (e.g. the RAM mirrors of the NVRAM Manager) must not be initialized after a power-on reset. It shall be possible to map them to a RAM section that is not initialized after a reset.
- ▶ **Usage of specific ROM properties:** In large ECUs with external flash memory there is the requirement to map modules with functions that are called very often to the internal flash memory that allows for fast access and thus higher performance. Modules with functions that are called rarely or that have lower performance requirements are mapped to external flash memory that has slower access.
- ▶ **Support of Memory Protection:** The usage of hardware memory protection requires a separation of the modules variables into different memory areas. Internal variables are mapped into protected memory, buffers for data exchange are mapped into unprotected memory.

### Rule Abstr\_MemMap\_001: Usage of Memory Mapping Concept instead of `#pragma` Keywords

**Instruction** Direct use of compiler and platform specific "`#pragma`" keywords is not allowed. Memory mapping of code, variables and constants shall be done by using the memory mapping concept.

Compilers are providing in general "`#pragma`" keywords to map and control the assignment of variables, constants and functions to specific sections and memory areas. These `#pragma` keywords are very compiler and linker specific, therefore a mechanism has to be found to encapsulate and abstract these keywords. Memory mapping header files will be used containing macros which encapsulate the specific `#pragma` keywords. The use of these macros and the include of the memory mapping header files inside the c and h files provides a memory allocation mechanism which is independent from compiler and microcontroller specific keywords and properties.

**Hint** The assignment of the sections to dedicated memory areas / address ranges is not the scope of the memory mapping concept because it is typically and additionally done via linker control files.

**Hint** The following concept is based on AUTOSAR 4.0 **Rev. 3**. The main concept is already there with AUTOSAR 4.0 Rev. 0. But especially the macro names and the initialization policy are based on current version AUTOSAR 4.0 Rev. 3.

**Hint** Some words about **older AUTOSAR releases**:

In older AUTOSAR releases the macro names are different:

1. Missing initialization policy
2. Suffix "BIT"

Quote from AUTOSAR 3.1: Each AUTOSAR software module shall support the configuration of at least the following different memory types. **It is allowed to add module specific sections** as they are mapped and thus are configurable within the module's configuration file.

So for older AUTOSAR releases this Pragma Concept should be valid in terms of "adding module specific sections".

AUTOSAR 3.1 is only valid for BSW. So any Pragma Concept for ASW is an extension for AUTOSAR 3.1.

## Rule Abstr\_MemMap\_002: Structure of Memory Mapping Concept

**Instruction** Declarations and definitions of code, variables and constants shall be wrapped with a start symbol (syntax: `<MODULE>_START_SEC_<NAME>`) and stop symbol (syntax: `<MODULE>_STOP_SEC_<NAME>`) and includes of memory mapping headers ("`<Mip>_MemMap.h`" for BSW modules and "`<SWC>_MemMap.h`" for ASW components).

The code sequence for memory mapping consists of the following five steps:

1. Definition of **start symbol** for module memory section with syntax `<MODULE>_START_SEC_<NAME>`.
2. Inclusion of the **memory mapping header** file.
3. Declaration/definition of code, variables or constants belonging to the specified section.
4. Definition of **stop symbol** for module memory section with syntax `<MODULE>_STOP_SEC_<NAME>`.
5. Inclusion of the **memory mapping header** file.

The mapping concept is the same for declaration and definition and has to be done for both in the same way.

The start and stop symbol has following syntax: `<MODULE>_START_SEC_<NAME>` and `<MODULE>_STOP_SEC_<NAME>` where `<MODULE>` is the module abbreviation:

- ▶ For BSW from the corresponding AUTOSAR module list, for example [\[Document AUTOSAR V4.0 Rev. 3 module list / URL: \\Si8256\autosar\\$\SVN3-COPY\22\\_Releases\40\\_Release4.0R0003\02\\_Auxiliary\AUTOSAR\\_TR\\_BSWModuleList.pdf\]](#) in UPPERCASE letters (e.g. 'EEP' or 'CAN').
  - ▶ For ASW the short name of the **SoftW**are **C**omponent (**SWC**) in case sensitive letters.
- `<NAME>` is the keyword of the specific section (all keywords can be found in tables 7, 8 and 9).

Memory mapping headers are different for BSW and ASW:

- ▶ For BSW "`{Mip}_MemMap.h`" has to be included.

The Module implementation prefix `{Mip}` shall be formed in the following way:

`<Ma>[_<vi>_<ai>]`

Where `<Ma>` is the Module abbreviation of the BSW Module (SWS\_BSW\_101, see also [\[Document AUTOSAR V4.0 Rev. 3 module list / URL: \\Si8256\autosar\\$\SVN3-COPY\22\\_Releases\40\\_Release4.0R0003\02\\_Auxiliary\AUTOSAR\\_TR\\_BSWModuleList.pdf\]](#)), `<vi>` is its vendorId and `<ai>` is its vendorApiInfix. The sub part in square brackets `[_<vi>_<ai>]` is omitted if no vendorApiInfix is defined for the BSW Module (BSW00300, BSW00347).

- ▶ For ASW "`<SWC>_MemMap.h`" has to be included.

## Examples:

```

/* Memory location of a variable for BSW: */
#define EEP_START_SEC_VAR_CLEARED_16
#include "Eep_MemMap.h"
static VAR(uint16, EEP_VAR_CLEARED_16) EepTimer;
static VAR(uint16, EEP_VAR_CLEARED_16) EepRemainingBytes;
#define EEP_STOP_SEC_VAR_CLEARED_16
#include "Eep_MemMap.h"

/* This example is only valid for code generators: */
/* Referencing CAN variables within EEP for BSW (if it is not done with RTE): */
#define CAN_START_SEC_VAR_CLEARED_32
#include "Can_MemMap.h"
extern VAR(uint32, CAN_VAR_CLEARED_32) CanMessage1;
extern VAR(uint32, CAN_VAR_CLEARED_32) CanMessage2;
#define CAN_STOP_SEC_VAR_CLEARED_32
#include "Can_MemMap.h"
/* For referencing: The module name of the defining module has to be used. */
/* For hand coding the header file concept shall be followed */

/* Memory location of a function for BSW: */
#define EEP_START_SEC_CODE_SLOW
#include "Eep_MemMap.h"
FUNC(void, EEP_CODE_SLOW) Eep_ExampleFunction(void);
#define EEP_STOP_SEC_CODE_SLOW
#include "Eep_MemMap.h"

```

The start and stop memory allocation keywords are configured with section identifiers in "{Mip}\_MemMap.h" or "<SWC>\_MemMap.h".

Table 5 Overview of Meaning of &lt;INIT\_POLICY&gt; Postfix

Keyword	Description
<b>NO_INIT</b>	AUTOSAR usage: Used for variables that are never cleared and never initialized. RB usage: Not to be used.
<b>CLEARED</b>	AUTOSAR usage: Used for variables that are cleared to zero after every reset. RB usage: Used for variables that are cleared to zero after every reset.
<b>POWER_ON_CLEARED</b>	AUTOSAR usage: Used for variables that are cleared to zero only after power on reset. RB usage: Used for variables that are cleared to zero only after power on reset.
<b>INIT</b>	AUTOSAR usage: Used for variables that are initialized with values after every reset. RB usage: Used for variables that are initialized with values after every reset.
<b>POWER_ON_INIT</b>	AUTOSAR usage: Used for variables that are initialized with values only after power on reset. RB usage: Not to be used.

Table 6 Overview of Meaning of &lt;SIZE&gt; Postfix

Keyword	Description
<b>BOOLEAN</b>	Used for variables and constants of size 1 bit.
<b>8</b>	Used for variables and constants which have to be aligned to 8 bit or used for composite data types: arrays, structs and unions containing elements of maximum 8 bits.
<b>16</b>	Used for variables and constants which have to be aligned to 16 bit or used for composite data types: arrays, structs and unions containing elements of maximum 16 bits.
<b>32</b>	Used for variables and constants which have to be aligned to 32 bit or used for composite data types: arrays, structs and unions containing elements of maximum 32 bits.
<b>UNSPECIFIED</b>	Used for variables, constants, structure, array and unions when alignment does not fit the criteria of 8, 16 or 32 bit.

Hint: Data type "\_bit" was only supported for HighTec compilers V3.x for Infineon TriCore. It won't be supported for any other compiler. Therefore data type "\_bit" is not to be used.

Table 7 Overview of Memory Allocation Keywords for Variables

Type of variables	Syntax of memory allocation keywords	Comments
Normal variables	<code>&lt;MODULE&gt;_START_SEC_VAR_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_VAR_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code>	To be used for global or static variables. Usage within CDG: <b>For variables used in tasks from 5 ms to 20 ms.</b>
Variables with frequently usage	<code>&lt;MODULE&gt;_START_SEC_VAR_FAST_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_VAR_FAST_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code>	To be used for all global or static variables that have at least one of the following properties: <ul style="list-style-type: none"> <li>▶ frequently used</li> <li>▶ high number of accesses in source code</li> <li>▶ accessed bitwise</li> </ul> Usage within CDG: <b>For variables used in tasks faster than 5 ms and frequently used interrupts.</b>
Variables with rarely usage	<code>&lt;MODULE&gt;_START_SEC_VAR_SLOW_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_VAR_SLOW_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR. Usage within CDG: <b>For variables used in tasks slower than 20 ms</b> and other very slow called functions.
Variables for calibration	<code>&lt;MODULE&gt;_START_SEC_INTERNAL_VAR_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_INTERNAL_VAR_&lt;INIT_POLICY&gt;_&lt;SIZE&gt;</code>	To be used for global or static variables accessible from a calibration tool.
RAM buffers in NVRAM	<code>&lt;MODULE&gt;_START_SEC_VAR_SAVED_ZONE&lt;X&gt;_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_VAR_SAVED_ZONE&lt;X&gt;_&lt;SIZE&gt;</code>	To be used for RAM buffers of variables saved in non volatile memory. Description for <X> is missing in AUTOSAR.

Table 8 Overview of Memory Allocation Keywords for Constants

Type of constants	Syntax of memory allocation keywords	Comments
Normal constants	<code>&lt;MODULE&gt;_START_SEC_CONST_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONST_&lt;SIZE&gt;</code>	To be used for global or static constants.
Constants with frequently usage	<code>&lt;MODULE&gt;_START_SEC_CONST_FAST_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONST_FAST_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants with rarely usage	<code>&lt;MODULE&gt;_START_SEC_CONST_SLOW_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONST_SLOW_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants for calibration	<code>&lt;MODULE&gt;_START_SEC_CALIB_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CALIB_&lt;SIZE&gt;</code>	To be used for calibration constants.
Constants for calibration with frequently usage	<code>&lt;MODULE&gt;_START_SEC_CALIB_FAST_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CALIB_FAST_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants for calibration with rarely usage	<code>&lt;MODULE&gt;_START_SEC_CALIB_SLOW_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CALIB_SLOW_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants for cartography	<code>&lt;MODULE&gt;_START_SEC_CARTO_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CARTO_&lt;SIZE&gt;</code>	To be used for cartography constants.

Type of constants	Syntax of memory allocation keywords	Comments
Constants from link-time configuration	<code>&lt;MODULE&gt;_START_SEC_CONFIG_DATA_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONFIG_DATA_&lt;SIZE&gt;</code>	Constants with attributes that show that they reside in one segment for <b>link-time</b> module configuration.
Constants from link-time configuration with frequently usage	<code>&lt;MODULE&gt;_START_SEC_CONFIG_DATA_FAST_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONFIG_DATA_FAST_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants from link-time configuration with rarely usage	<code>&lt;MODULE&gt;_START_SEC_CONFIG_DATA_SLOW_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONFIG_DATA_SLOW_&lt;SIZE&gt;</code>	This is an extension to AUTOSAR.
Constants from post-build configuration	<code>&lt;MODULE&gt;_START_SEC_CONFIG_DATA_POSTBUILD_&lt;SIZE&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_CONFIG_DATA_POSTBUILD_&lt;SIZE&gt;</code>	Constants with attributes that show that they reside in one segment for <b>post-build</b> module configuration.
ROM buffers in NVRAM	<code>&lt;MODULE&gt;_START_SEC_VAR_SAVED_RECOVERY_ZONE&lt;X&gt;</code> <code>&lt;MODULE&gt;_STOP_SEC_VAR_SAVED_RECOVERY_ZONE&lt;X&gt;</code>	To be used for ROM buffers of variables saved in non volatile memory. Description for <X> is missing in AUTOSAR.

Table 9 Overview of Memory Allocation Keywords for Code

Type of code	Syntax of memory allocation keywords	Comments
Normal code	<code>&lt;MODULE&gt;_START_SEC_CODE</code> <code>&lt;MODULE&gt;_STOP_SEC_CODE</code>	To be used for mapping code to application block, boot block, external flash etc. Usage within CDG: <b>Tasks from 5 ms to 20 ms.</b>
Fast code	<code>&lt;MODULE&gt;_START_SEC_CODE_FAST[_&lt;NUM&gt;]</code> <code>&lt;MODULE&gt;_STOP_SEC_CODE_FAST[_&lt;NUM&gt;]</code>	To be used for code that shall go into fast code memory segments. The optional suffix _<NUM> can qualify the expected access commonness, e.g. typical period of code execution. Usage within CDG: <b>Tasks faster than 5 ms and frequently used interrupts.</b> The optional suffix _<NUM> is not used (and seems to be removed in future AUTOSAR releases).
Slow code	<code>&lt;MODULE&gt;_START_SEC_CODE_SLOW</code> <code>&lt;MODULE&gt;_STOP_SEC_CODE_SLOW</code>	To be used for code that shall go into slow code memory segments. Usage within CDG: <b>Tasks slower than 20 ms and other very slow called functions.</b>
Library code	<code>&lt;MODULE&gt;_START_SEC_CODE_LIB</code> <code>&lt;MODULE&gt;_STOP_SEC_CODE_LIB</code>	To be used for code that shall go into library segments for BSW module or Software Component.
Callout code	<code>&lt;MODULE&gt;_START_SEC_CALLOUT_CODE</code> <code>&lt;MODULE&gt;_STOP_SEC_CALLOUT_CODE</code>	To be used for mapping callouts of the BSW modules.

**Hint** The inclusion of the memory mapping header files within the code is a MISRA violation. As neither executable code nor symbols are included (only #pragmas) this violation is an approved exception without side effects.



### Rule Abstr\_MemMap\_003: Exception of Memory Mapping Concept: Function Local Variables

**Instruction** For function local variables within a C function no memory mapping is possible therefore memory mapping concept shall not be used. Memory mapping header files shall not be included inside the body of a function.

Function local variables are out of scope for memory mapping concept. There is nothing to be done for such variables.

Example:

```
void process(void)
{
    uint32 my_counting_var_u32; /* No memory mapping possible resp. needed */
}
```

### Rule Abstr\_MemMap\_005: No Usage inside Function Bodies

**Instruction** Memory mapping header files shall not be included inside the body of a function.

To avoid nested #pragmas it is forbidden to use memory mapping inside the body of a function. So any static variables or constants must not be defined inside of a function body.

**Hint** The [\[Abstr\\_MemMap\\_005\]](#) violates MISRA Rule 8.7. But this is accepted by AUTOSAR.

### Rule Abstr\_MemMap\_006: Usage of Module Name of Memory Mapping Keywords

**Instruction** For declarations the module name of the defining module has to be used for the memory mapping keywords.

To have a single source of information and to ensure that there is no conflicting information for the abstraction keywords always the module name of the defining module has to be used.

Example: See previous example for it.

## 3.2.5 Abstraction of Inline Functions

Inline functions are not really defined within C90 standard and therefore they are implicitly a violation of [\[CCode\\_001\]](#). But the benefit of an inline function is enormous so that it is now a valid extension.

Inline functions are quite similar to macros but they have a check of their parameters which is an important advantage. Identical to a macro is that inline functions are executed directly at that position where they are placed. Therefore they give a benefit in the runtime and RAM usage (because of saving of registers, no handover of parameters and no jump and return to and from the function), but, when using an inline function multiple times, it needs more code. Otherwise inline functions have the benefits that they need no stack memory, also the stack consumption can be reduced.

It has to be checked from case to case if an inline function can be used. Following points can help to decide when an inline function makes sense:

- ▶ The number of µc-cycles is less than the number of µc-cycles for a function call (mov to save parameters, call, ret). In other words: when the executed code for this functionality is smaller or equal to the overhead which is needed for a call and return of a sub function.
- ▶ The function is called only once. This also helps in the structuring of huge functions.

- ▶ The function is used very often, but is very small in code size. (The other way round: An inline function should not be used when the function is used very often and has a large size and therefore the common code size increases. In that case the runtime benefit could be secondary.)
- ▶ The function is called in a fast raster. Then it could be that the runtime advantage is more important and the disadvantage in code size could be acceptable.

## Rule Abstr\_Inline\_001: Usage of LOCAL\_INLINE

**Instruction** Direct use of compiler and platform specific inline keywords like `"__inline__"` or `"_inline"` are not allowed. To define an inline function macro `"LOCAL_INLINE"` shall be used.

Some compilers need special syntaxes to define an inline function. And sometimes attributes are needed that an inline function is really implemented as such one. The macro `"LOCAL_INLINE"` encapsulates all needed keywords and properties to define an inline function.

If the inline function is used multiple times in different c files or the inline function is an exported function of the module it has to be defined in a header file. If the inline function is only used in a single c file it can be defined in the corresponding c file.

The prototype has to be defined too, at best at top of the respective header or c file where the inline function is defined.

Example:

*Prototype and definition has to be placed in a header file (if the inline function is used multiple times in different c files) or it can be placed in the c file (if the inline function is only called inside the c file):*

```
/* Prototype: */
LOCAL_INLINE uint8 MyModule_Func(sint16 arg1, sint16 arg2);

/* Definition: */
LOCAL_INLINE uint8 MyModule_Func(sint16 arg1, sint16 arg2)
{
    <code>
}
```

*The inline function can be called within a c file:*

```
/* Call of the inline function: */
MyModule_Func(MyArg1, MyArg2);
```

## Rule Abstr\_Inline\_002: Inlines Without RTE APIs

**Instruction** Within inline functions no *RTE* APIs shall be called.

There shall be no accesses to Ports, InterRunnableVariables or Exclusive Areas within an inline function. It is a problem of ARXML description of RTE APIs which are used in inline functions. Because inline functions are executed directly where they are called all existing RTE APIs inside the inline function are executed too.

Usually all RTE APIs that are used have to be described via ARXML files. Sometimes the names of ports and runnables are part of the name of the RTE API. This results in a conflict that an inline functions can be used more than once in different runnables and thus the name of the RTE APIs within the inline function cannot be constant. Therefore normal functions shall be used and RTE APIs shall not be called in inline functions.



## 3.2.6 Handling of Assembler Instructions

### Rule Abstr\_Asm\_001: Usage of Assembler Code

**Instruction** Assembler instructions inside C code shall not be used unless they are encapsulated and isolated for special cases.

Generally assembler code is not portable because it is  $\mu$ C dependent and each  $\mu$ C has its own set of assembler instructions. This is a reason to use an abstract language like C for coding. With this one is able to create target specific code by different compilers.

Additionally inline assembler is an extension to C90 standard and therefore a forbidden add-on, [CCode\_001]. Nevertheless there are exceptions where an assembler block is needed, for example to reduce runtime of library services. Here assembler instructions can be used if they are encapsulated and selectable via configuration mechanism. In parallel a C implementation has to exist as portable code. By an individual configuration setting the assembler implementation is selectable.

## 3.2.7 Prohibition of Dynamic Memory

### Rule Abstr\_DynMem\_001: Prohibition of Dynamic Heap Memory

**Instruction** Dynamic heap memory allocation shall not be used.

This prohibits the use of the functions `calloc`, `malloc`, `realloc` and `free`. There is a whole range of unspecified, undefined and implementation-defined behaviour associated with dynamic memory allocation, as well as a number of other potential pitfalls. Dynamic heap memory allocation may lead to memory leaks, data inconsistency, memory exhaustion, non-deterministic behaviour.

### Rule Abstr\_DynMem\_002: Usage of Memory Areas

**Instruction** An area of memory shall not be reused for unrelated purposes.

This rule refers to the technique of using memory to store some data, and then using the same memory to store unrelated data at some other time during the execution of the program. Clearly it relies on the two different pieces of data existing at disjoint periods of the program's execution, and never being required simultaneously.

This practice is not recommended for safety-related systems as it brings with it a number of dangers. For example: a program might try to access data of one type from the location when actually it is storing a value of the other type (e.g. due to an interrupt). The two types of data may align differently in the storage, and overwrite upon other data. Therefore the data may not be correctly initialized every time the usage switches. The practice is particularly dangerous in concurrent systems.

However it is recognised that sometimes such storage sharing may be required for reasons of efficiency. Where this is the case it is essential that measures are taken to ensure that the wrong type of data can never be accessed, that data is always properly initialized and that it is not possible to access parts of other data (e.g. due to alignment differences). The measures taken shall be documented and justified in the derivation that is raised against this rule.

There is no intention to place restrictions on how a compiler actually translates source code as the user generally has no effective control over this. So for example, memory allocation within the compiler whether dynamic heap, dynamic stack or static, may vary significantly with only slight code changes even at the same level of optimisation.

## 3.2.8 Pointer Type Conversions

Only certain types of pointer conversions are defined in C and the behaviour of some conversions is implementation--defined. With a focus on portability such pointer conversions are not allowed and the defined types of pointer conversions shall be used very carefully. Generally pointer types can be classified as follows:

- ▶ Pointer to object (value, array, struct, ...)
- ▶ Pointer to function
- ▶ Pointer to void
- ▶ The null pointer constant (the value 0 to type void \*)

Conversions involving pointer types require an explicit cast except when:

- ▶ The conversion is between a pointer to object and a pointer to void, and the destination type carries all the type qualifiers of the source type
- ▶ A null pointer constant (void \*) is converted automatically to a particular pointer type when it is assigned to or compared for equality with any type of pointer

These are valid pointer conversions, others are not allowed.

### Rule Abstr\_PtrConv\_001: Prohibited Function Pointer Conversions

**Instruction** Conversions shall not be performed between a pointer to a function and any type other than an integral type.

An integral type is a signed integer type, unsigned integer type, char type or an enumeration type. A Conversion of a function pointer to a different type of pointer results in an undefined behaviour. That means that a pointer to a function cannot be converted to a pointer to a different type of function or to an object or to a floating type. Such conversions (and vice versa) are mostly not defined and therefore shall not be used.

### Rule Abstr\_PtrConv\_002: Prohibited Object Pointer Conversions

**Instruction** Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.

Casting an expression of floating type to a pointer or vice versa is not defined. Therefore such conversions are not allowed.

### Rule Abstr\_PtrConv\_003: Prohibition of Casts between Pointer and Integral Types

**Instruction** A cast should not be performed between a pointer type and an integral type.

The size of integer that is required when a pointer is converted to an integer is implementation-defined. Concerned are casts, for example, between an integral type and a pointer to volatile or non-volatile object or to a function.

Casting between a pointer and an integer type should be avoided where possible, but may be unavoidable when memory mapped register or other hardware specific features have to be addressed. Sometimes modules of *MCAL* have this use case and accordingly this rule cannot be kept from them.

## Rule Abstr\_PtrConv\_004: Prohibition of Casts between Object Pointer Types

**Instruction** A cast should not be performed between a pointer to object type and a different pointer to object type. Sole exception is a cast to an uint8 object type because uint8 has the smallest alignment.

If the type of the new pointer has a stricter alignment a conversion to that pointer type may be invalid. This applies to all pointers to object types with the sole exception of an uint8 pointer object type. Independent from the addressing architecture of a processor an uint8 pointer can address each element in memory because uint8 has always the smallest alignment. Bigger types than uint8 have a bigger alignment and potential to address elements can be restricted by the architecture of the processor.

Examples:

```
uint8 * Ptr_pu8;
uint32 * Ptr_pu32;

Ptr_pu32 = (uint32 *)Ptr_pu8;    /* Not OK, because alignment could be incompatible */

Ptr_pu8 = (uint8 *)Ptr_pu32;    /* OK because uint8 has the smallest alignment */
                                /* The uint8 pointer can here be used e.g. to */
                                /* copy the values of the uint32 element within a */
                                /* memcpy service. */
```

## Rule Abstr\_PtrConv\_005: Preservation of Pointer Qualifications

**Instruction** A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.

Any attempt to remove the qualification associated with the addressed type by using casting is a violation of the principle of type qualification. Notice that the qualification referred to here is not the same as any qualification that may be applied to the pointer itself.

Example:

```
uint16      a_u16;                /* uint16 variable */
uint16 * const b_cpu16 = &a_u16; /* const pointer to uint16 value */
const uint16 * c_pcu16 = 0;      /* Pointer to constant value */
volatile uint16 * d_pvul16 = 0;  /* Pointer to volatile value */
uint16      * e_pu16;            /* Pointer to uint16 */

e_pu16 = b_cpu16;                /* OK because it is only a pointer to a */
                                /* const pointer, no cast is needed */
e_pu16 = (uint16 *)c_pcu16;      /* Not OK: const qualifier gets lost */
e_pu16 = (uint16 *)d_pvul16;     /* Not OK: volatile qualifier gets lost */
```

## 3.2.9 Pointer Arithmetic and Arrays

### Rule Abstr\_PtrArith\_001: Allowed Form of Pointer Arithmetic

**Instruction** Pointer arithmetic shall only be applied to pointers that address an array or array element.

Addition and subtraction of integers (including increment and decrement) from pointers that do not point to an array or array element results in undefined behaviour.

### Rule Abstr\_PtrArith\_002: Allowed Form of Pointer Subtraction

**Instruction** Pointer subtraction shall only be applied to pointers that address elements of the same array.

Subtraction of pointers only gives well-defined results if the two pointers point (or at least behave as if they point) into the same array object.

### Rule Abstr\_PtrArith\_003: Allowed Form of Comparison of Pointers

**Instruction** `>`, `>=`, `<`, `<=` shall not be applied to pointer types except where they point to the same array.

Attempting to make comparisons between pointers will produce undefined behaviour if the two pointers do not point to the same object.

Note: it is permissible to address the next element beyond the end of an array, but accessing this element is not allowed.

### Rule Abstr\_PtrArith\_004: Validity of Pointer Objects

**Instruction** The address of an object with automatic storage shall not be assigned to another object that may be used after the first object has ceased to exist.

If the address of an automatic object is assigned to another automatic object of larger scope, or to a static object, or returned from a function then the object containing the address may exist beyond the time when the original object ceases to exist (and its address becomes invalid).

Example:

```
extern uint8 * ExtModule_pu8;

uint8 * MyModule_Func(void)
{
    uint8 local_auto;
    uint8 *local_ptr;

    ...
    local_ptr = &local_auto;      /* OK, within function element is visible */

    ...
    return &local_auto;          /* Not OK: Return address of a local object */
}
...

ExtModule_pu8 = MyModule_Func(); /* Not OK: Automatic object is ceased after */
                                /* call of sub function */
```

### Rule Abstr\_PtrArith\_005: Maximum Number of Pointer Indirection

**Instruction** The declaration of objects should contain no more than 2 levels of pointer indirection.

Use of more than 2 levels of indirection can seriously impair the ability to understand the behaviour of the code, and should therefore be avoided.

Examples:

```
void MyFunc(uint8 *   Arg1,          /* OK, normal pointer to uint8 */
            uint8 ** Arg2,          /* OK, pointer to pointer to uint8 */
            uint8 *** Arg3,         /* Not OK, triple pointer to uint8 */
            uint8 * const * const Arg4, /* OK, expanded to a type of const pointer to
                                     /* a const pointer to uint8
            uint8 ** Arg5[]))        /* Not OK, because triple pointer to uint8
                                     /* Arrays are converted to a pointer to the
                                     /* initial element of the array

{
    uint8 *   Ptr1;                 /* OK, normal pointer to uint8 */
    uint8 **  Ptr2;                 /* OK, pointer to pointer to uint8 */
    uint8 *** Ptr3;                 /* Not OK, triple pointer to uint8 */
}
```

```

uint8 * * const * const Ptr4;      /* Not OK, expanded to a type of const pointer */
                                   /* to a const pointer to a pointer to uint8 */
uint8 ** Ptr5[10];                /* OK, Ptr5 is of type array of pointer to */
...                                /* pointer to uint8 */
}

```

### 3.2.10 Ensure Usability of BSW Modules in C++ Environments

Some product lines are using C++ compilers for their application software. It has to be ensured that such projects also can use the BSW modules which are programmed with the C language. Problems can occur if the C++ code calls macros or inline functions from the BSW modules. Therefore no C code construct should be used in macros and inline functions defined in BSW interface header files, that could inhibit the compilation of C++ source code including those interface header files or that could lead to errors in the C++ code. The following rules specify how the C based software can safely be used in C++ projects. Also hints are given how to check that rule to ensure the compatibility.

Note that the rules in this chapter are no enumeration of all differences between C and C++. Only those differences are mentioned which are relevant for the product line spanning base software development and which are not yet covered by other rules in this Coding Guidelines. For the exact details of the differences between C and C++ see the C++ ISO/IEC 14882 2011-09, appendix C "Compatibility".

#### Rule Abstr\_CPP\_001: Don't use C++ Keywords not Defined as Keywords in C

**Instruction** C++ defines new keywords in addition to C. Externally usable macros and inline functions shall not use those keywords. The respective keywords are:

- ▶ alignas alignof and and\_eq asm
- ▶ bitand bitor bool
- ▶ catch char16\_t char32\_t class compl constexpr const\_cast
- ▶ decltype delete dynamic\_cast
- ▶ explicit export
- ▶ false friend
- ▶ inline
- ▶ mutable
- ▶ namespace new noexcept not not\_eq nullptr
- ▶ operator or or\_eq
- ▶ private protected public
- ▶ reinterpret\_cast
- ▶ static\_assert static\_cast
- ▶ template this thread\_local throw true try typeid typename
- ▶ using
- ▶ virtual
- ▶ wchar\_t

► xor xor\_eq

*Rule Check:* This rule can be checked by simply compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language constructs.

## Rule Abstr\_CPP\_002: Type of Character Literal

**Instruction** Externally usable macros and inline functions shall not depend on the type of character literal.

While in C `sizeof('a') == sizeof(int)` holds but in C++ `sizeof('a') == sizeof(char)` is valid.

*Rule Check:* This rule cannot be checked by compilation only.

## Rule Abstr\_CPP\_003: Use Cast to Assign String Literals

**Instruction** Externally usable macros and inline functions shall use a cast to non-const type when a string literal is assigned to a pointer.

The type of a string literal is in C++ changed from "array of char" to "array of const char".

```
char* p = "abc";           /* Not OK: valid in C, invalid in C++      */
char* p = (char*)"abc";    /* OK: cast added, usable for C and C++    */
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_004: Don't use Doubled Definition

**Instruction** Externally usable macros and inline functions shall not use repeated type definitions.

Example:

```
uint8 var_u8;
...
uint8 var_u8;           /* Not OK: is valid in C but invalid in C++ */
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_005: Don't Refer to Structs Defined in Structs from Outside

**Instruction** Externally usable macros and inline functions shall not refer to structs, enumerations or enumerator names outside the struct in which those are defined.

In C, names of nested structures are placed in the same scope as the structure in which they are nested. For example:

```
struct S
{
    struct T
    {
        ...
    };
    ...
};

struct T x;           /* OK in C meaning "S::T x;" but Not OK in C++ */
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_006: Declare const Objects with External Linkage in C code Explicitly extern

**Instruction** Externally usable macros and inline functions shall declare *const* objects with external linkage in C code explicitly *extern*.

A name of file scope that is explicitly declared *const*, and not explicitly declared *extern*, has internal linkage in C++, while in C it has external linkage.

**Rule Check:** This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_007: Use Cast to Convert void\* to a Pointer Type

**Instruction** Externally usable macros and inline functions shall use a cast to convert a void\* to a pointer type.

Converting void\* to a pointer-to-object type requires casting in C++, see example:

```
uint8 array_au8[10];
void *b = array_au8;

void foo()
{
    uint8 *c = b;           /* Not OK: because it is only valid in C */
    uint8 *c = (uint8 *)b;  /* OK: Valid in C and C++ */
}
```

**Rule Check:** This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_008: Only Pointers to non-const and non-volatile Objects may be Implicitly Converted to void\*

**Instruction** Within externally usable macros and inline functions only pointers to non-const and non-volatile objects may be implicitly converted to void\*.

**Rule Check:** This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_009: const Objects shall be Initialized

**Instruction** Externally usable macros and inline functions shall initialize const objects.

const objects shall be initialized in C++ but can be left uninitialized in C.

**Rule Check:** This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## Rule Abstr\_CPP\_010: Don't Use auto as a Storage Class Specifier

**Instruction** Externally usable macros and inline functions shall not use auto as a storage class specifier.

**Example:**

```
void ModuleFunc()
{
    auto uint8 x_u8;           /* Not OK: because it is invalid in C++ */
}
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

### Rule Abstr\_CPP\_011: Assign only Values of the same Type to Enumeration Objects

**Instruction** Within externally usable macros and inline functions only values of the same type shall be assigned to enumeration objects.

Example:

```
enum color { red, blue, green };
enum color c = 1;          /* Not OK: because it is invalid in C++ */
}
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

### Rule Abstr\_CPP\_012: Don't let the Code Depend on the Type and Size of Enumeration Objects

**Instruction** Within externally usable macros and inline functions code shall not depend on the type and size of enumeration objects.

Example:

```
enum e { A };
sizeof(A) == sizeof(int) /* true in C */
sizeof(A) == sizeof(e)  /* true in C++ */
/* and sizeof(int) is not necessarily equal to sizeof(e) */
```

*Rule Check:* This rule cannot be checked by compilation only.

### Rule Abstr\_CPP\_013: Use a Cast or Separate Implementations to Copy Volatile Structs

**Instruction** Externally usable macros and inline functions code shall use a cast when copying volatile structs to non-volatile structs.

Copying volatile structs to volatile targets is in C++ only possible with user-defined constructors or assignments so that this can only be achieved with separated implementations (separated by `#ifdef __cplusplus`).

Example:

```
struct X
{
    int i;
};

volatile struct X x1 = {0};

struct X x2(x1);          /* Not OK: because it is invalid in C++ */
struct X x3;
x3 = x1;                  /* Not OK: because it is invalid in C++ */
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

### Rule Abstr\_CPP\_014: Don't use `__STDC__`

**Instruction** Within externally usable macros and inline functions the value of `__STDC__` shall not be used.

The value of `__STDC__` is implementation defined in C++.



*Rule Check:* This rule cannot be checked by compilation only.

### Rule Abstr\_CPP\_015: Encapsulate C code Conflicting with C++ and Provide C++ Substitution Code

**Instruction** If usage of C code conflicting with C++ cannot be avoided then that conflicting code shall be encapsulated and C++ compliant code shall also be provided.

Example:

```
#ifdef __cplusplus
/* Here C++ compliant code shall be provided */
#else
/* Here C compliant code shall be provided */
#endif
```

*Rule Check:* This rule can be checked by compiling test code calling all macros and inline functions with a C++ compiler sensitive for the according language construct.

## 3.2.11 Prohibition of Open Source Software

### Rule Abstr\_OSS\_001: Prohibition of Open Source Software

**Instruction** Open Source Software (OSS) shall not be used within BSW modules.

Open Source Software is often critical regarding licensing, liability, publication and loss of control regarding technical contents. Therefore Open Source Software shall not be used.

## 3.3 Rule Set: Types and Symbols

All platform data types and standard symbols of this rule set are defined within headers "Platform\_Types.h", "Std\_Types.h" and "ComStack\_Types.h". Header "Platform\_Types.h" is included in "Std\_Types.h", this header is included in "ComStack\_Types.h". Communication software c files have to include header "ComStack\_Types.h". All other SW has to include "Std\_Types.h". For more details see [Chapter 3.5.2 "Header Include Concept"](#).

It is strongly recommended that all platform data types and standard symbols are unique within the AUTOSAR community to guarantee unique types and to avoid type changes when changing from supplier A to B. All described types and symbols in this rule set corresponds to what AUTOSAR call standard types, platform types and comstack types. Those types and symbols are the base of BSW software and shall be used from every module (independent if a module has an AUTOSAR specification or not).

This rule set describes the usage of types and symbols for C code and headers. The counterpart of the description of types and symbols in ARXML format is illustrated in [Chapter 10 "Rule Set: Central Elements"](#). Refer to that chapter to get more information about types and symbols in ARXML format.

### 3.3.1 Base Integer and Float Data Types

In this chapter all valid base integer and float data types are described.

#### Rule CCode\_Types\_001: Platform Integer and Float Data Types

**Instruction** Following common platform integer and float data types are allowed and can be used within SW and APIs: boolean, uint8, sint8, uint16, sint16, uint32, sint32, float32, float64.

The float data types shall be used carefully and their usability shall be ensured.

An overview over all Platform integer and float data types is given in [Table 10](#).

Table 10 Overview Platform Integer and Float Data Types

Type	Range	Description
<b>boolean</b>	0...1 FALSE, TRUE	boolean unsigned integer
<b>uint8</b>	0 ... 255 0x00 ... 0xFF	8 bit unsigned integer
<b>sint8</b>	-128 ... +127 0x80 ... 0x7F	8 bit signed integer
<b>uint16</b>	0 ... 65535 0x0000 ... 0xFFFF	16 bit unsigned integer
<b>sint16</b>	-32768 ... +32767 0x8000 ... 0x7FFF	16 bit signed integer
<b>uint32</b>	0 ... 4294967295 0x00000000 ... 0xFFFFFFFF	32 bit unsigned integer
<b>sint32</b>	-2147483648 ... +2147483647 0x80000000 ... 0x7FFFFFFF	32 bit signed integer
<b>float32</b>	1.17549435E-38 ... +3.40282347E38	Single precision (32 bit) floating point number (consider 2nd hint below)
<b>float64</b>	2.225E-308 ... +1.797E308	Double precision (64 bit) floating point number (consider 2nd hint below)

**Hint** Concerning the signed integer types only two's complement arithmetic is supported. This directly impacts the chosen ranges for these types.

**Hint** On different  $\mu$ Cs often no floating point unit is available. Therefore float data types (float32 and float64) have to be used carefully. It has to be ensured that float can be used. Additionally float64 is more critical than float32 because  $\mu$ C supports more often float32 than float64.

Float data types shall not be used if the developed SW has a focus on complete HW and compiler independence. In such a use case float data types are not portable.

## Rule CCode\_Types\_002: Prohibition of Plain C Data Types

**Instruction** The usage of plain C data types "int", "short" and "long" is forbidden.

The size and the sign of the native C data types are not unambiguously defined. Code is only portable and reusable if base data types are used. Here size and sign is well defined for each platform.

## Rule CCode\_Types\_003: Usage of Char Data Type

**Instruction** The plain "char" data type shall only be used for the storage and use of character values or data.

Arrays or single values can be defined with data type "char" to handle strings or single characters. Outside of that usage char data type shall not be used. The signedness of the plain char data type is implementation-defined and should not be relied upon. The only permissible operators on plain char data types are assignment and equality operators (=, ==, !=).

## Rule CCode\_Types\_004: Handling of 64 Bit Data Types

**Instruction** 64 bit integer data types are no base data types and shall not be used in APIs. A local definition is allowed (e.g. libraries) but shall only used in exceptional cases.

64 bit integer data types are based on C type specifier "long long" which is defined in ISO C99 standard initially. Therefore 64 bit data types are in conflict with [CCode\_001]. Additionally 64 bit variables are not portable and can be problematical concerning "Rule Set: Compiler Abstraction / Portability". But sometimes 64 bit integer variables are necessary e.g. in mathematical calculations inside libraries. They are allowed to be used internally if portability of code is ensured (if the µC platform can handle it). Otherwise calculations in 64 bit context have to be implemented without using 64 bit integer data types (emulation of 64 bit).

## Rule CCode\_Types\_005: Handling of Own Defined Data Types

**Instruction** Do not define own data types based on base data types if this is not necessary and the data width is known at specification time.

This rule helps to avoid a flood of different cryptic types and improves readability of source code.

Example 1: The parameter *DeviceIndex* is known during specification time (8 bit).

*Do not define it in following way:*

```
typedef uint8 DeviceIndexType;  
...  
static DeviceIndexType DeviceIndex
```

*Please define it like this:*

```
static uint8 DeviceIndex
```

Example 2: The parameter *DeviceAddress* is platform dependent (could be 16 or 32 bit). It is required for runtime efficiency that the best type is chosen for a specific platform.

*On 16 bit platforms:*

```
typedef uint16 DeviceAddressType;
```

*On 32 bit platforms:*

```
typedef uint32 DeviceAddressType;
```

**Hint** This rule has its focus in definition of data types based on base data types (redefinition of base data types). Typedefs for enumerators and structures are not affected and can still be used.

**Hint** If within a specification of AUTOSAR a data type would be defined which is contrary to this rule, this data type shall be defined conformal to AUTOSAR. AUTOSAR has the preference.

## 3.3.2 Optimized Integer Data Types

### Rule CCode\_Types\_006: Common Optimized Integer Data Types

**Instruction** Following common optimized integer data types are allowed and can be used in a local scope inside a module but not in public APIs: `uint8_least`, `sint8_least`, `uint16_least`, `sint16_least`, `uint32_least`, `sint32_least`.

The optimized integer data types (`<typename>_least`) shall have at least the size given by the type name. They are implemented in that way that the best performance on the specific platform is achieved. "Best performance" is defined in this context as "least processor cycles for variable access as possible".

Example: on a TC1796, uint8\_least is mapped to unsigned int (32 bit) because access to this type requires less processor cycles than e.g. unsigned char (8 bit).

Table 11 Overview Optimized Data Types

Type	Range	Description
<b>uint8_least</b>	At least 0...255 At least 0x00...0xFF	At least 8 bit unsigned integer
<b>sint8_least</b>	At least -128...+127 At least 0x80...0x7F	At least 8 bit signed integer
<b>uint16_least</b>	At least 0...65535 At least 0x0000 ... 0xFFFF	At least 16 bit unsigned integer
<b>sint16_least</b>	At least -32768...+32767 At least 0x8000...0x7FFF	At least 16 bit signed integer
<b>uint32_least</b>	At least 0...4294967295 At least 0x00000000...0xFFFFFFFF	At least 32 bit unsigned integer
<b>sint32_least</b>	At least -2147483648...+2147483647 At least 0x80000000...0x7FFFFFFF	At least 32 bit signed integer

**Hint** The optimized integer data types "uint32\_least" and "sint32\_least" are not really useful. Up to now for all available processors these data types have a size of 32 bits and therefore are equivalent to uint32 resp. sint32. But they are a part of the AUTOSAR standard, therefore they are a part of the table above.

**Hint** Optimized integer data types shall replace old previous data types like "int", "uint" or "sint" used in same context. Optimized data types are "better" because the data range limitation is visible.

**Hint** It could be a problem if inside a function a variable based on an optimized data type is used and from that variable a return value based on a standard data type is derived. A cast is needed to convert the optimized data type to the standard data type. In such a case it is better to use the final return data type inside the function instead of the optimized data type.

As an exception locally used inline functions can use optimized data types for parameters and return values.

## Rule CCode\_Types\_007: Handling of Optimized Integer Data Types

**Instruction** Operations on the optimized integer data types (<typename>\_least) shall not expect a specific size of this type. The size specified by the name is guaranteed, but can be larger. It is not allowed to use rollover mechanisms during counting and shifting.

Examples of usage:

- ▶ Loop counters (e.g. maximum loop count = 124 ? use uint8\_least)
- ▶ Switch case arguments (e.g. maximum number of states = 17 ? use uint8\_least)

### 3.3.3 Standard Symbols

In this chapter all standard symbols are described.

## Rule CCode\_Symbols\_001: TRUE and FALSE

**Instruction** TRUE is defined as "1" and FALSE is defined as "0" can be used in conjunction with the standard data type "boolean" and should not be redefined.

TRUE and FALSE are defined in following way:

```
#ifndef TRUE
    #define TRUE    1
#endif

#ifndef FALSE
    #define FALSE    0
#endif
```

TRUE and FALSE have to be used in conjunction with the standard data type boolean. A correct assignment is needed.

```
extern boolean MyFunc(void);
boolean var_b;
...
var_b = TRUE;
var_b = FALSE;
var_b = MyFunc();
```

## Rule CCode\_Symbols\_002: CPU Specific Symbols

**Instruction** Following CPU specific common symbols are available: CPU\_TYPE and CPU\_BYTE\_ORDER.

Each specified symbol has an adequate value for an appropriate processor / compiler. The symbols can be used if a specific dependency to CPU type, bit or byte order is needed.

Table 12 Overview Common Symbols

Symbol	Values	Description
<b>CPU_TYPE</b>	<b>CPU_TYPE_8</b>	indicates a 8 bit processor
	<b>CPU_TYPE_16</b>	indicates a 16 bit processor
	<b>CPU_TYPE_32</b>	indicates a 32 bit processor
<b>CPU_BYTE_ORDER</b>	<b>HIGH_BYTE_FIRST</b>	indicates that within an uint16, the high byte is located before the low byte.
	<b>LOW_BYTE_FIRST</b>	indicates that within uint16, the low byte is located before the high byte.

Example to use CPU specific symbols:

```
#if (CPU_BYTE_ORDER == HIGH_BYTE_FIRST)
{
    ...
    /* Do all the things which has to be done if high byte order is active */
}
#else
{
    ...
    /* Do all the things which has to be done if low byte order is active */
}
#endif
```

## Rule CCode\_Symbols\_003: Common Symbols

**Instruction** Following other common symbols are usable: E\_OK, E\_NOT\_OK, STD\_HIGH, STD\_LOW, STD\_ACTIVE, STD\_IDLE, STD\_ON, STD\_OFF.

Table 13 Overview Other Common Symbols

Symbol	Value	Description
<b>E_OK</b>	0x00	Positive result, used within Std_ReturnType
<b>E_NOT_OK</b>	0x01	Negative result, used within Std_ReturnType
<b>STD_HIGH</b>	0x01	Physical state 5V or 3.3V
<b>STD_LOW</b>	0x00	Physical state 0V
<b>STD_ACTIVE</b>	0x01	Logical state active
<b>STD_IDLE</b>	0x00	Logical state idle
<b>STD_ON</b>	0x01	Logical state on
<b>STD_OFF</b>	0x00	Logical state off

## Rule CCode\_Symbols\_004: Standard Version Info Type

**Instruction** Type "Std\_VersionInfoType" shall be used to request the version of a BSW module.

In detail the type Std\_VersionInfoType has following structure:

```
typedef struct
{
    uint16 vendorID;
    uint16 moduleID;
    uint8 sw_major_version;
    uint8 sw_minor_version;
    uint8 sw_patch_version;
} Std_VersionInfoType;
```

Explanation of the elements of the structure:

- ▶ "vendorID" represents the vendor identification number defined from HIS.
- ▶ "moduleID" represents the module identifier.
- ▶ "sw\_major\_version", "sw\_minor\_version" and "sw\_patch\_version" represents the major/minor/patch version of the vendor specific implementation of the module.

This type shall be used to request the version of an AUTOSAR based module using the <Module>\_GetVersionInfo() function. More details about that API can be found in [\[BSW\\_VersionInfo\\_005\]](#).

## Rule CCode\_Symbols\_005: Usage of Null Pointer

**Instruction** For null pointers "NULL\_PTR" shall be used.

NULL\_PTR is defined as void pointer to zero definition (void \*)0. This symbol is defined in header "Compiler.h".

NULL\_PTR can be used for pointer checks (e.g. if (ptr != NULL\_PTR) ), to initialize pointers, but shall not be used as parameter in a call of a function with pointer argument.

NULL which represents the value "0" is not defined as macro. Here "0" can be used directly.

## 3.3.4 Specific Types and Symbols for Communication Software

This chapter specifies the AUTOSAR communication stack types. All types are used across several modules of the communication stack of the basic software. These types are defined in header "ComStack\_Types.h". Communication related SW modules shall include this header to get all base data types, standard symbols and communication specific types. Inside "ComStack\_Types.h" "Std\_Types.h" is included by default. For more details see ["Rule Set: Module Design and Implementation"](#).

## Rule CCode\_ComStackTypes\_001: Type for PDU Identifiers

**Instruction** Type "PduIdType" has to be used to define variables for unique identifiers for PDUs.

"PduIdType" is based on standard integer data type "uint8" or "uint16" depending on the maximum number of PDUs used within one software module. If no software module deals with more PDUs than 256, this type can be set to uint8. If at least one software module handles more than 256 PDUs, this type shall globally be set to uint16. The maximum number of a PduId is the count of PDUs minus 1 (256 PDUs --> range of PduIds from 0 to 255), as defined by the corresponding type of operation within that module.

Variables of type "PduIdType" serve as an unique identifier of a PDU within a software module or a set thereof, and also for interaction of two software modules where the PduId of the corresponding target module is being used for referencing.

## Rule CCode\_ComStackTypes\_002: Handling of Types for PDU Identifiers

**Instruction** Variables of type "PduIdType" shall be zero-based and consecutive, in order to be able to perform table-indexing within a software module.

There might be several ranges of PduIds in a module, one for each type of operation performed within that module (e.g. sending and receiving).

## Rule CCode\_ComStackTypes\_003: Type for Length Information of a PDU

**Instruction** Type "PduLengthType" shall be used to define length information of a PDU which is provided in number of bytes.

The size "PduLengthType" depends on the maximum length of PDUs to be sent by an ECU. The maximum length of a PDU is the length of the largest (possibly segmented) PDU to be sent by the ECU.

## Rule CCode\_ComStackTypes\_004: Type for Basic Information of a PDU

**Instruction** Type "PduInfoType" has to be used to store the basic information about a PDU.

"PduInfoType" is defined in following form:

```
typedef struct
{
    P2VAR(uint8, AUTOMATIC, AUTOSAR_COMSTACKDATA) SduDataPtr;
    PduLengthType SduLength;
} PduInfoType;
```

"SduDataPtr" is an uint8 pointer to the SDU (Service Data Unit, i.e. payload data) of the PDU. The type of this pointer depends on the memory model being used at compile time.

"SduLength" is the length of the SDU in bytes.

**Hint** P2VAR and AUTOMATIC shown in example above are from concept to encapsulate keywords for addressing of data and code [Chapter 3.2.3 "Abstraction of Addressing Keywords"](#). This concept is still not elaborated within this guideline. The example above is taken from an AUTOSAR specification.

## Rule CCode\_ComStackTypes\_005: Type for Result of a Buffer Request

**Instruction** Type "BufReq\_ReturnType" shall be used to define variables to store the result of a buffer request.

"BufReq\_ReturnType" is defined in following form. [Table 14](#) shows the explanation of each enumerator value.

```
typedef enum
{
    BUFREQ_OK,
    BUFREQ_E_NOT_OK,
    BUFREQ_E_BUSY,
    BUFREQ_E_OVFL
} BufReq_ReturnType;
```

**Caution** BUFREQ\_E\_BUSY shall not be used because of proprietary reasons.

Table 14 Explanation of each Enumerator Values of Type "BufReq\_ReturnType"

Symbol	Value	Description
<b>BUFREQ_OK</b>	0x00	Buffer request accomplished successful.
<b>BUFREQ_E_NOT_OK</b>	0x01	Buffer request not successful. Buffer cannot be accessed.
<b>BUFREQ_E_BUSY</b>	0x02	Temporarily no buffer available. It's up the requester to retry request for a certain time.
<b>BUFREQ_E_OVFL</b>	0x03	No Buffer of the required length can be provided. .

## Rule CCode\_ComStackTypes\_006: Type for Result Status of a Notification

**Instruction** Type "NotifResultType" shall be used to define variables to store the result status of a notification (confirmation or indication).

"NotifResultType" is based on standard integer data type "uint8". [Table 15](#) shows an overview of all valid ranges within this type.

Table 15 Overview of all Valid Ranges within Type "NotifResultType"

Range	Description
<b>0x00 – 0x1E</b>	General return codes. A detailed specification is listed in <a href="#">Table 16</a> . Please consider that there are small differences between AUTOSAR release R3.1 and R4.0 which are shown in table.
<b>0x1F – 0x3C</b>	Error notification codes specific for the communication system CAN. For a detailed definition please refer to the AUTOSAR specification of CAN TP [CANTP].
<b>0x3D – 0x5A</b>	Error notification codes specific for the communication system LIN. A detailed definition is still open, because currently there is not AUTOSAR specification of Lin TP.
<b>0x5B – 0x78</b>	Error notification codes specific for the communication system FlexRay. For a detailed definition please refer to the AUTOSAR specification of FlexRay TP [FlexRayTP]
<b>&gt; 0x78</b>	Currently values in this range are invalid. In future further return codes might be specified for other communication systems.

Table 16 Overview Valid General Return Codes for "NotifResultType"

Return code	Value	Description
<b>NTFRSLT_OK</b>	0x00	Action has been successfully finished: <ul style="list-style-type: none"> <li>▶ message sent out (in case of confirmation)</li> <li>▶ message received (in case of indication)</li> </ul>
<b>NTFRSLT_E_NOT_OK</b>	0x01	Error notification: <ul style="list-style-type: none"> <li>▶ message not successfully sent out (in case of confirmation)</li> <li>▶ message not successfully received (in case of indication)</li> </ul>



Return code	Value	Description
<b>NTFRSLT_E_TIMEOUT_A</b>	0x02	Error notification: <ul style="list-style-type: none"> <li>timer N_Ar/N_As (according to ISO specification [ISONM]) has passed its time-out value N_Asmax/N_Armax.</li> </ul> This value can be issued to service user on both the sender and receiver side.
<b>NTFRSLT_E_TIMEOUT_BS</b>	0x03	Error notification: timer N_Bs has passed its time-out value N_Bsmax (according to ISO specification [ISONM]). This value can be issued to the service user on the sender side only.
<b>NTFRSLT_E_TIMEOUT_CR</b>	0x04	Error notification: timer N_Cr has passed its time-out value N_Crmax. This value can be issued to the service user on the receiver side only.
<b>NTFRSLT_E_WRONG_SN</b>	0x05	Error notification: unexpected sequence number (PCI.SN) value received. This value can be issued to the service user on the receiver side only.
<b>NTFRSLT_E_INVALID_FS</b>	0x06	Error notification: invalid or unknown FlowStatus value has been received in a flow control (FC) N_PDU. This value can be issued to the service user on the sender side only.
<b>NTFRSLT_E_UNEXP_PDU</b>	0x07	Error notification: unexpected protocol data unit received. This value can be issued to the service user on both the sender and receiver side.
<b>NTFRSLT_E_WFT_OVRN</b>	0x08	Error notification: flow control WAIT frame that exceeds the maximum counter N_WFTmax received. This value can be issued to the service user on the receiver side.
<b>NTFRSLT_E_ABORT</b>	0x09	Error notification: Flow control (FC) N_PDU with FlowStatus = ABORT received. It indicates an abort of a transmission. A possible reason for this is that the receiver is currently busy and cannot take the request at that point in time.
<b>NTFRSLT_E_NO_BUFFER</b>	0x0A	Error notification: Flow control (FC) N_PDU with FlowStatus = OVFLW received. It indicates that the buffer on the receiver side of a segmented message transmission cannot store the number of bytes specified by the FirstFrame DataLength (FF_DL) parameter in the FirstFrame and therefore the transmission of the segmented message was aborted. No buffer within the TP available to transmit the segmented I-PDU. This value can be issued to the service user on both the sender and receiver side.
<b>NTFRSLT_E_CANCELATION_OK</b>	0x0B	Action has been successfully finished: Requested cancellation has been executed.
<b>NTFRSLT_E_CANCELATION_NOT_OK</b>	0x0C	Error notification: Due to an internal error the requested cancellation has not been executed. This will happen e.g., if the to be cancelled transmission has been executed already.
<b>NTFRSLT_PARAMETER_OK</b>	0x0D	The parameter change request has been successfully executed.
<b>NTFRSLT_E_PARAMETER_NOT_OK</b>	0x0E	The request for the change of the parameter did not complete
<b>NTFRSLT_E_RX_ON</b>	0x0F	The parameter change request not executed successfully due to an ongoing reception
<b>NTFRSLT_E_VALUE_NOT_OK</b>	0x10	The parameter change request not executed successfully due to a wrong value

Return code	Value	Description
-	0x11 -- 0x1E	Reserved values for future usage.

**Hint** Currently this type is only used for communication between DCM and TP to enable the notification that an error has occurred and a dedicated buffer can be unlocked.

## Rule CCode\_ComStackTypes\_007: Naming of Return Codes of a Notification

**Instruction** Return codes of a notification shall be named as follows: NTFRSLT\_E\_<Communication System Abbreviation>\_<Error Code Name>.

Communication System Abbreviation:

- ▶ CAN: for Controller area network
- ▶ LIN: for Local Interconnect Network
- ▶ FR: for FlexRay

Error Code Name: self explaining name of error return code.

Example for a CAN specific return value:

NTFRSLT\_E\_FR\_NEG\_ACK: Negative acknowledgement on received

## Rule CCode\_ComStackTypes\_008: Type for Bus Status Evaluated by a Transceiver

**Instruction** Type "BusTrcvErrorType" shall be used to define variables to return the bus status evaluated by a transceiver.

"NotifResultType" is based on standard integer data type "uint8". [Table 17](#) shows an overview of all valid ranges within this type.

Table 17 Overview of all valid ranges within type "BusTrcvErrorType"

Range	Description
<b>0x00 – 0x1E</b>	General return codes. A detailed specification is listed in <a href="#">Table 18</a>
<b>0x1F – 0x3C</b>	Error notification: Error notification codes specific for the communication system CAN. For a detailed definition please refer to the AUTOSAR specification of CAN Transceiver Driver [CAN-TRCV].
<b>0x3D – 0x5A</b>	Error notification: Error notification codes specific for the communication system LIN. A detailed definition is still open, because currently there is not AUTOSAR specification of Lin Interface.
<b>0x5B – 0x78</b>	Error notification: Error notification codes specific for the communication system FlexRay. For a detailed definition please refer to the AUTOSAR specification of FlexRay Transceiver Driver [FRTRCV].
<b>&gt; 0x78</b>	Currently values in this range are invalid. In future it might be possible that further return codes are specified for other communication systems.

Table 18 Overview valid general return codes for "BusTrcvErrorType"

Return code	Value	Description
<b>BUSTRCV_OK</b>	0x00	There is no bus transceiver error seen by the driver, or the transceiver does not support the detection of bus errors.
<b>BUSTRCV_E_ERROR</b>	0x01	Bus transceiver detected an unclassified error.
-	0x02 -- 0x1E	Reserved values for future usage.

## Rule CCode\_ComStackTypes\_009: Naming of Return Codes of a Bus Status Notification

**Instruction** Return codes of a bus status notification shall be named as follows: BUSTRCV\_E\_<Communication System Abbreviation>\_<Error Code Name>.

Communication System Abbreviation:

- ▶ CAN: for Controller area network
- ▶ LIN: for Local Interconnect Network
- ▶ FR: for FlexRay

Error Code Name: self explaining name of error return code.

Example for a FlexRay specific return value:

BUSTRCV\_E\_CAN\_SINGLE: CAN bus transceiver has detected that the fault tolerant bus is in single wire mode.

## Rule CCode\_ComStackTypes\_010: Type for State of a Transport Protocol Buffer

**Instruction** Type "TpDataStateType" shall be used to define variables to store the state of a Transport Protocol (TP) buffer.

"TpDataStateType" is defined in following form. [Table 19](#) shows the explanation of each enumerator value.

Definition AR4.0.2:

```
typedef enum
{
    TP_DATACONF,
    TP_DATARETRY,
    TP_CONFPENDING,
    TP_NORETRY
} TpDataStateType;
```

Definition AR4.0.3:

```
typedef enum
{
    TP_DATACONF,
    TP_DATARETRY,
    TP_CONFPENDING
} TpDataStateType;
```

Table 19 Explanation of each enumerator values of type "TpDataStateType"

Element	Description
<b>TP_DATACONF</b>	TP_DATACONF indicates that all data, that have been copied so far, are confirmed and can be removed from the TP buffer. Data copied by this API call is excluded and will be confirmed later
<b>TP_DATARETRY</b>	TP_DATARETRY indicates that this API call shall copy already copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset of the first byte to be copied by the API call.

Element	Description
<b>TP_CONFENDING</b>	TP_CONFENDING indicates that the previously copied data shall remain in the TP.
<b>TP_NORETRY</b>	TP_NORETRY indicates that the copied transmit data can be removed from the buffer after it has been copied

### Rule CCode\_ComStackTypes\_011: Type for Transport Protocol Buffer Handling

**Instruction** Type "RetryInfoType" shall be used to define variables to store the information about TP buffer handling.

"RetryInfoType" is defined in following form:

```
typedef struct
{
    TpDataStateType TpDataState;
    PduLengthType TxTpDataCnt;
} RetryInfoType;
```

"TpDataState" is the enum type to be used to store the state of Tp buffer.

"TxTpDataCnt" is the length of the SDU in bytes.

### Rule CCode\_ComStackTypes\_012: Type for Identifiers of Communication Channels

**Instruction** Type "NetworkHandleType" shall be used to define variables to store the identifier of a communication channel.

"NetworkHandleType" is based on standard integer data type "uint8".

### Rule CCode\_ComStackTypes\_013: Type to Specify a Parameter

**Instruction** Type "TpParameterType" shall be used in "ChangeParameter" interfaces to specify the parameter to which the value has to be changed.

"TpParameterType" is defined as enumerator:

Definition AR4.0.2:

```
typedef enum
{
    STMIN,
    BS
} TPParameterType;
```

Definition AR4.0.3:

```
typedef enum
{
    TP_STMIN,
    TP_BS,
    TP_BC
} TPParameterType;
```

Table 20 Explanation of each enumerator values of type "TPParameterType"

Element	Description
<b>STMIN / TP_STMIN</b>	Separation Time
<b>BS / TP_BS</b>	Block Size
<b>TP_BC</b>	The Band width control parameter used in FlexRay transport protocol module

### 3.3.5 Module Specific Add Ons

#### Rule SpecificTypes\_5: Handling of Additional Types and Symbols

**Instruction** Module specific symbols may be defined and used if a module needs more symbols than are provided by standard symbols or communication software specific symbols.

In addition to standard symbols and communication software specific symbols module specific symbols can be defined and used. Such additional symbols have to be defined module specific. This will avoid that auxiliary common headers have to be provided in addition to a module. It is helpful for the module to have the control over module specific symbols, and delivery process has a lower complexity.

As an example for module specific symbols minimum and maximum limits for integer data types can be defined. Such symbols are not defined in the common types and symbols headers. [Table 21](#) will give an example how to define such symbols in a module specific header. Depending on the visibility of such symbols (internal or external) their definition can be made in a private or global header of the module.

Table 21 Overview of symbolic constants for maximum and minimum limits

Symbolic constant	Value	Definition	Description
<b>&lt;MODULE&gt;_MAXUINT8</b>	255 0xFF	(0xff)	Maximum value of standard data type "uint8"
<b>&lt;MODULE&gt;_MINUINT8</b>	0 0x00	(0x0)	Minimum value of standard data type "uint8"
<b>&lt;MODULE&gt;_MAXSINT8</b>	127 0x7F	(0x7f)	Maximum value of standard data type "sint8"
<b>&lt;MODULE&gt;_MINSINT8</b>	-128 0x80	(-(<MODULE>_MAXSINT8) - 1)	Minimum value of standard data type "sint8"
<b>&lt;MODULE&gt;_MAXUINT16</b>	65535 0xFFFF	(0xffff)	Maximum value of standard data type "uint16"
<b>&lt;MODULE&gt;_MINUINT16</b>	0 0x0000	(0x0)	Minimum value of standard data type "uint16"
<b>&lt;MODULE&gt;_MAXSINT16</b>	32767 0x7FFF	(0x7fff)	Maximum value of standard data type "sint16"
<b>&lt;MODULE&gt;_MINSINT16</b>	-32768 0x8000	(-(<MODULE>_MAXSINT16) -- 1)	Minimum value of standard data type "sint16"
<b>&lt;MODULE&gt;_MAXUINT32</b>	4294967296 0xFFFFFFFF	(0xffffffffuL)	Maximum value of standard data type "uint32"
<b>&lt;MODULE&gt;_MINUINT32</b>	0 0x00000000	(0x0uL)	Minimum value of standard data type "uint32"
<b>&lt;MODULE&gt;_MAXSINT32</b>	2147483647 0x7FFFFFFF	(0x7fffffffL)	Maximum value of standard data type "sint32"
<b>&lt;MODULE&gt;_MINSINT32</b>	-2147483648 0x80000000	(-(<MODULE>_MAXSINT32) -- 1L)	Minimum value of standard data type "sint32"

## 3.4 Rule Set: Naming Convention

This naming convention only applies to SW which resides inside an *ECU*. SW which is used for example for tools running on a PC, is not considered here.

This naming convention is valid for all names, identifiers, symbols, file names of the SW which is written and delivered by CDG and is written in the language "C". This includes also the names used e.g. in assembler files with external visible names, even across the linker.

The naming convention described here, applies also to AUTOSAR-SW. But this naming convention does not apply to names defined by standards. If for example a name of a function is specified by AUTOSAR, then the specified name has to be used.

This naming convention does not apply to SW which is only used internally by CDG.

This rule set of naming convention has a close connection to *"Rule Set: Module Design and Implementation"*. The Naming Convention only defines names of files and elements. The contents of the files are described in *"Rule Set: Module Design and Implementation"*.

The programming language C has several concepts of name identification. The following aspects are specified in the standard C90 and are basics of name identification.

#### ► **Name space of identifiers**

The term name space is defined in C90 in chapter 6.1.2.3. There are four name spaces. The same name can be used independently, if it is defined in different namespaces. The name spaces are:

- Label names:

This means that a label name can be used even if a variable of the same name is defined. The context determines which interpretation is used. This name space is of less relevance for CDG, because 'goto' statements and the goto-labels are banned.

- Tag names:

The tag name is the name which follows immediately the keywords "enum", "struct" or "union". There is only one namespace for tags. This means a "struct x" and a "union x" cannot be used at the same time. But a variable x and the type struct x can be used at same time (and a label x, too).

- Members of structs or unions:

Each struct or union has a separate name space. A member in one struct or union can have the same name as a member in another struct or union without a conflict.

- All other identifiers:

All other identifiers share the same name space. Especially enum members are also part of this name space.

#### ► **Scope**

An identifier is visible (i.e. can be used) only within a region of program text. This is called its scope. There are four scopes defined in C90:

- Function scope:

A label name is the only kind of identifier which has function scope. It can be used in a goto statement anywhere in the function in which it appears. Label names have to be unique within a function. Because goto statements and goto labels are banned, the function scope is not relevant.

Every other identifier's scope is determined by the placement of its declaration.

- File scope:

If the declarator of an identifier appears outside of any block (=compound statement enclosed in { }) or outside any list of function parameters this identifier has file scope.

Note that e.g. an enumeration list enclosed in { } is not a block!

- Block scope:

If the declarator of an identifier appears inside a block or list of function parameters this identifier has block scope and terminates at the } that closes the associated block.

An identifier in block scope hides a lexically identical identifier in file scope or an outer block scope.

#### – Function prototype scope

The identifier for the parameters in a function prototype has function prototype scope. This is similar to block scope with the exception that no block follows the function prototype.

Two identifiers have the same scope if and only if their scopes terminate at the same point.

### ► **Linkage**

There are three kinds of linkage of an identifier:

#### – External linkage:

Any object or function declared at file scope without the storage-class specifier "static".

#### – Internal linkage:

Any object declared at file scope or block scope with the storage-class specifier "static". Any function at file scope declared with the storage-class specifier "static". Note that a function cannot be declared at block scope.

#### – No linkage:

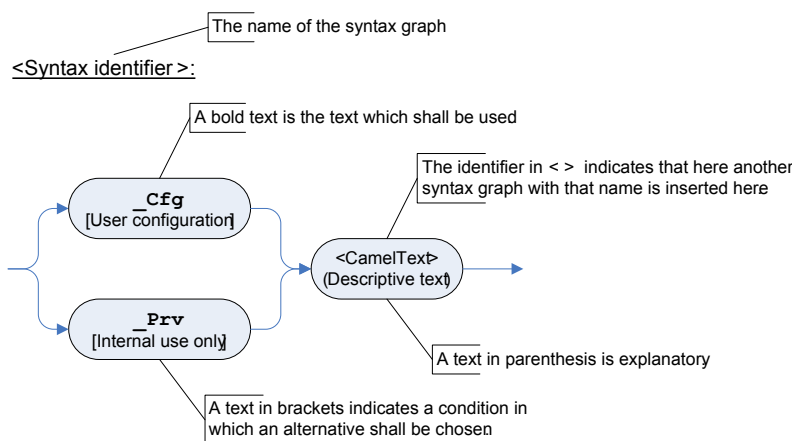
All other identifiers, e.g. automatic variables, typedefs, enum members

### ► **Name space and scope of macro identifier**

There is only one name space for macro identifiers. The scope of a macro identifier starts after its #define preprocessing directive and ends at the end of the translation unit or ends at its #undef preprocessing directive. So name space and scope of macro identifiers uses a different concept compared to the name space and scope of C-identifiers.

This naming convention uses syntax graphs to visualize how a name shall be built. The meaning of the elements of a syntax graph is shown in [Figure 2](#).

Figure 2 Description of Syntax Graphs



Naming part <CamelText> uses camel notation. Camel Notation is a method for concatenating multiple words to one word without spaces. The first character of each word is changed to an uppercase character; all other characters are changed to lowercase letters. Then all these words are concatenated to one word. This rule is also applied to well known abbreviations. E.g. "RAM" and "test" are concatenated to "RamTest".

In opposite to camel notations is underscore notation which is normally used within macro names. Underscore notation is a method for concatenating multiple words to one word without spaces. All words are concatenated together with an underscore in between. This rule is typically used if a case distinction is not possible or not relevant. Often all letters are turned to same case (uppercase or lowercase). E.g. "RAM" and "test" are concatenated to "RAM\_TEST".

Last common point is the definition of visibility. The usability of a SW-component depends on a stable and clear defined interface. This includes clearness about the identifiers which are allowed to be used by the user of a component. So the internally used declarations shall be separated from the declarations which are allowed to be used outside of a component. Two visibilities can be distinguished: "public visibility" and "internal (resp. private) visibility". All public visible interfaces and objects are located in the component header file (or in sub headers which are included in the component header). All other interfaces and objects which shall not be exported and have only an internal visibility shall be defined within the private component header (or in sub headers which are included in the private component header). It shall be noticed that a declaration can be public visible even if it was private. Such a declaration shall contain a "\_Prv" / "\_PRV" in the name, see [CDGNaming\_002] and [CDGNaming\_003].

## Rule CDGNaming\_001: Definition of a Component Prefix

**Instruction** Every name with a global visibility shall be prefixed with a component prefix.

One major goal of this naming convention is to avoid naming conflicts (internally and with SW used by customers of CDG). This will be achieved by assigning a prefix to every identifier which can result in a naming conflict. All SW which is delivered to a customer of CDG is clustered in components. So every piece of code is part of a component and shall have an unique name. Every name with global visibility has to be prefixed with this "component prefix".

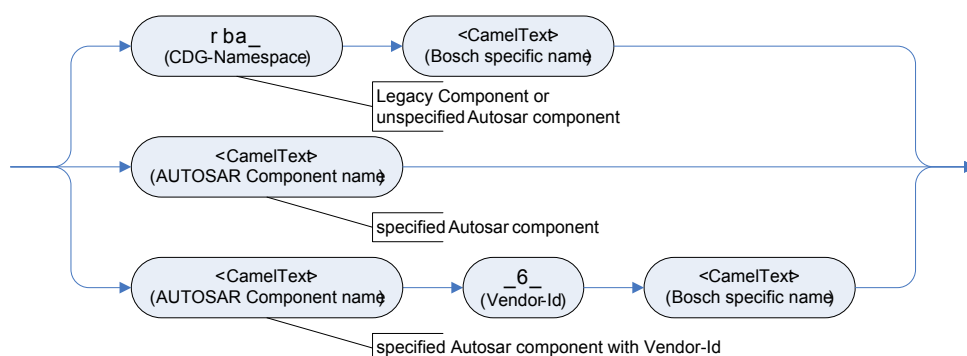
Building the component prefix follows one of the following criteria:

- CDG has an own name space represented by a prefix. The prefix for CDG is "rba\_" and "RBA\_". This prefix is used for all components which are not defined in a standard. An additional name follows this prefix. Both naming parts build the component prefix in this case. A component prefix has to be cleared by the "CDG BSW Component Name Clearing Board".
- A component defined in a standard (e.g. if an AUTOSAR specification is available and used) shall use the component name as defined. Such a component shall not be prefixed with "rba\_" / "RBA\_" (e.g. additional functions for Dem will be prefixed with Dem\_). In this case the component prefix is only built with the component name defined in AUTOSAR.
- For some cases AUTOSAR allows multiple implementations of drivers. This rule is related to BSW00347 in the AUTOSAR specification. An example is "Can". There are external CAN-devices. The drivers for such external devices can be implemented independently from the uC-internal Can-driver. So two or more Can-Driver will be linked together. To avoid a naming conflict, the drivers for the external CAN-device get an additional infix consisting of a vendor-Id and an additional name. In case of Bosch the vendor-Id is 6. The additional name has also to be cleared by the "CDG BSW Component Name Clearing Board". The name of the uC-internal driver consists only of the AUTOSAR component name.

Following [Figure 3](#) shows the rule to create a component prefix in graphical form.

Figure 3 Creation of a Component Prefix

<Component Prefix>:



Component prefix for macro identifiers is built with same criteria shown above, but all letters shall be uppercase letters. Additionally macro identifiers cannot be hidden by a new declaration, even if this declaration is in block scope. Also the



usage of macros is complex and error prone. A separate name space for macro identifiers is defined to make this obvious. A most common name space for macro identifiers is using only uppercase letters, digits and underscores. Outside the naming conventions *<Module>* or *<MODULE>* is used instead of *<Component Prefix>* or *<COMPONENT PREFIX>*.

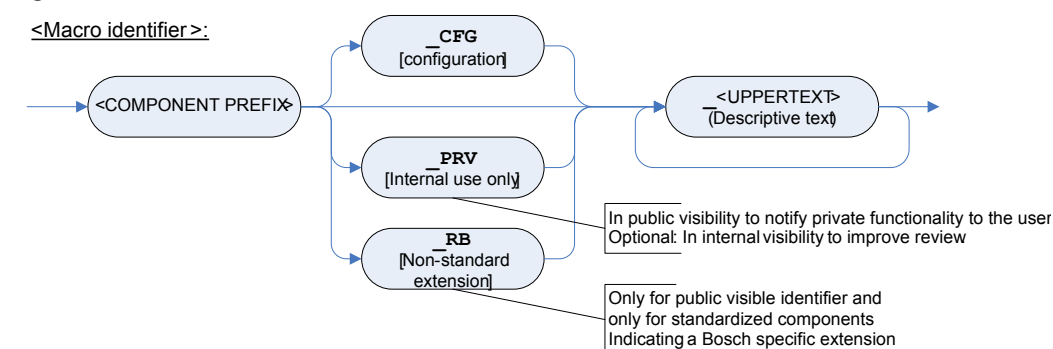
## Rule CDGNaming\_002: Macros

**Instruction** Macro names shall be defined in following form: *<COMPONENT PREFIX>{\_IDENTIFIER}\_UPPERTEXT>*.

In macro names underscore notation is used for naming part *<UPPERTEXT>*.

In *Figure 4* a graphical overview is shown.

Figure 4 Creation of Macro Identifier



Examples:

- ▶ The macro which denotes a start address and is declared in a header with internal visibility of component *rba\_PdmFs*: *RBA\_PDMFS\_STARTADDRESS*
- ▶ The macro, which denotes the number of blocks and is declared in the public header of component *rba\_PdmFs*, but shall be not used by the user of the component, shall have the name: *RBA\_PDMFS\_PRV\_BLOCK\_NUM*

## Rule CDGNaming\_012: SymbolicNameValue

**Instruction** The values of configuration parameters which are defined as *symbolicNameValue* = true shall be generated into the header file of the declaring module as *#define*. The symbol shall be composed of

- ▶ the component prefix of the declaring component followed directly by the literal "Conf\_" followed by
- ▶ the *shortName* of the *EcucParamConfContainerDef* of the declaring module followed by "\_" followed by
- ▶ the *shortName* of the *EcucContainerValue* container which holds the *symbolicNameValue* configuration parameter value.

Taking the specification requirements above the configuration snippet results in the according symbolic name definition in the header file of the providing Dem module:

```
#define DemConf_DemEventParameter_CORTST_E_CORE_FAILURE_1 17
```

This rule refers to [ecuc\_sws\_2108] in AUTOSAR\_TPS\_ECUConfiguration of version 4.0.3. This rule shall be followed even for 4.0.2 to avoid name clashes. Also this rule overrule all other rules in case of conflicts.

## Rule CDGNaming\_003: C-Identifiers with File Scope

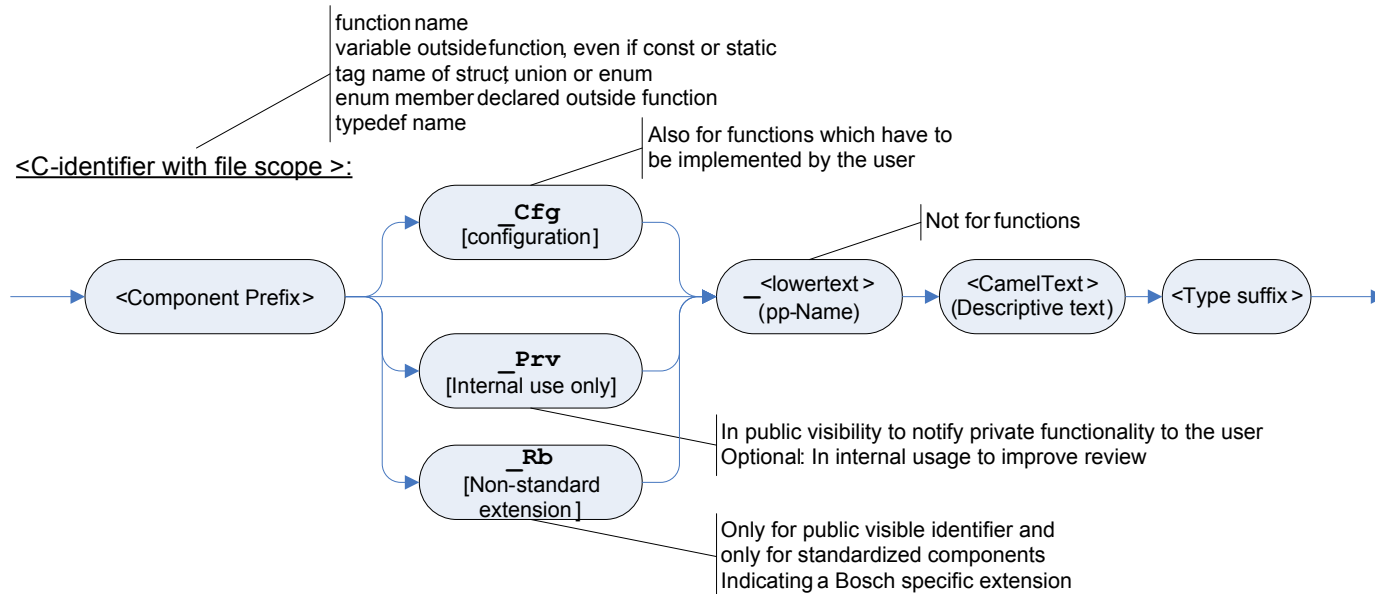
**Instruction** C-identifiers with file scope shall be defined in following form: *<Component prefix>{\_identifier}\_pp<-DescriptiveText>{\_Typesuffix}*.

This rule is relevant for following identifiers:

- Function names
- Variables outside of functions, even if it is const or static
- Tag name of a struct, union or enum
- Enum member, declared outside of a function
- Name of a typedef

In [Figure 5](#) a graphical overview is shown.

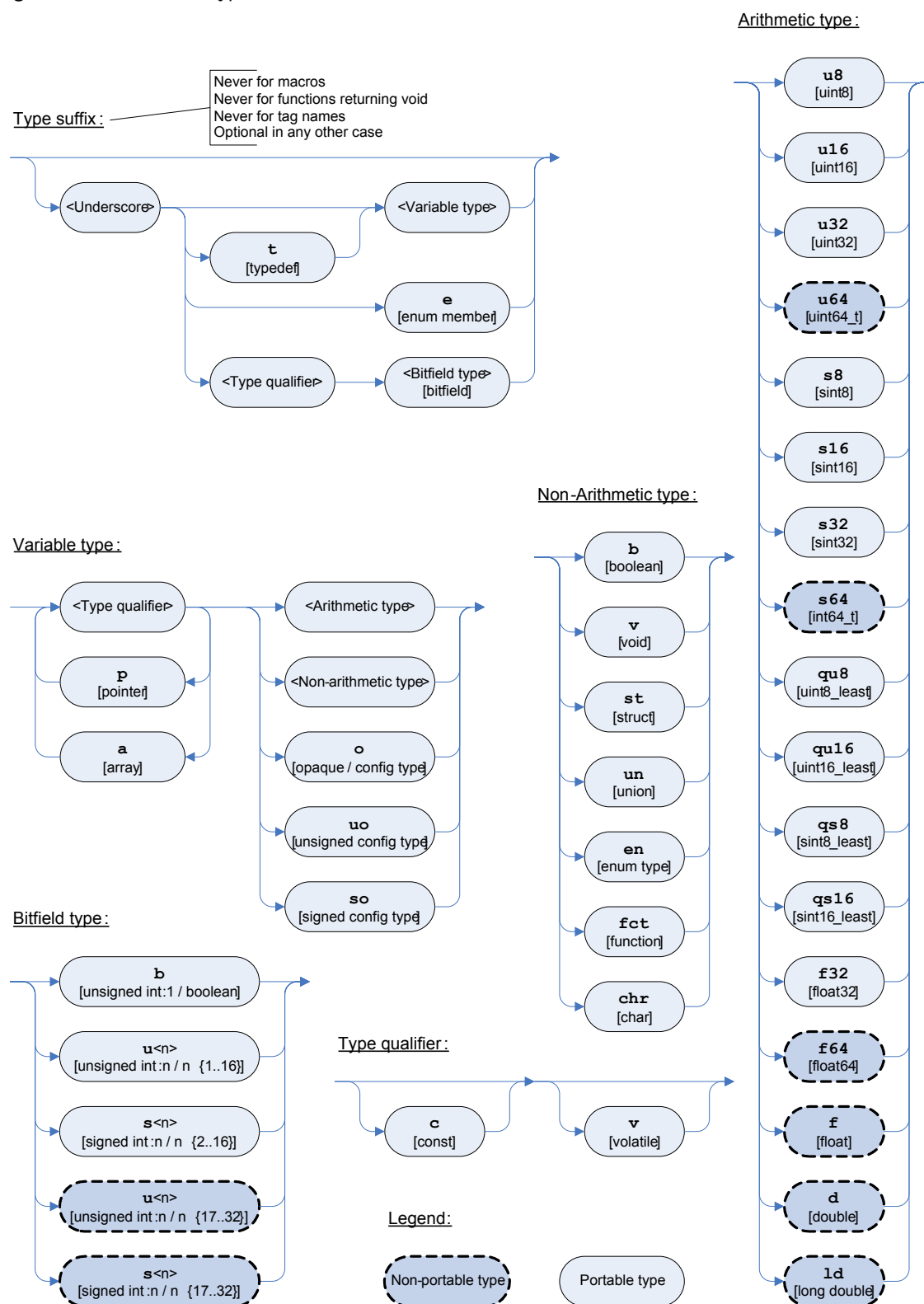
Figure 5 Creation of C-identifiers with File Scope



Instead of using camel notation for enum members is it also allowed to use upper case notation.

In [Figure 6](#) a graphical overview to create a type suffix is shown. Physical and logical types for (pp-name) can be found in appendix in [Chapter "Physical and Logical Types"](#).

Figure 6 Creation of Type Suffixes



**Hint** It shall be noticed that some datatypes (float32, float64, float, double, long double, uint64\_t, int64\_t) are not portable. In those rare cases where these datatypes are being used, a deviation shall be documented, and the specified type suffix is still recommended

The correct choice of type suffix depends on the change scenario. This will be illustrated at the type PduLengthType from ComStack\_Types.h. This type is defined in AUTOSAR, but this can be either uint8, uint16 or uint32. The users of this

type do not know which of them is assigned. So a variable of this type shall have the type suffix "\_uo". The type suffix "\_uo" indicates, that there is no reliance on the size of the type because this type is opaque. So the user of this type shall implement his code in a way, that it is correct in all possible implementations of PduLengthType. It is a good practice to document the proposed type suffix in the user documentation of the component which provides such a type.

Examples:

- ▶ The initialization function, which returns void, shall have the name without type suffix: void rba\_PdmFs\_Init(void);
- ▶ The initialization function of the component Can according AUTOSAR has the name: void Can\_Init(void);
- ▶ A variable containing the status could have the name: uint8 rba\_PdmFs\_Status\_u8;  
In "Type suffix:" is chosen "<underscore>" and "<Variable type>" since these is a variable.  
In "Variable type:" is chosen "<Type qualifier>" and "<Arithmetic type>".  
In "Type qualifier:" is chosen nothing since the type is neither const nor volatile.  
In "Arithmetic type: " is chosen "u8" since the type is uint8.
- ▶ An array with internal usage could have the following name: uint8 rba\_PdmFs\_Priv\_dataMain\_au8[10];  
In "Type suffix:" is chosen "<underscore>" and "<Variable type>" since these is a variable.  
In "Variable type:" is chosen "<Type qualifier>" and "a" and again "<Type qualifier>" and then "<Arithmetic type>".  
In both "Type qualifier:" is chosen nothing since neither the array nor the element is const or volatile.  
In "Arithmetic type: " is chosen "u8" since the type is uint8.
- ▶ A pointer to a function, which returns a sint8, while the pointer has file scope, could have the following name: sint8 (\*rba\_PdmFs\_adrCallBack\_pfct)(uint32 dataLength\_u32);  
In "Type suffix:" is chosen "<underscore>" and "<Variable type>" since these is a variable (a pointer).  
In "Variable type:" is chosen "<Type qualifier>" and "p" and again "<Type qualifier>" and then "<Non-arithmetic type>".  
In first "Type qualifier:" is chosen neither const nor volatile since the pointer is none of this  
In second "Type qualifier:" is chosen nothing since neither const nor volatile is useful for functions.  
In "Non-arithmetic type: " is chosen "fct" since the derefernced type is function. The concrete type of function shall not be considered. So the return type is not part of the resulting suffix "\_pfct".
- ▶ An array of pointer to const structs could have the following declaration: const rba\_PdmFs\_MyData\_tst \*rba\_PdmFs\_dataInfo\_apcst[10];  
For the type name is in "Type suffix:" chosen "<underscore>", "t" and "<Variable type>", since this is a type.  
For the type name is in "Variable type" chosen "<Type qualifier>" and "<Non-arithmetic type>".  
For the type name is in "Type qualifier:" chosen nothing since in the typedef is neither volatile nor const used.  
For the type name is in "Non-arithmetic type:" chosen "st" since the type is a structure. Different structures are not distinguished in the type suffix.  
For the type name we will get "\_tst" as suffix.  
For the variable name is in "Type suffix:" chosen "<underscore>" and "<Variable type>", since this is a variable.  
For the variable name is in "Variable type" chosen "<Type qualifier>", "a", "<Type qualifier>", "a", "<Type qualifier>", "a", and "<Non-arithmetic type>".  
For the variable name is in first "Type qualifier:" chosen nothing since the array is neither const nor volatile.  
For the variable name is in second "Type qualifier:" chosen nothing since the array element, which is a pointer, is neither const nor volatile.  
For the variable name is in third "Type qualifier:" chosen "c" since where the pointer points to is const.

For the variable name is in "Non-arithmetic type:" chosen "st" since the type, where the pointer points to, is a structure. Different structures are not distinguished in the type suffix.

For the variable name we will get "\_apcst" as suffix.

## Rule CDGNaming\_004: Typedefs

**Instruction** A typedef name shall be a unique identifier.

No typedef name shall be reused either as a typedef name or for any other purpose. Typedef names shall not be reused anywhere within a program. The same typedef shall not be duplicated anywhere in the source code files even if the declarations are identical. Where the type definition is made in a header file, and that header file is included in multiple source files, this rule is not violated.

## Rule CDGNaming\_005: Tag Names

**Instruction** A tag name shall be a unique identifier.

No tag name shall be reused either to define a different tag or for any other purpose within the program. ISO 9899:1990 [2] does not define the behaviour when an aggregate declaration uses a tag in different forms of type specifier (struct or union). Either all uses of the tag should be in structure type specifiers, or all uses should be in union type specifiers. The same tag definition shall not be duplicated anywhere in the source code files even if the declarations are identical. Where the tag definition is made in a header file, and that header file is included in multiple source files, this rule is not violated.

Example:

```
struct MyModule_xTag_tst
{
    uint16 stHugo_ul6;
    uint16 stAnna_ul6;
};

struct MyModule_xTag_tst a1_st = { 0, 0 }; /* OK, compatible with above */
union MyModule_xTag_tst  a2_st = { 0, 0 }; /* Not OK, not compatible with */
                                         /* previous declarations */
```

## Rule CDGNaming\_006: C-Identifiers with Block Scope

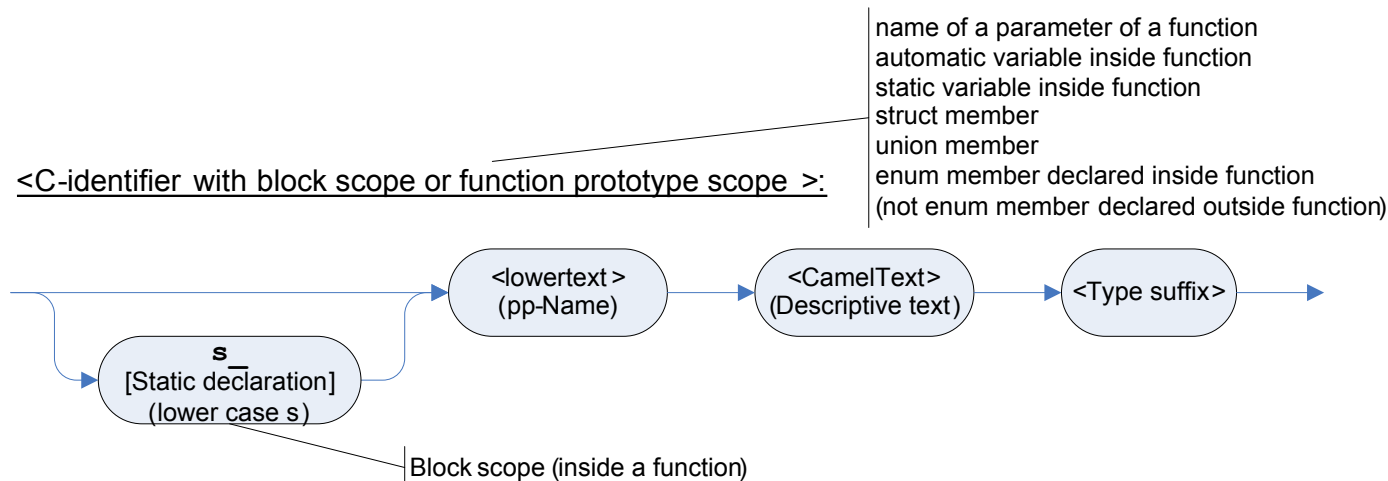
**Instruction** C-identifiers with block scope or function prototype scope shall be defined in following form: {s\_<pp><DescriptiveText>\_<Typesuffix>.

This rule is relevant for following identifiers:

- ▶ Names of parameters of a function
- ▶ Automatic variables inside a function
- ▶ Static variables inside a function
- ▶ Member of a struct
- ▶ Member of an union
- ▶ Member of an enum declared inside a function (For enum members outside a function [\[CDGNaming\\_006\]](#) is relevant.)

In [Figure 7](#) a graphical overview is shown.

Figure 7 Creation of C-identifiers with Block Scope or Prototype Scope



Examples:

- ▶ A pointer to a const data structure of unknown type, declared as parameter, shall have the following name if you decide to use the type suffix: `const void * dataSrc_pcv;`
- ▶ A const pointer to a volatile data structure of unknown type, declared as parameter, shall have the following name if you decide to use the type suffix: `volatile void * const dataDest_cpvv;`

### Rule CDGNaming\_007: Usage of Object or Function Identifier

**Instruction** No object or function identifier with static storage duration should be reused.

Regardless of scope, no identifier with static storage duration should be re-used across any source files in the system. This includes objects or functions with external linkage and any objects or functions with the static storage class specifier. While the compiler can understand this and is in no way confused, the possibility exists for the user to incorrectly associate unrelated variables with the same name. One example of this confusion is having an identifier name with internal linkage in one file and the same identifier name with external linkage in another file.

An allowed exception of this rule is the usage of the same name of block scope objects with the specifier "static". The usage of such identifiers in different functions is allowed.

### Rule CDGNaming\_008: Spelling of Identifiers in Different Name Spaces

**Instruction** No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.

Name space and scope are different. This rule is not concerned with scope. For example, ISO C allows the same identifier for both a tag and a typedef at the same scope. Please see above for an explanation of name space and scope.

This rule also allows the use of the same name for a local variable inside of a function and for a struct or union member. This is an allowed derivation of MISRA rule 5.6.

### Rule CDGNaming\_009: Identifiers of Inner and Outer Scope

**Instruction** Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.

The naming convention ensures this rule for two identifiers at file scope and for two identifiers where one is file scope and the other is block scope. This rule is added to avoid the same name in an inner and an outer block scope.

## Rule CDGNaming\_010: Maximum Length of Identifiers

**Instruction** Identifiers (internal and external) shall not rely on the significance of more than 60 characters.

C-code has to be portable to different compilers. Originally 31 characters was intended within ISO C90. Because of requirements from AUTOSAR the number of significant characters shall be higher, although currently no compilers are known which do not support more than 128 characters or more significance. But to minimize the risk for porting to foreign compilers the 60 characters are now the limit for names in software developed from CDG. Moreover readability and handling of names is considered.

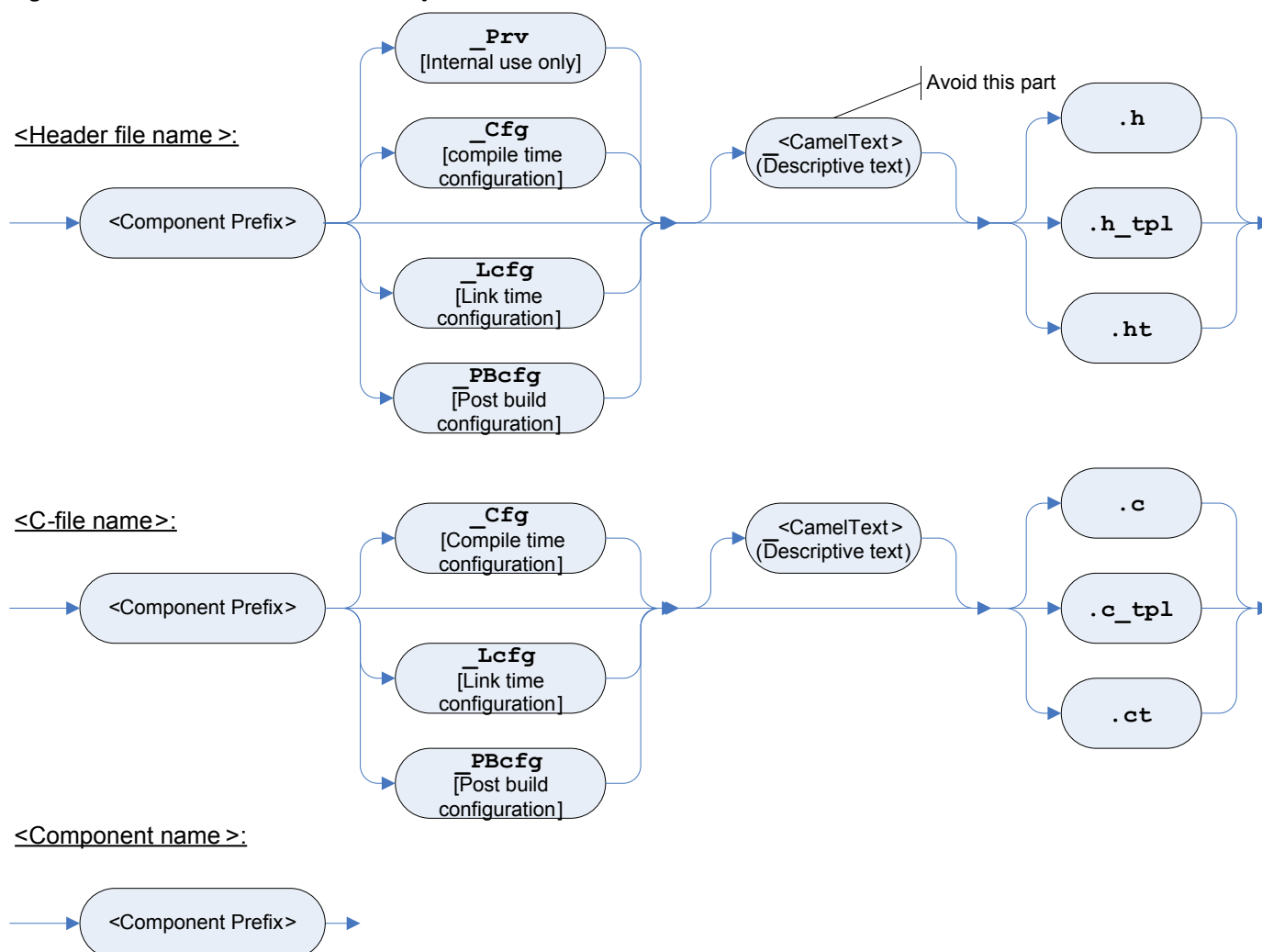
This rule shall apply across all name spaces.

## Rule CDGNaming\_011: File and Directory Names

**Instruction** File and directory names shall be defined in a specified form.

In [Figure 8](#) a graphical overview is shown.

Figure 8 Creation of File and Directory Names



The usage of an identifier requires the inclusion of a header. The name of the header shall be specified. This name should not change even if the version of a module changes. The simplest way to implement this requirement is to use one module header which is the public visible header in a component. Non public objects are located in private header files (containing "\_Prv" in the name). More details about contents of c files and headers and about header include concept can be found in [Chapter "Basis Set of Module Files"](#) and [Chapter "Header Include Concept"](#).

Example:

- The public header file of a non-standard component "PdmFs" shall have the name: rba\_PdmFs.h.

## 3.5 Rule Set: Module Design and Implementation

This rule set describes common requirements for module design and implementation. All valid module files are defined, an include strategy for module header files is given, demands for interfaces (APIs and processes) are mentioned and other implementation rules are listed. Regarding file and header names this rule set has a close dependency to *"Rule Set: Naming Convention"*.

### 3.5.1 Basis Set of Module Files

The BSW software is decomposed into Not every module needs all files and headers, single SW modules. A module contains a set of files which vary in number and type of files. The following rules introduce all types of files and their tasks, relevance and incidences.

#### Rule BSW\_Files\_001: Basic Set Of Module Files

**Instruction** All modules shall provide at least the following files: "<Module>.c", "<Module>.h" and "<Module>\_BSWMD-arxml". Other files have to be provided if they are needed.

This rule lists all possible c files and headers of a module. Not every module needs all files and headers, decisive is the demand of a corresponding context. The ARXML files and configuration processor files are discussed in other rule sets.

In *Table 22* an overview of all possible files is given.

Table 22 Overview of Basic Set of Module C Files and Headers

File Name	Incidence	Description
<Module>.c *	1	Module source file(s)
<Module>_<Sub>.c	0...n	Additional sub module source files. <Sub> can be chosen with individual names. Sub module source files can be used to structure a module but need not be used. Minimum requirement is that a <Module>.c file exists.
<Module>.h *	1	Module header file containing public information of the module. This header is the export header to be included in other modules.
<Module>_<Sub>.h	0...n	Additional and optional module header files to structure the public header <Module>.h of a module. They are included in <Module>.h and exported to other modules. <Sub> can be chosen with individual names. Such headers can be used but they have not to be used.
<Module>_<Sub>_Inl.h	0...n	Additional and optional module header to separate inline functions from central module header <Module>.h for a better structure of the module. Is included in <Module>.h for export to other modules. <Sub> can be chosen with individual names. Such headers can be used but they have not to be used, public inline functions can also be placed inside <Module>.h too.
<Module>_CbK.h *	0...1	Module callback header file, if callbacks are provided to other modules
<Module>_<User>.h *	0...n	Additional module header with interfaces which are provided exclusively for one other module, <User> is the name of the other module (This header will be included in another module as <ExtModule>_<Module>.h)
<Module>_Cfg.c *	0...1	(Mostly script generated) module configuration file for pre-compiled configuration. This file is optional and can be used if pre-compile configuration is made.



File Name	Incidence	Description
<b>&lt;Module&gt;_Cfg_&lt;Sub&gt;.c</b>	0...n	(Mostly script generated) additional and optional sub module source files for pre-compiled configuration. <Sub> can be chosen with individual names. Sub module source files can be used to structure a module but need not be used. If a module have only one pre-compiled configuration source file <Module>_Cfg.c shall be used.
<b>&lt;Module&gt;_Cfg.h *</b>	0...1	(Mostly script generated) module configuration header file for pre-compile configuration. This file is optional and can be used if pre-compile configuration is made.
<b>&lt;Module&gt;_Cfg_&lt;Sub&gt;.h</b>	0...n	(Mostly script generated) additional and optional module header file to structure pre-compiled configuration headers. <Sub> can be chosen with individual names. Such headers can be used but they have not to be used.
<b>&lt;Module&gt;_Lcfg.c *</b>	0...1	(Mostly script generated) module configuration file for link time configuration. This file is optional and can be used if link time configuration is made.
<b>&lt;Module&gt;_Lcfg_&lt;Sub&gt;.c</b>	0...n	(Mostly script generated) additional and optional sub module source files for link time configuration. <Sub> can be chosen with individual names. Sub module source files can be used to structure a module but need not be used. If a module have only one link time configuration source file <Module>_Lcfg.c shall be used.
<b>&lt;Module&gt;_Lcfg.h *</b>	0...1	(Mostly script generated) module configuration header file for link time configuration. This file is optional and can be used if link time configuration is made.
<b>&lt;Module&gt;_Lcfg_&lt;Sub&gt;.h</b>		(Mostly script generated) additional and optional module header file to structure link time configuration headers. <Sub> can be chosen with individual names. Such headers can be used but they have not to be used.
<b>&lt;Module&gt;_PBcfg.c *</b>	0...1	(Mostly script generated) module configuration file for post build time configuration. This file is optional and can be used if post build configuration is made.
<b>&lt;Module&gt;_PBcfg_&lt;Sub&gt;.c</b>	0...n	(Mostly script generated) additional and optional sub module source files for post build time configuration. <Sub> can be chosen with individual names. Sub module source files can be used to structure a module but need not be used. If a module have only one post build time configuration source file <Module>_PBcfg.c shall be used.
<b>&lt;Module&gt;_PBcfg.h *</b>	0...1	(Mostly script generated) module configuration header file for post build time configuration. This file is optional and can be used if post build configuration is made.
<b>&lt;Module&gt;_PBcfg_&lt;Sub&gt;.h</b>	0...n	(Mostly script generated) additional and optional module header file to structure post build time configuration headers. <Sub> can be chosen with individual names. Such headers can be used but they have not to be used.
<b>&lt;Module&gt;_Types.h *</b>	0...1	Module specific types and symbols
<b>&lt;Module&gt;_Prv.h</b>	0...1	Additional private header which is not exported via <Module>.h, only for internal purpose. This header is optional and has to be included in the module c files ( <Module>.c, <Module>_<Sub>.c, <Module>_Cfg.c, <Module>_Cfg_<Sub>.c, <Module>_Lcfg.c, <Module>_Lcfg_<Sub>.c, <Module>_PBcfg.c, <Module>_PBcfg_<Sub>.c).
<b>&lt;Module&gt;_Prv_&lt;Sub&gt;.h</b>	0...n	Additional private module header to separate information from <Module>_Prv.h for a better structure of the module. Is included in <Module>_Prv.h and contains only module internal elements. Sub module header files can be used to structure a module but need not be used, they are optional. <Sub> can be chosen with individual names.

File Name	Incidence	Description
<b>&lt;Module&gt;_Prv_&lt;Sub&gt;_Inl.h</b>	0...n	Additional private module header to separate inline functions from module header <Module>_Prv.h for a better structure of the module. Is included in <Module>_Prv.h and contains only module internal elements.  Sub module inline header files can be used but need not be used, they are optional. Private inline functions can also be placed inside <Module>_Prv.h header too. <Sub> can be chosen with individual names.

**Hint** Files which are optional (depending on implementation / configuration) have an incidence starting with 0.

**Hint** Files marked with a " \* " are defined by AUTOSAR. Other files are possible add ons. All other files than "<Module>.c" and "<Module>.h" are optional and should only be used if their content is needed.

In [Table 23](#) an overview of ARXML files is given. For a BSW module the BSW software module description file (BSWMD) is mandatory. A software component description file (SWCD) is only used in exceptional cases within a BSW module. A detailed description is done in ["Rule Set: Data Description"](#). If the ARXML files are generated by script the file name is expanded with a qualifier(e.g. <Module>\_Cfg\_BSWMD.arxml). For more details take a look to Rule [\[ECUC\\_P013\]](#).

Table 23 Overview of ARXML Files

File Name	Incidence	Description
<b>&lt;Module&gt;_BSWMD.arxml</b>	1	BSW Software Module Description
<b>&lt;Module&gt;_SWCD.arxml</b>	0...1	Software Component Description
<b>&lt;Module&gt;_EcucParamDef.arxml</b>	0...1	ECU Configuration Parameter Definition
<b>&lt;Module&gt;_{&lt;Sub&gt;}EcucValues.arxml</b>	0...n	ECU Configuration Values
<b>&lt;Module&gt;_Prot_EcucValues.arxml</b>	0...1	protected ECU Configuration Values

If a module is configurable a script based processor has to be provided by default to handle the configuration use case. [Table 24](#) shows an overview over all files of that use case. Details are described in ["Rule Set: Data Description"](#).

E.g. for some legacy SW such a handling of configuration data is not used, but only template files are provided to configure a module without a script based processor. [Table 25](#) shows the possible files for that use case. The template files are blueprints for configuration c and h files which has to be created from a user of a module.

Table 24 Overview Script Processor Based Files for Configuration

File Name	Incidence	Description
<b>&lt;Module&gt;_{&lt;Sub&gt;}.mwe</b>	0...n	oAW Model Workflow Engine (replaces the .oaw file)
<b>&lt;Module&gt;_{&lt;Sub&gt;}.chk</b>	0...n	oAW Check Script
<b>&lt;Module&gt;_{&lt;Sub&gt;}.ext</b>	0...n	oAW Xtend Script
<b>&lt;Module&gt;_{&lt;Sub&gt;}.xpt</b>	0...n	oAW Xpand Script
<b>&lt;Module&gt;_{&lt;Sub&gt;}.pm</b>	0...n	Perl module
<b>&lt;Module&gt;_{&lt;Sub&gt;}.ct</b>	0...n	c template for configuration processor
<b>&lt;Module&gt;_{&lt;Sub&gt;}.ht</b>	0...n	h template for configuration processor
<b>&lt;Module&gt;_{&lt;Sub&gt;}.xt</b>	0...n	ARXML template for configuration processor
<b>&lt;Module&gt;_{&lt;Sub&gt;}.rt</b>	0...n	report template for configuration processor
<b>&lt;Module&gt;_{&lt;Sub&gt;}.hxt</b>	0...n	hex template for configuration processor
<b>&lt;Module&gt;.bamf</b>	0...1	Build Action Manifest (mandatory if script is called from BCT)

Table 25 Overview Non Script Processor Based Template Files for Configuration

File name	Incidence	Description
<b>&lt;Module&gt;{&lt;Sub&gt;}.c_tpl</b>	0...1	C template file for non script based configuration
<b>&lt;Module&gt;{&lt;Sub&gt;}.h_tpl</b>	0...1	Header template file for non script based configuration

In [Table 26](#) all other specific files are listed with their file extensions.

Table 26 Overview Other Specific Files

File Name	Incidence	Description
<b>&lt;Module&gt;{&lt;Sub&gt;}.mcs</b>	0...n	MCS is a programmable sub module of GTM (Generic Timer Module). The .mcs files contains an assembler like programming language. This file is GTM specific.
<b>&lt;Module&gt;{&lt;Sub&gt;}.inc</b>	0...n	Include header for a .mcs file. This file is Infineon Tricore specific.
<b>{&lt;Module&gt;}{&lt;Sub&gt;}.exe</b>	0...n	Executables (e.g. specific compilers). Name of the file can be independent from the name of the module.

{<Sub>} is optional and can be chosen with an individual name. Different names are needed if one file type exists more than once.

In which folders of the SCM system the module files are located is described in the SCM plan.

## Rule BSW\_Files\_002: Additional Module C Files

**Instruction** If a module provides several functions and processes additional module source files "<Module>\_<Sub>.c" may be used. Name <Sub> can be chosen freely.

To have sub module c files is not mandatory but they can help to structure a module. Furthermore used memory resources can be reduced if functions are split to several files. This allows the linker to omit the objects of unused functions and processes. A reason for this feature is that a linker locates only complete objects derived from a file and such an object cannot be subdivided. If more sub files exists linker can work more efficient. It is a good practice to use sub files containing groups of functions and processes which belongs together.

## Rule BSW\_Files\_003: Header for Callback Functions

**Instruction** Declarations of callback functions shall be grouped and out-sourced in a separate header file "<Module>\_Cbkh.h".

Separate and decouple callback declarations from explicitly exported functions. Limit access and prevent misuse of unintentionally exposed API. Promote better maintainability of callback declarations, implementations and configurations. This header is only necessary for declarations of functions, which shall be called from a lower AUTOSAR layer. This technique helps to avoid the recursive inclusion problem.

A module which invokes callback functions from another module has to include only the callback header file from the other module <Extmodule>\_Cbkh.h. The callback functions are not exported via common module header <Module>.h to minimize the effort on provider and user side of callback functions.

Example:

Callback functions of NVRAM-Manager in header "NvM\_Cbkh.h":

```
...
extern void NvM_NotifyJobOk(void);
extern void NvM_NotifyJobError(void);
...
```

This header is optional and only needed if callback functions are available and needed. Incidence of this header is 0...1.

## Rule BSW\_Files\_004: Files for ECU Configuration

**Instruction** Configuration parts shall strictly be separated from implementation of functionality. Configuration data (not to be modified after compile time) shall be grouped and out-sourced to following configuration files: "<Module>\_Cfg.c", "<Module>\_Cfg\_<Sub>.c", "<Module>\_Cfg.h" and "<Module>\_Cfg\_<Sub>.h" for pre-compile configuration data, "<Module>\_Lcfg.c", "<Module>\_Lcfg\_<Sub>.c", "<Module>\_Lcfg.h" and "<Module>\_Lcfg\_<Sub>.h" for link time configuration data and "<Module>\_PBcfg.c", "<Module>\_PBcfg\_<Sub>.c", "<Module>\_PBcfg.h" and "<Module>\_PBcfg\_<Sub>.h" for post build configuration data.

Static configuration data has to be decoupled from implementation. Separation of configuration dependent data at compile time furthermore enhances flexibility, readability and reduces version management as no source code is affected.

Example:

```
In Tp_Cfg.h:
#define TP_CFG_USE_NORMAL_ADDRESSING KTPOFF
#define TP_CFG_USE_NORMAL_FIXED_ADDRESSING KTPOFF
#define TP_CFG_USE_EXTENDED_ADDRESSING KTPON
...

in Tp.c:
#include "Tp_Cfg.h"
...
#if (TP_CFG_USE_NORMAL_ADDRESSING == KTPOFF)
... do something
#endif
```

These files are optional and only needed if configuration context is needed. Incidence of these files is 0...1. Also all files which are generated by a configuration tool shall have such names and shall comply with [\[CDGNaming\\_011\]](#).

## Rule BSW\_Files\_005: Header for Module Specific Types

**Instruction** Module specific types and symbols can be defined in header "<Module>\_Types.h".

This header is optional and only needed if module specific types and symbols are needed and if they cannot be located in file "<Module>.h". Incidence of this header is 0...1.

## Rule BSW\_Files\_006: Header for Exclusive Interfaces for another Module

**Instruction** Interfaces which are provided exclusively for one module should be separated into a dedicated header file "<Module>\_<User>.h".

<User> is the name of the other module which is the user of the contents of this header. This header will be included in another module as "<ExtModule>\_<Module>.h".

This header is optional and only needed if interfaces are exclusively provided to another module. Incidence of this header is 0...1.

## Rule BSW\_Files\_007: Additional Module Headers

**Instruction** To structure headers of a module additional sub header(s) can be used: "<Module>\_<Sub>.h" and "<Module>\_<Sub>\_Inl.h" to structure "<Module>.h" and "<Module>\_Prv\_<Sub>.h" and "<Module>\_Prv\_<Sub>\_Inl.h" to structure "<Module>\_Prv.h". Name <Sub> can be chosen freely.

This rule helps to keep the central module header <Module>.h as small as possible. To get an overview and to structure header information these additional headers can be used. Some specifications of AUTOSAR modules intend such headers.

These sub headers are optional and only needed if header information is structured. Incidence of these headers are 0...n. Independently from both headers inline functions can be defined in `<Module>.h` header, too.

### Rule BSW\_Files\_008: Private Module Header

**Instruction** Elements which shall not be exported via module header file "`<Module>.h`" shall be placed in private header "`<Module>_Prv.h`".

All elements with an internal focus which shall not be exported to other modules shall be defined in the private header of the module. Incidence of this header is 0...1.

## 3.5.2 Header Include Concept

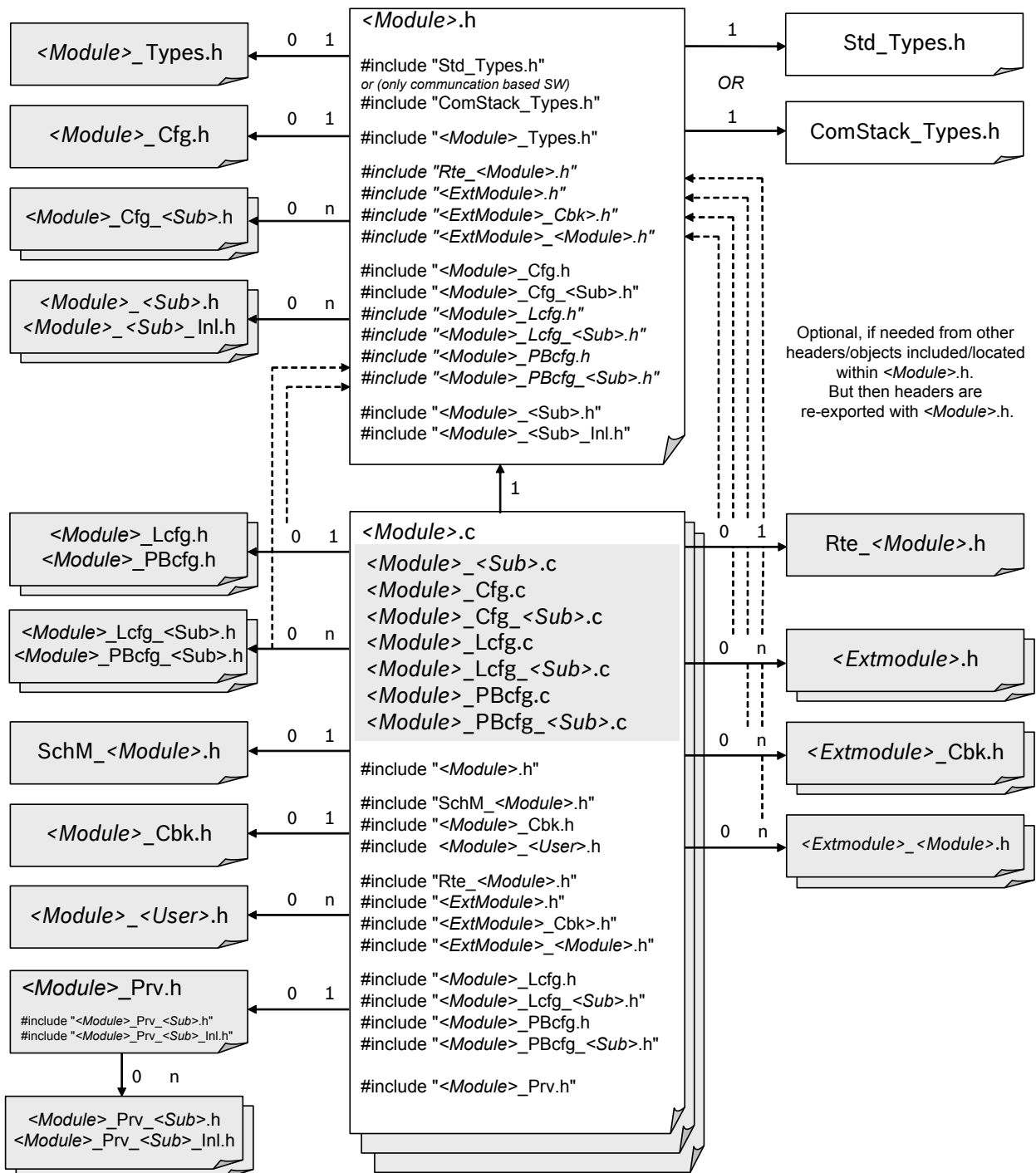
A module can contain a various set of header files. In addition headers from outside of the module are needed and have to be included. To provide a uniform visibility of header information a header include concept is needed. Following rules describe which headers have to be included in which files and in which order.

### Rule BSW\_HeaderInc\_001: Definition of BSW Header Include Concept

**Instruction** Following header include concept has to be followed.

The header include concept is shown in *Figure 9*. The following sub rules gives additional details.

Figure 9 Overview Header Include Concept



**Hint** Files which are optional (depending on implementation / configuration) are shown in grey.

**Hint** Possibly some AUTOSAR module specifications uses different include strategies to those shown here. In that case AUTOSAR module specification has a preference. Or the specification can be modified to be conform to common include strategy.

## Rule BSW\_HeaderInc\_002: Include Order of Module Header

**Instruction** "<Module>.h" header has to be included as first header in every c file (functional and configuration) of a module.

"<Module>.h" is the central header of a module.

## Rule BSW\_HeaderInc\_003: Include Order after Module Header

**Instruction** Further preferred include order of headers in module c files: BSW scheduler header ("SchM\_<Module>.h"), module callback header ("<Module>\_Cbkg.h"), friends header ("<Module>\_<User>.h"), RTE generated header ("Rte\_<Module>.h"), external module headers ("<Extmodule>.h", "<Extmodule>\_Cbkg.h", "<Extmodule>\_<Module>.h"), configuration header files ("<Module>\_Lcfg.h", "<Module>\_Lcfg\_<Sub>.h", "<Module>\_PBcfg.h", "<Module>\_PBcfg\_<Sub>.h") private header file ("<Module>\_Prv.h").

Only that headers shall be included which are needed from the corresponding module c file. The include order can be changed if the visibility and dependency of contents of headers needs another include order than the preferred one.

"SchM\_<Module>.h" is the header with module specific functionalities provided by the BSW scheduler.

The module callback header "<Module>\_Cbkg.h" has to be included in that c file where the callback function(s) is/are implemented.

The friends header "<Module>\_<User>.h" has to be included in that c file where the function(s) is/are implemented.

The include of the RTE generated header "Rte\_<Module>.h" is only needed for those modules where such a header is generated. Few BSW modules provides a Software Component Description file <Module>\_BSWMD.arxml which is the base for the generation of that header.

Headers from external modules ("<Extmodule>.h", "<Extmodule>\_Cbkg.h", "<Extmodule>\_<Module>.h") could be included in those module c files where the information of those headers is needed (More details can be found in [\[BSW\\_HeaderInc\\_007\]](#)). Sometimes a module needs that information in a more global form, even than if APIs or inline functions of the module are dependent from external modules. In those cases the include of external module headers have to be made in the module header file <Module>.h and not in a module c file. But in that case the information from the external headers are re-exported by the module header file. This re-export could be avoided, therefore the include of such headers could be done (if possible) on c file level.

Include of configuration headers for link time and post build configuration ("<Module>\_Lcfg.h", "<Module>\_Lcfg\_<Sub>.h", "<Module>\_PBcfg.h", "<Module>\_PBcfg\_<Sub>.h") could be done by standard in the module c file. Similar to external headers sometimes it is needed that these headers have to be included in the module header file <Module>.h because if their information are needed from module APIs or inline functions. In that case the information of those headers has a module global visibility and their contents are re-exported to other modules via module header file.

The private module header <Module>\_Prv.h has to be included to use module specific objects with internal linkage.

All listed header files here are optional and shall only be included if they are needed and available.

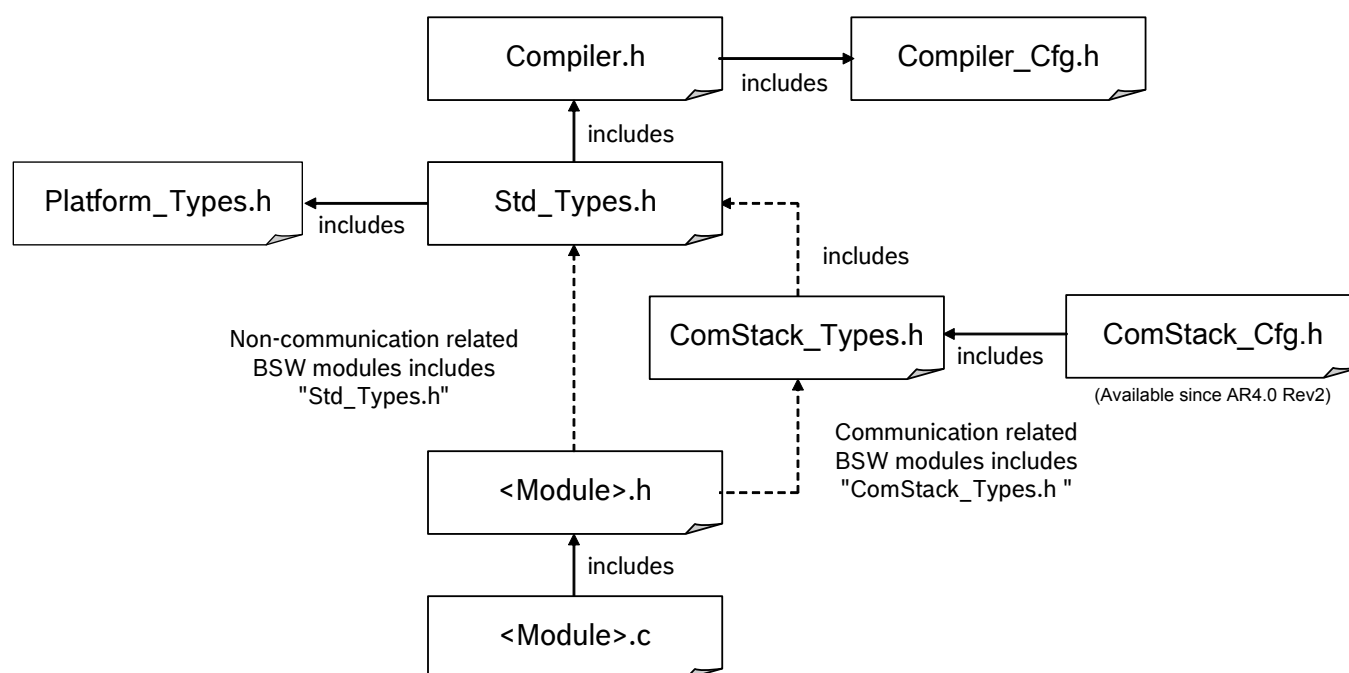
## Rule BSW\_HeaderInc\_004: Include Order of Types Header inside Module Header

**Instruction** Header for standard base data types and symbols shall be included in "<Module>.h" as first header. Communication related modules include "ComStack\_Types.h", all other non communication related modules include "Std\_Types.h".

Both headers provide all common data types and symbols for communication related software modules respectively all other basic software modules. Headers "Platform\_Types.h" and "Compiler.h" are included inside "Std\_Types.h". Contents of these headers are described in [Chapter 3.3 "Rule Set: Types and Symbols"](#). [Figure 10](#) shows a detailed overview of the include structure for types and symbols header.



Figure 10 Include Structure for Headers with Common Types and Symbols



Headers for data types and symbols are included via module header file and not inside module c file. The data types and symbols are provided implicitly over the module header.

### Rule BSW\_HeaderInc\_005: Include Order of other Headers inside Module Header

**Instruction** Further preferred include order of headers in module header file: Header for module specific types ("`<Module>_Types.h`"), optional RTE generated header ("`Rte_<Module>.h`"), optional external module headers ("`<Extmodule>.h`", "`<Extmodule>_Cbk.h`", "`<Extmodule>_<Module>.h`"), configuration header ("`<Module>_Cfg.h`", "`<Module>_Cfg_<Sub>.h`"), optional other configuration headers ("`<Module>_Lcfg.h`", "`<Module>_Lcfg_<Sub>.h`", "`<Module>_PBcfg.h`", "`<Module>_PBcfg_<Sub>.h`"), finally structural sub headers ("`<Module>_Sub.h`", "`<Module>_Sub_Inl.h`").

Only that headers shall be included which shall be exported with the module header. The include order can be changed if the visibility and dependency of contents of headers needs another include order than the preferred one.

As written in explanation of [\[Rule BSW\\_HeaderInc\\_003\]](#) headers "`Rte_<Module>.h`", "`<Extmodule>.h`", "`<Extmodule>_Cbk.h`", "`<Extmodule>_<Module>.h`", "`<Module>_Lcfg.h`", "`<Module>_Lcfg_<Sub>.h`", "`<Module>_PBcfg.h`", "`<Module>_PBcfg_<Sub>.h`" can optionally included in module header file `<Module>.h`. This has to be done if information of those headers is needed on header file level for module specific APIs or inline functions. But it has to be considered that all headers included in the module header file `<Module>.h` will be re-exported to other modules.

The include order can be changed if the visibility and dependency of contents of headers needs another include order than the preferred one.

Examples:

- ▶ The header "`<Module>_Types.h`" could be dependent from "`<Module>_Cfg.h`". In that case "`<Module>_Types.h`" could be included after "`<Module>_Cfg.h`".
- ▶ If the module specific types are not relevant for the API of the module the header "`<Module>_Types.h`" could be included in a corresponding c file.
- ▶ If the configuration headers ("`<Module>_Cfg.h`" and "`<Module>_Cfg_<Sub>.h`") contains no API relevant elements they could be included in a corresponding c file(s) and not in the module header.

This rule is applicable if the listed headers exist.



## Rule BSW\_HeaderInc\_006: Include Order of Private Headers

**Instruction** Include order of headers in private module header file "<Module>\_Prv.h" is: "<Module>\_Prv\_<Sub>.h" for structural private sub headers of the module and "<Module>\_Prv\_<Sub>\_Inl.h" for structural private headers containing inline functions.

This rule is applicable if the listed headers exists. Incidence of the sub headers is 0 ... n.

## Rule BSW\_HeaderInc\_007: Module Export Headers

**Instruction** "<Module>.h", "<Module>\_Cbk.h" and "<Module>\_<User>.h" are module export headers to be included in other modules. These headers shall only export that kind of information which is explicitly needed by other modules.

This rule is there to prevent other modules from accessing functionality and data which is not of their concern. The three export headers have its own use cases.

<Module>.h is the module header and exports APIs and public module elements to other modules. This header has to be included as "<Extmodule>.h" if any API of a module will be used from another module.

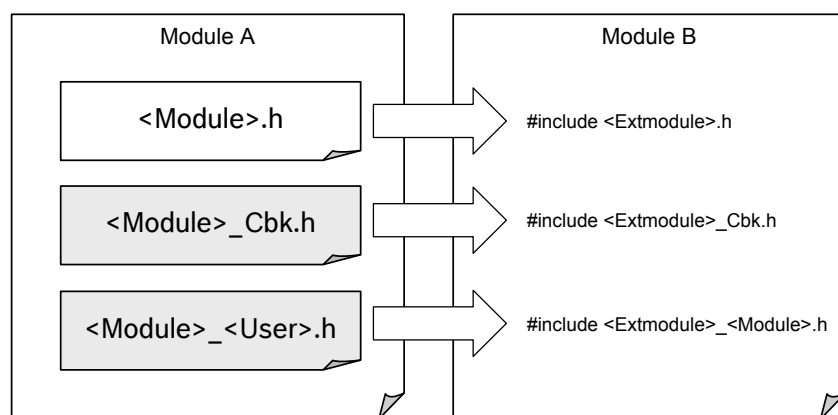
Declarations of callback functions are not a part of the module header, they have to be separated to <Module>\_Cbk.h ( [BSW\_Files\_003] ). Every module that calls callback functions from other modules has to include the callback header from the other module. If no other APIs are needed an include of the callback header (from view of the user module: "<Extmodule>\_Cbk.h") is enough.

The same logic takes place with a header specially provided for a specific module ( [BSW\_Files\_006] ). In this use case it is sufficient that two modules can exchange their APIs by that header (exported as <Module>\_<User>.h, imported as <Extmodule>\_<Module>.h) and an additional include of the module header is not needed.

As specified within [BSW\_Files\_001] <Module>\_Cbk.h and <Module>\_<User>.h are only optional headers (Incidence 0...1 resp. 0...n) and shall only be provided if there is a need/benefit for having them.

In [Figure 15](#) an overview is given for all possible export headers.

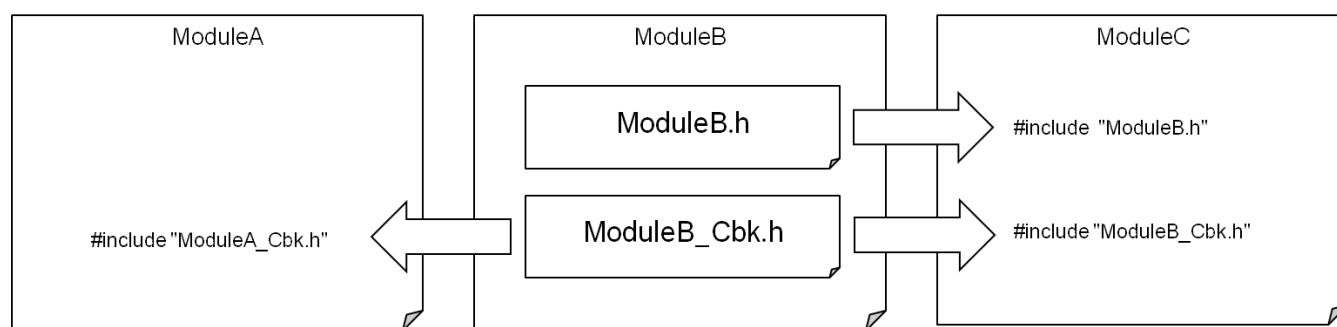
Figure 11 Overview of Export Headers



In the following two examples are given. There are three modules ModuleA, ModuleB and ModuleC. Interfaces from ModuleB are used from ModuleA and ModuleC. Two examples reflect the core of the rule exporting and importing only the relevant headers. Thus the two examples differ in usage and visibility of the provided interfaces.

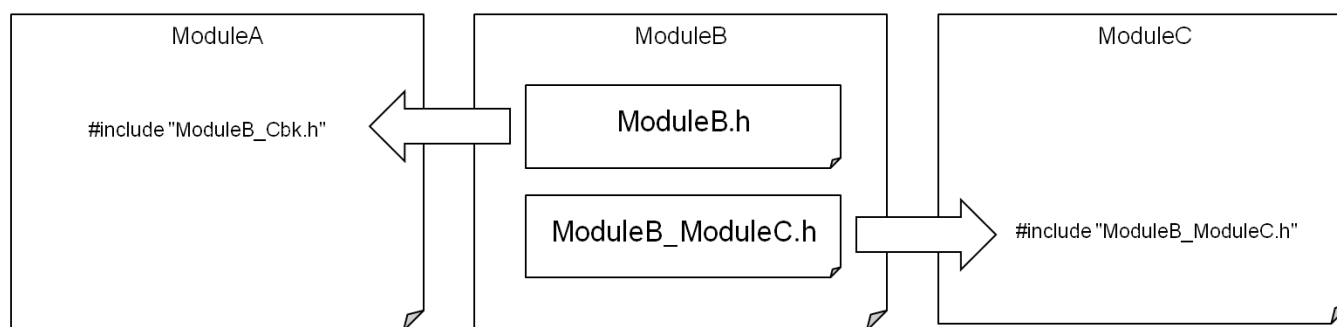
Example 1: ModuleA has only the need for callback functions from ModuleB. Therefore ModuleA only includes the callback header from ModuleB. ModuleC needs callback functions from ModuleB too but in addition also some other exported interfaces from ModuleB. Therefore ModuleC includes the common export header and the callback header from ModuleB.

Figure 12 Example 1



Example 2: ModuleB shares interfaces exclusive with ModuleC and ModuleC does not need other interfaces from ModuleB. ModuleC includes the friends header from ModuleB. ModuleA needs only common exported interfaces from ModuleB. Therefore only the module header is included.

Figure 13 Example 2



## Rule BSW\_HeaderInc\_008: Import of Headers from Other Modules

**Instruction** A module shall only import the necessary header files from other modules which are required to fulfill the modules functional requirements ("*<Extmodule>.h*" and/or "*<Extmodule>\_Cbkg.h*" and/or "*<Extmodule>\_<Module>.h*").

This rule promotes a defensive module layout. Modules shall not import functionality that could be misused. This will shorten compile times. If a module only needs a callback function from another module only the header "*<Extmodule>\_Cbkg.h*" shall be included. If other APIs and objects from another module are needed "*<Extmodule>.h*" shall be included additionally. "*<Extmodule>\_<Module>.h*" provides explicit APIs and elements for a module. The receiver module shall include only the headers which are really needed. For more details see explanations in [\[BSW\\_HeaderInc\\_007\]](#).

## Rule BSW\_HeaderInc\_009: Protection against Multiple Inclusion

### Instruction

**Scope:** All headers except MemMap headers

Each header file shall protect itself against multiple inclusion.

The protection against multiple inclusion avoids multiple re-definitions of headers and objects inside headers.

The protection shall be done in following form:

```
#ifndef HEADERNAME_H
#define HEADERNAME_H
```

```
...
```

```
/* HEADERNAME_H */  
#endif
```

The protection keyword `HEADERNAME_H` has to be replaced by the individual name of the header, written in upper case, and suffixed with a `_H`. E.g. header "eep.h" has a protection keyword `EEP_H` or header "rba\_DioHal.h" has a protection keyword `RBA_DIOHAL_H`.

**Hint** This rule is not applicable for MemMap headers. The basic principle of MemMap headers is that they are included many times. A protection against multiple includes is here counterproductive.

## Rule BSW\_HeaderInc\_010: Preprocessor Check for AUTOSAR Headers

**Instruction** A pre-processor check shall be performed for all included AUTOSAR based header files (Inter Module Check).

With this rule the integration of incompatible imported files shall be avoided. For the update of BSW modules version conflicts shall be detected. Relevant for the check are the major and minor number of AUTOSAR release (defined in [\[BSW\\_VersionInfo\\_001\]](#) . SW versions (which was required within AUTOSAR release R3.1) will not be checked.

Sometimes APIs or other identifiers are different if a module is implemented for a specific AUTOSAR release. Unfortunately global data types and symbols have small differences (e.g. `Std_VersionInfoType` [\[CCode\\_Symbols\\_004\]](#) . That a module will be based on stable information a check for valid versions is recommendable.

Example:

```
#include "Std_Types.h"  
#if (!defined(STD_TYPES_AR_RELEASE_MAJOR_VERSION) || (STD_TYPES_AR_RELEASE_MAJOR_VERSION != 4))  
# error "AUTOSAR major version undefined or mismatched"  
#endif  
#if (!defined(STD_TYPES_AR_RELEASE_MINOR_VERSION) || (STD_TYPES_AR_RELEASE_MINOR_VERSION != 0))  
# error "AUTOSAR minor version undefined or mismatched"  
#endif
```

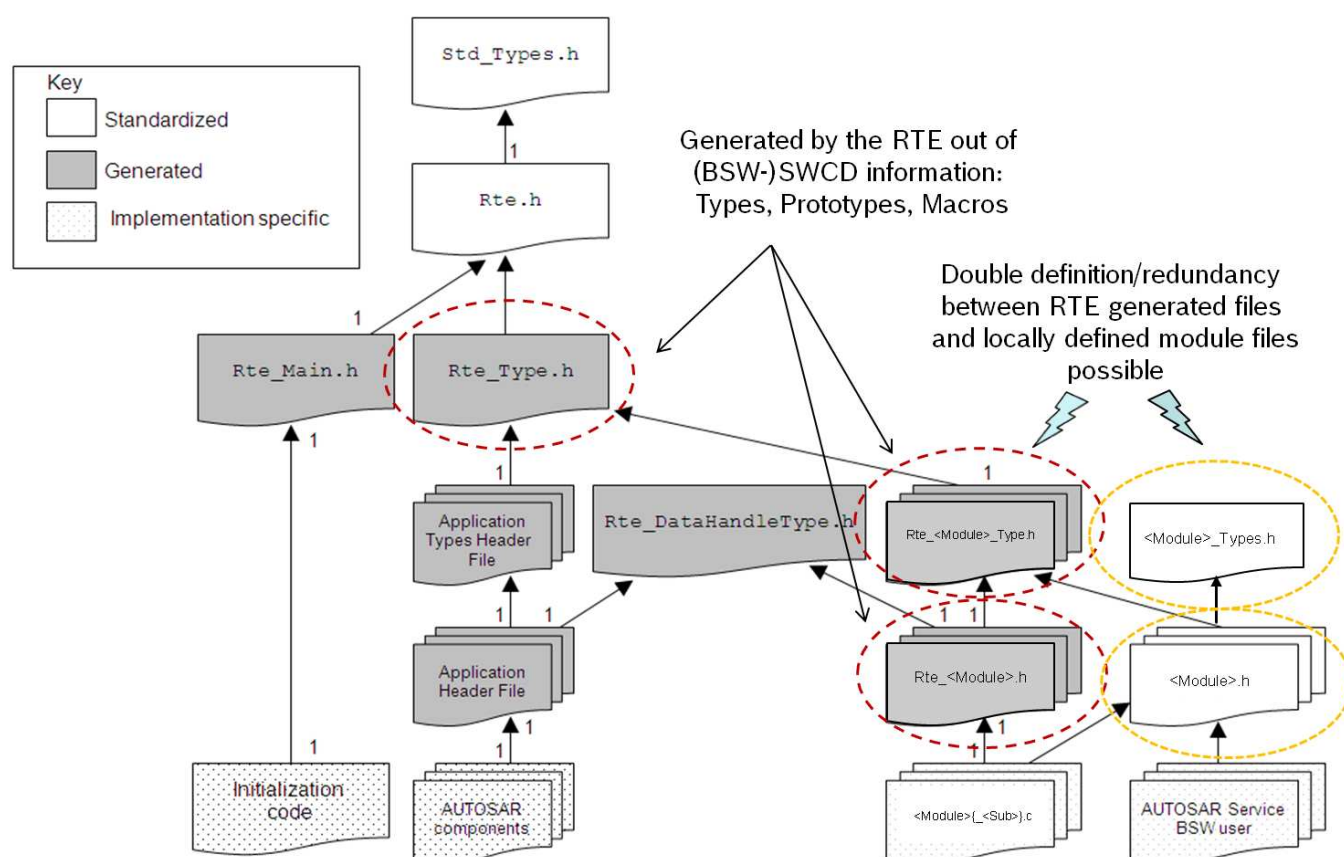
## 3.5.3 BSW Service Module with and without RTE

BSW service modules mostly have a close connection to the *RTE* because such modules provide a SWCD file which specifies types and service functions. The *RTE* will generate headers which contain definitions, macros and functions which are based on the elements described in the SWCD file. In parallel the service module has local headers containing also definitions of types and functions. Because of the defined include order there can be a conflict or double definition of such elements. Otherwise it is possible that a project does not use a *RTE* but will use the BSW service module. It shall be possible that a BSW service module can be used without the availability of the *RTE*. This chapter will describe a concept how to avoid the type/function conflicts and to be independent of the existence of a *RTE* generator in the project. Finally following goals will be reached:

- ▶ Ensured avoidance of conflicts in project with a *RTE* generator caused of multiple declarations of types, macros, functions, ...
- ▶ To make it possible that a service module can be used in projects with and without a *RTE* generator without changing the concerned BSW module or without changing other BSW modules

The following [Figure 14](#) will illustrate the issue based on the relevant headers and the include order of the headers.

Figure 14 Overview Service Module and RTE



## Rule BSW\_ServiceRTE\_001: BSW Service Module with and without RTE: Conditions for RTE headers

### Instruction

**Scope:** BSW service modules which provide a SWCD file and have a close connection to the *RTE* generator

To support that a BSW service module can work with and without a *RTE* generator following *RTE* specific files shall be provided with following content:

- ▶ For the header "Rte\_<Module>\_Type.h" a header template "Rte\_<Module>\_Type.h\_tpl" shall be provided including all the types and macros required from the BSW service module (all elements which are specified in the SWCD file)
- ▶ For the header "Rte\_<Module>.h" also a header template "Rte\_<Module>.h\_tpl" shall be provided containing only the include of "Rte\_<Module>\_Type.h", but nothing more
- ▶ The "<Module>\_SWCD.arxml" file shall be provided even if no *RTE* is available, but in case of an existing *RTE* generator this file is anyway mandatory

In case of an integration of the BSW service module to a project without a *RTE* generator the "real" headers "Rte\_<Module>\_Type.h" and "Rte\_<Module>.h" are derived from the template headers by the integrator. With that proceeding the ASW component can include the *RTE* headers conform to the ASW header include concept and with no change of the ASW component. Also the BSW component have the access to all headers which are needed from the BSW header include concept. Otherwise it would also be possible that the BSW service module itself generates the real *RTE* headers by the configuration script. The decision is up to the module developer.

In case if a *RTE* generator is available the template files are unused and the real headers are generated from the *RTE* generator. Therefore the BSW service module can be integrated to a project containing the *RTE* without a specific integration effort.

**Hint** It is obligatory that users of the mechanism described in this chapter are documenting it in the delivery note in the section "Necessary code changes". This is needed to inform the integrators that the template headers "Rte\_<Module>\_Type.h\_tpl" and "Rte\_<Module>.h\_tpl" has to be derived to "real" headers in case of that no *RTE* is available in the project. Text module for the delivery note could be similar to:

"In case of a project without a *RTE* generator the header "Rte\_<Module>\_Type.h\_tpl" has to be derived to "Rte\_<Module>\_Type.h" and "Rte\_<Module>.h\_tpl" has to be derived to "Rte\_<Module>.h". Both headers have to be handled as integration code. If a *RTE* generator is existing in the project the mentioned headers have not to be derived."

## Rule BSW\_ServiceRTE\_002: BSW Service Module with and without RTE: Conditions for Module Header

### Instruction

**Scope:** BSW service modules which provide a SWCD file and have a close connection to the *RTE* generator

To support that a BSW service module can work with and without a *RTE* generator the module header / the module types header shall fulfill following conditions:

- ▶ The module header file "<Module>.h" and the header file "<Module>\_Types.h" (this is only an optional header) shall contain only that elements which are needed from the BSW service module itself and which are not part of the *RTE* based headers
- ▶ Function prototypes shall be placed in "<Module>.h" even if they are also generated in "Rte\_<Module>.h". Here a crosscheck in the "<Module>{<Sub>}.c" file is intended.
- ▶ The module header file "<Module>.h" shall not include the header "Rte\_<Module>.h"
- ▶ The module header file "<Module>.h" shall include the header "Rte\_<Module>\_Type.h"
- ▶ The module c file "<Module>{<Sub>}.c" shall include "<Module>.h" and "Rte\_<Module>.h"

Example based on a NvM module:

#### **NvM.h:**

```
#include "Rte_NvM_Type.h" /* Contains possibly RTE generated types */
#include "NvM_Types.h"
#include "NvM_Cfg.h"
/* Further declarations of interfaces required also by BSW */
/* This includes function prototypes which already appear in Rte_NvM.h in the RTE case */
```

#### **NvM\_Types.h:**

```
/* Further declarations for BSW-exclusive interfaces */

Rte_NvM.h: (Without RTE: This header is stubbed empty derived from a template file)
#include "Rte_NvM_Type.h" /* Only if RTE is generated */
extern NvM_ReadBlock() /* Only if RTE is generated */
...
```

**Rte\_NvM\_Type.h:** (Without RTE: This header is derived from a template file or is generated based on the template if config-dependent.  
The content is similar to the expected types that come via Rte\_Type.h and Rte\_NvM\_Type.h in case of existing RTE)

```
#define NVM_REQ_OK 0 /* Similar to RTE */
typedef uint8 NvM_RequestResult; /* With a RTE, this would be in Rte_Type.h */
```

**Rte\_Type.h:** => Does not exist with systems without RTE

**NvM.c:**

```
#include "Std_Types.h"
#include "NvM.h"
#include "Rte_NvM.h"          /* Intended crosscheck of the function prototypes are      */
                              /* identical between for BSW and ASW/SWC for some modules      */
                              /* according to AUTOSAR                                          */
```

**Nvm\_SWCD.xml:** (Contains all relevant types and definitions. In case without RTE it is ignored, in case of a RTE it triggers the generation of the RTE based headers)

## 3.5.4 Design of APIs (Application Programming Interfaces)

### Rule BSW\_APIDesign\_001

Is removed and replaced by rule [\[BSW\\_ProcISR\\_001\]](#) .

### Rule BSW\_APIDesign\_002

Is removed and replaced by rule [\[BSW\\_ProcISR\\_002\]](#) .

### Rule BSW\_APIDesign\_003: Callback Functions

**Instruction** Callback functions shall avoid return types other than void if possible.

Callback functions routed to Software Components (SWCs) via the RTE shall have return type Std\_ReturnType and not void. In this case the caller can assume that always E\_OK is returned. Callbacks shall be used for notifications and they should never fail. Callback functions are allowed to have parameters.

### Rule BSW\_APIDesign\_004: Prohibition of Function Pointers as Parameter

**Instruction** Function pointers shall not be used as a parameters of an API.

Rationale for this rule is a protected Operating System compatibility. Callbacks shall be defined statically at compile time, not during runtime.

### Rule BSW\_APIDesign\_005: Separation of Error and Status Information

**Instruction** All software modules shall strictly separate error and status information in return values and also in internal variables.

This rule is a common API specification of AUTOSAR Basic Software Modules.

Example (EEPROM driver):

A module status is e.g. the state of a state machine and can be read by a separate Eep\_GetStatus() function:

```
- EEP_UNIT
- EEP_IDLE
- EEP_BUSY
```

Error values are reported to the Development Error Tracer (if enabled):

```
- EEP_E_BUSY
- EEP_E_PARAM_ADDRESS
- EEP_E_PARAM_LENGTH
```

If the EEPROM driver is idle (EEP\_IDLE) and is called with wrong parameters, the error is reported to the Debug Error Tracer, but the module status stays EEP\_IDLE!!

## Rule BSW\_APIDesign\_006: Design of Instances of BSW Modules

**Instruction** Instances of BSW modules shall be accessed index based if they are characterized by same vendor, same functionality and same hardware device.

Example:

```
MyFunction(uint8 MyIdx, MyType MyParameters, ... );
Or optimised for sourcecode delivery:
#define MyInstance(index, p) Function##index (p)
```

## Rule BSW\_APIDesign\_007: Unit of Time Parameters

**Instruction** The unit of time for specification and configuration of BSW modules shall be preferably in physical time unit, not ticks.

The duration of a "tick" varies from system to system. The software specification defines the unit (e.g. µs, s) and software configuration uses these units.

## Rule BSW\_APIDesign\_008: Usage of Standard Return Type

**Instruction** APIs with a connection to *RTE* have to use "Std\_ReturnType" as standard API return type.

Definition of standard return type:

```
typedef uint8 Std_ReturnType;
```

Elements of the standard return type are shared between the *RTE* and the BSW module. The layout of the Std\_ReturnType is stated in the *RTE* specification. Standard symbols E\_OK and E\_NOT\_OK can be used as first bit of the standard return type. The following 5 bits can be specific bits for further error codes. Bit 7 and Bit 8 are reserved and defined by the *RTE* specification. The standard return type has to be used always even when there is success at all the time.

APIs with no connection to the *RTE* can have no return type or a module specific one. There are two possibilities for a module specific return type:

- ▶ "uint8" is used as return type. Standard symbol E\_OK is used plus additional return values defined as #define.

Example:

```
Definition of an API: uint8 Can_Write(...)
Return values: E_OK (0), CAN_E_BUSY (1), CAN_E_FAILED (2)
E_OK is taken from Std_Types.h, CAN_E_BUSY and CAN_E_FAILED are #defines in can.h.
```

**Hint** No strong type checking is possible because return type is an uint8 and values are only #defines. E\_OK can be used in this case.

- ▶ A module specific return type is used defined with "typedef enum". Here standard symbol E\_OK cannot be used (because E\_OK is still a #define).

Example:

```
Definition of an API: Can_ReturnType Can_Write(...)
Return values: CAN_OK_E, CAN_E_BUSY_E, CAN_E_FAILED_E
```

```
Can_ReturnType is an enumeration type in can.h:
typedef enum
{
```



```
05     CAN_OK_E = 0,  
        CAN_E_BUSY_E,  
        CAN_E_FAILED_E  
    } Can_ReturnType;
```

**Hint** Strong type checking is possible because only the values of the enumeration may be assigned to variables of type Can\_ReturnType. E\_OK cannot be used in this case!

## Rule BSW\_APIDesign\_009: Test of Error Information

**Instruction** If a function returns error information, then that error information shall be tested.

A function may provide some means of indicating the occurrence of an error. This may be via an error flag, some special return value or some other means. Whenever such a mechanism is provided by a function the calling program shall check for the indication of an error as soon as the function returns.

**Hint** SW developer has to check that a function returns error information or not. A return value which represents no error informations could be ignored but a return value with error information shall be tested. Check tools cannot distinguish that a return value represents error information or not. A comment can be set if a warning comes up.

## Rule BSW\_APIDesign\_010: Stable Number of Parameters

**Instruction** Functions shall not be defined with a variable number of arguments.

There are a lot of potential problems with such features.

## Rule BSW\_APIDesign\_011: Prohibition of Recursive Function Calls

**Instruction** Functions shall not call themselves, either directly or indirectly. Recursive function calls are not allowed.

Recursive function calls might lead to exceeding of available stack space, which depends on parameters of the function and finally the number of recursive function calls. Unless recursion is very tightly controlled, it is not possible to determine before execution what the worst-case stack (and other resources) usage could be. Therefore recursive function calls shall not be used.

## Rule BSW\_APIDesign\_012: Call of Functions with Parameters

**Instruction** The number of arguments passed to a function shall match the number of parameters.

Normally a misuse of a function is completely avoided by the use of function prototypes. Before a function can be called a header file has to be included where the function is declared. [\[BSW\\_FuncObj\\_006\]](#) and the *"Header Include Concept"* will help to fulfill this rule. On the other hand the compiler gives a warning if a function is called with too few or too many arguments.

## Rule BSW\_APIDesign\_013: Explicit Return Statement

**Instruction** All exit paths from a function with non-void return type shall have an explicit return statement with an expression.

This expression gives the value that the function returns. The absence of a return with an expression leads to undefined behaviour (and the compiler may not give an error). Each function with a non-void return type shall have at least one



explicit return statement with an explicit expression inside of the function. It is not allowed to write no return statement or to write a return statement without an expression.

Examples:

```
uint32 MyModule_Func(uint16 Arg1_u16, uint16 Arg2_u16)
{
    uint32 result_u32;

    result_u32 = Arg1_u16 + Arg2_u16;

    return (result_u32);    /* OK: It is a failure if this line is not */
                          /* existing or there is only a 'return;'   */
}
```

## Rule BSW\_APIDesign\_014: Call of Functions via Identifier

**Instruction** A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.

When a function shall be explicitly called the function name has to be followed by parentheses (even if they are empty because of a void function). Without parentheses the function name is treated as a function pointer. In that case the function name shall be preceded by a & address operator. The advantage of inserting a & operator is that the nature of the function reference is made explicit and there is no possibility of confusion when parentheses are omitted. If the function name is used e.g. in an if-expression without a & operator or parentheses a constant non-zero value is returned. It is here not clear if the intent was to test that the address of the function is not NULL or a call of the function should be made.

Example:

```
extern uint32 ExtModule_Func(uint32 Arg1_u32);

void MyModule_Func(void)
{
    uint32 Val_u32;
    ...

    ExtFunc;                /* Not OK: Parameter list is missing */

    if (ExtModule_Func)     /* Not OK: Here it is not clear if the address */
    {                       /* of the function is checked or a call is intended */
        ...                /* Only a constant non-zero value is given. */
    }

    if (ExtModule_Func(Val_u32) > 0)
    {
        ...                /* OK: A correct function call with return check */
    }
}
```

## Rule BSW\_APIDesign\_015: Structure Passed to Functions as Pointer

**Instruction** A structure as parameter for a function should be passed as pointer.

A pointer to a structure is the most efficient way to pass a structure to a function via parameter. But consider that the content of the structure shall be stable until the function call is finished. The function could be interrupted and the values of the structure could be modified (side effects). If this constraint is not fulfilled other mechanisms have to be used to ensure the intended behaviour (semaphores, disable interrupts, etc). In all other cases this rule helps to write efficient code.

Example:

```
/* Typedef in a header file: */
typedef struct
{
    uint8 x;
    uint8 y;
    uint8 z;
} MyModule_Vector_tst;

/* Prototype of a function: */
uint8 MyModule_VectorGetLength(const MyModule_Vector_tst *vector);

/* Usage in a c file: */
uint8 vectorlength;
MyModule_Vector_tst Vector1_st;
...
vectorlength = MyModule_VectorGetLength(&Vector1_st);
```

## Rule BSW\_APIDesign\_016: Handling of Unused Parameters

**Instruction** For unused parameters a special handling is used to avoid compiler warnings.

Sometimes there are APIs which have parameters which are not used but shall be available regarding other reasons (compatibility of the API, multiple instances of an API). Some compilers give a warning when a parameter is not used inside the function. To avoid that warning the corresponding unused parameter can be handled with a void cast as shown in the example below. With this construct the compiler warning is suppressed because the parameter is used. This construct generates no code and no MISRA warning.

Example:

```
void MyApi(uint32 Value1, uint32 Value2)
{
    /* "Value2" is defined because of compatibility aspects but is not used. */
    /* Following line uses the parameter and avoids a compiler warning: */
    (void)Value2;

    ...
}
```

## Rule BSW\_APIDesign\_017: Change of an Interface leads to a New Interface

**Instruction** To ensure compatibility a change of an interface or an changed interpretation of interface parameters shall always leads to a definition of a new interface.

It is possible that changes in software results in an implicit modification of an interface e.g. parameters have to be interpreted in a different way. It is not allowed that the name of the interface is kept while parameters of the interface (which are still unmodified in their data type and name) will be interpreted in a different way. In such cases a new interface with a new name has to be provided in parallel to the existing interface.

Another example are adapters for translation of interfaces. To avoid failures in late development phase it is not allowed to use same adapters with same function name and same function interface with different meaning of the interface. A different function name shall be used instead to detect the failure at compilation time.

Example:

```
/* Existing interface in MyModule: */
/* The first parameter is interpreted as byte position inside the field */
MyModule_AdaptGetValue_u8(uint8 position_u8, uint8 * field_pu8)
{
    return field_pu8[position_u8];
}
```

*Now there is a request to modify the interface that the position*

*parameter interprets the bit position inside the field.*

```
/* Not OK: Unchanged name of interface but changed functionality */
MyModule_AdaptGetValue_u8(uint8 position_u8, uint8 * field_pu8)
{
    return (field_pu8[position_u8 >> 4] >>(position_u8 % 8));
}

/* OK: New name of the interface, changed functionality */
MyModule_AdaptGetValueBitPos_u8(uint8 position_u8, uint8 * field_pu8)
{
    return (field_pu8[position_u8 >> 4] >>(position_u8 % 8));
}
```

### Rule BSW\_APIDesign\_018: APIs are Functions Called by Other Components

**Instruction** APIs shall be executable on all cores.

Other modules running on different cores directly execute the API functions on the same core. The API functions therefore shall be reentrant, see 3.5.7. They also shall be able to run in parallel to the scheduled processes and interrupt service routines.

## 3.5.5 Design of Processes and Interrupt Service Routines

### Rule BSW\_ProcISR\_001: Return Value of Initialization Processes

**Instruction** The return type of initialization processes shall be void.

No return value is used when calling initialization processes.

### Rule BSW\_ProcISR\_002: Parameters of Initialization Processes

**Instruction** If post build selectable configuration is enabled for a module, the initialization processes shall have a pointer to it as parameter, otherwise there shall be no parameter.

Initialization Processes called in EcuM\_DriverInitListOne receives the selected configuration via parameter if this feature is enabled for the module.

Errors in initialization data shall be detected during configuration time (e.g. by configuration tool or configuration processor).

### Rule BSW\_ProcISR\_003: Interface of Scheduled Processes

**Instruction** Scheduled processes shall have no parameters and no return value.

Many modules have a function that has to be called cyclically (e.g. within an OS Task) and that does the main work of the module.

Example:

```
void Eep_MainProcess(void);
```

### Rule BSW\_ProcISR\_004: Internal Design of Scheduled Processes

**Instruction** Scheduled processes shall be designed to run in parallel with APIs and ISRs on other cores.

It can be assumed that one process is not interrupted by itself or runs two times in parallel.

## Rule BSW\_ProcISR\_005: Interface of Interrupt Service Routines

**Instruction** Interrupt Service Routines shall have no parameters and no return value.

## Rule BSW\_ProcISR\_006: Internal Design of Interrupt Service Routines

**Instruction** Interrupt Service Routines shall be executable on any core.

Interrupt service routines are started by hardware events and shall return as fast as possible. It can be assumed that an ISR is not interrupted by itself or two instances run in parallel.

## 3.5.6 Definition and Declaration of Functions and Objects

### Rule BSW\_FuncObj\_001: Definitions only in C Files

**Instruction** There shall be no definitions of objects or functions (except inline functions) in a header file. Definitions shall take place in a c file.

Header files should be used to declare objects (global visible variables and constants), functions, typedefs, and macros. Header files shall not contain or produce definitions of objects or functions (or fragments of functions or objects) that occupy storage. This makes it clear that only C files contain executable source code and that header files only contain declarations. This rule avoids multiple definitions and uncontrolled spreading of global objects and functions and limits visibility of such elements.

The only exception are inline functions which have to be defined in header files ( [\[Abstr\\_Inline\\_001\]](#) ). If a compiler cannot handle inline functions this rule can be skipped.

### Rule BSW\_FuncObj\_002: Explicit Functions and Objects

**Instruction** Whenever an object or function is declared or defined, its object or return type shall be explicitly stated. Functions with no explicit return type shall be defined as void function.

This rule avoids implicit types on declaration or definition level.

Examples (simplified, without memory mapping concept):

```
static      Func1(void);    /* Not OK - implicit type */
static sint8 Func1(void);   /* OK - explicit type      */

static      Func2(void);    /* Not OK - implicit type */
static void  Func2(void);   /* OK - void function      */

const       x;              /* Not OK - implicit type */
const sint32 x_S32;         /* OK - explicit type      */

extern      y;              /* Not OK - implicit type */
extern uint8 y_u8;          /* OK - explicit type      */
```

### Rule BSW\_FuncObj\_003: Void Functions

**Instruction** Function with no parameters shall be declared and defined with the parameter list void.

Functions shall be declared with a return type (see [\[BSW\\_FuncObj\\_002\]](#) ), that type being void if the function does not return any data. Similarly, if the function has no parameters, the parameter list shall be declared as void.

Example for a function with no parameter and no return value:

```
void MyFunc(void)
```

## Rule BSW\_FuncObj\_004: Singulary Definitions of Objects and Functions

**Instruction** An object or function with external linkage or global visibility shall have exactly one definition in one file.

The behaviour is undefined if an identifier is used for which multiple definitions exist (in different files) or no definition exists at all. Multiple definitions in different files are not permitted even if the definitions are the same, and it is obviously serious if they are different, or the identifiers are initialized to different values.

## Rule BSW\_FuncObj\_005: Scope of Functions

**Instruction** Functions shall be declared at file scope and not on block scope.

An identifier has file scope when its declaration is situated outside of any function definition. Accordingly an identifier has block scope when its declaration occurs within a function definition, i.e. within a code block. Declaration of functions on block scope is not allowed because it may be confusing and can lead to undefined behaviour.

## Rule BSW\_FuncObj\_006: Prototype Declarations

**Instruction** External functions, inline functions and external objects shall have prototype declarations in one and only one header file. The prototypes shall be visible 1st at the function and object definition and 2nd the function call or object use. Static functions or static objects shall not have a prototype declaration in a header file. Static functions or static objects do not need a prototype.

The use of prototypes enables the compiler to check the integrity of function definitions and calls, and of object definition and use. Without prototypes the compiler is not obliged to pick up certain errors in function calls (e.g. different number of arguments from the function body, mismatch in types of arguments between call and definition). Function interfaces have appeared to be a cause of considerable issues, and therefore this rule is very important.

The header include concept (chapter 3.5.2) is the recommended method to ensure that this rule is kept and the functions and objects are visible at the definition and usage locations. For a module external visible function a prototype has to be declared in the module header file `<Module>.h` (or in another header file which is included in the module header file). The module header file is included in the `<Module>.c` file where the related function is implemented and a user has to include this header too, to be able to use the function in his module. Module global internal functions shall define the prototype in the private header `<Module>_Prv.h`. Inline functions need a prototype as well which is written at the top of the header where the inline functions is defined. Functions defined and used file local do not need a prototype declaration if these functions are defined before they are used in the corresponding c file.

Example (simplified, only the principle is shown):

### **Hugo.h:**

```
/* External prototype declaration: ****/  
extern void Hugo_Func(uint16 param1, uint16 param2);
```

```
/* External identifier: *****/  
extern uint8 Hugo_Variable_u8;
```

### **Hugo.c:**

```
/* Includes *****/  
#include "Hugo.h"
```

```
/* Global variable: *****/  
uint8 Hugo_Variable_u8;
```

```
/* Implementation *****/  
void Hugo_Func(uint16 param1, uint16 param2)  
{
```

```

05     ...
    }

Paula.c:
/* Includes *****/
...
#include "Paula.h"
#include "Hugo.h"
...

/* Use of external function/variable **/
void Paula_Func(void)
{
    uint8 Paula_Variable_u8 = Hugo_Variable_u8;
    ...

20    Hugo_Func(val1, val2);
    ...
}

```

**Hint** This rule cannot be maintained if a header file generation concept is used where specific header files are generated exactly focussed on the demand of a module. Here the same external prototypes and identifiers are generated in multiple module specific header file where the corresponding functions or objects are needed. Such a concept is used in ASW where the RTE generates component specific header files. BSW works with explicit module header files which are not generated and therefore the prototypes and identifiers can be located exactly in one header file.

## Rule BSW\_FuncObj\_007: Equality of Declaration and Definition of a Function

**Instruction** For each function parameter the type given in the declaration and definition shall be identical, and the return types shall be identical too. Additional parameter names shall be given for all parameters in the function prototype declaration and they shall be identical to the parameter names of the function definition.

The types of the parameters and return values in the prototype and the definition shall match. This requires identical types including typedef names and qualifiers, and not just identical base types. Names shall be given for all the parameters in the function declaration for reasons of compatibility, clarity and maintainability.

```

<Module>.h:
/* External prototype declaration: ****/
extern void Module_Func(uint16 param1, uint16 param2);
...

50 <Module>.c:
/* Includes *****/
#include "<Module>.h"
...

55 /* Implementation *****/
void Module_Func(uint16 param1, uint16 param2)
{
    ...
}

```

\ Return type, data types  
/ and names of parameters  
/ have to be identical.

## Rule BSW\_FuncObj\_008: Static Objects and Functions

**Instruction** All declarations and definitions of objects or functions at file scope shall have internal linkage using the static storage class specifier unless external linkage is required.

If a variable is only to be used by functions within the same file then the static storage class specifier shall be used. Similarly if a function is only called from elsewhere within the same file, it shall be defined as *static*. Use of the static storage-class specifier will ensure that the identifier is only visible in the file in which it is declared and avoids any possibility of confusion with an identical identifier in another file or a library. Therefore it is good practice to apply the *static* keyword consistently to all declarations of objects and functions with internal linkage.

All objects and functions with external linkage (shall be used from other modules) needs an external prototype and identifier in a header file as declared with [\[BSW\\_FuncObj\\_007\]](#).

**Hint** For the keyword "static" AUTOSAR defines encapsulated macros "STATIC" (R3.0 and R3.1) and "\_STATIC\_" (R2.x). They shall not be used, only "static" is allowed even if a module is designed for AUTOSAR R3.1 or smaller.

## Rule BSW\_FuncObj\_009: Constant Pointer Parameters

**Instruction** A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.

This rule leads to greater precision in the definition of the function interface. The const qualification should be applied to the object pointed to, not to the pointer, since it is the object itself that is being protected.

Example:

```
void MyModule_Func(uint16 * Param1_pul6, const uint16 * Param2_pcul6, uint16 * Param3_pul6)
    /* Param1_pul6:  Addresses an object which is modified - no const          */
    /* Param2_pcul6:  Addresses an object which is not modified - const required */
    /* Param3_pul6:  Addresses an object which is not modified - const missing  */
{
    *Param1_pul6 = *Param2_pcul6 + *Param3_pul6;
}

/* Data at address Param3_pul6 has not be changed          */
/* -> This parameter should be declared as pointer to const data */
```

## Rule BSW\_FuncObj\_010: Compatible Types of multiple declared Objects of Functions

**Instruction** If objects or functions are declared more than once their types shall be compatible.

The definition of compatible types is lengthy and complex. Two identical types are compatible but two compatible types need not be identical.

E.g. it is not allowed to have two variables with same name but different data types defined within one function, or a function which has two different external prototype declarations with different types.

```
void MyFunction1(void)
{
    extern uint32 GlobVar;
    /* ... */
}

void MyFunction2(void)
{
    extern uint16 GlobVar;    /* Not OK, two declarations for the same global */
                             /* identifier are in conflict. The declarations */
                             /* occur in different scopes but are inconsistent. */

    sint32 localvar;
    /* ... */
    sint8 localvar;          /* Not OK, two declarations of the same identifier */
                             /* are in conflict. Both declarations have been */
                             /* made at the same scope.                      */
```

```

05      /* ... */
    }

```

## Rule BSW\_FuncObj\_011: Global Constants

**Instruction** Global visible constant data shall be indicated with read-only purposes by explicitly assigning the `const` keyword.

In principle, all global data shall be avoided due to extra blocking efforts when used in preemptive runtime environments. Unforeseen effects will occur if no precautions were taken. If data is intended to serve as constant data, global exposure is permitted only if data is explicitly declared read-only using the `const` modifier keyword.

In dependence on [BSW\_FuncObj\_001] global constants have to be defined in a c file. Additionally an external declaration shall be placed in a header file to make the global constant visible for other modules ([BSW\_FuncObj\_006]).

Example (simplified, without memory mapping concept):

```
const uint8 MaxPayload_cu8 = 0x18;
```

## 3.5.7 Code Re-entrancy

### Rule BSW\_Reentrancy\_002: Services and APIs are Multicore Re-entrant

**Instruction** All services and APIs of a module shall be multicore re-entrant.

**Re-entrant Definition (SingleCore):** A function is re-entrant if it can be interrupted somewhere in its execution and then safely called again before its previous invocation completes its execution and produces correct results for both invocations.

**Re-entrant Definition (MultiCore):** A function is re-entrant if it can be called on one core while it is already executing on any other core and still produce correct results for all calls.

Consequence:

- ▶ No use of any static / global variable without resource protection (lock) if data consistency is required. Accesses from Services (APIs), Processes and especially Interrupt routines have to be considered.
- ▶ No assumptions about priorities and scheduling possible (e.g. for interrupt code)
- ▶ No modification of its own code
- ▶ No call of other non re-entrant functions

Examples:

**Non re-entrant code:**

```

sint32 a = 1;

sint32 foo(void)
{
    a += 2;
    return a;
}

sint32 bar(void)
{
    return (foo() * 10);
}

```

**Same functionality re-entrant:**



```

05  sint32 foo(sint32 a)
    {
        return (a + 2);
    }

10  sint32 bar(sint32 a)
    {
        return (foo(a) * 10);
    }

```

#### Non re-entrant code:

```

15  sint32 a = 1;

    sint32 foo(void)
20  {
        SCHM_ENTER_<Module>_<Res_a>();
        a += 2;
        SCHM_EXIT_<Module>_<Res_a>();
        /* -> value can be modified concurrently */
25  return a;
    }

```

#### Same functionality re-entrant:

```

30  sint32 a = 1;

    sint32 foo(void)
    {
        sint32 l_a;

35  SCHM_ENTER_<Module>_<Res_a>();
        a += 2;
        l_a = a;
        SCHM_EXIT_<Module>_<Res_a>();

40  return l_a;
    }

```

### Rule BSW\_Reentrancy\_003

is removed and replaced by rules of chapter 3.5.5 .

### Rule BSW\_Reentrancy\_004: Atomic data access

**Instruction** Assumption about atomic data access shall be documented in the chapter for integration in the product documentation.

On 32 bit CPU architectures access to naturally aligned 8, 16 and 32 bit values is atomic. As a consequence no lock for such simple accesses is needed.

Note: Read-Modify-Write C-Instructions ( e.g. ++, --, +=, -=, ...) are not atomic and still have to be protected.

## 3.5.8 Providing of Version Information

### Rule BSW\_VersionInfo\_001: Provision Version Information

#### Instruction

**Scope:** Software based on an AUTOSAR SWS

Each AUTOSAR based BSW module shall provide information to identify vendor, module and SW version.

The provision of version information is relevant for AUTOSAR related BSW modules. Such modules are able to provide a vendor ID, module ID, an AUTOSAR specification number and a SW version. This eases the integration of different BSW modules.

## Rule BSW\_VersionInfo\_002: Module Vendor Identifier

### Instruction

**Scope:** Software based on an AUTOSAR SWS

A module vendor identifier "<MODULE>\_VENDOR\_ID" shall be provided in the module header file.

For Bosch the vendor identification number is "6". The vendor Id is defined by HIS.

Example:

```
In file Adc.h:
#define ADC_VENDOR_ID 6
```

## Rule BSW\_VersionInfo\_003: Module Identifier

### Instruction

**Scope:** Software based on an AUTOSAR SWS

A module identifier "<MODULE>\_MODULE\_ID" shall be provided in the module header file.

Valid module identifiers of AUTOSAR based BSW modules are listed in [Chapter C "List of Basic Software Modules"](#).

Example:

```
In file Adc.h:
#define ADC_MODULE_ID 123
```

## Rule BSW\_VersionInfo\_004: Software Version Number and AUTOSAR Specification Version Number

### Instruction

**Scope:** Software based on an AUTOSAR SWS

A software version number and an AUTOSAR specification version number shall be provided in the module header file.

The software version number shall be updated even before a new version of the software will be released.

The AUTOSAR specification version number shall be updated even if the module supports a new version of an AUTOSAR specification.

This rule is needed to check compatibility and to supervise configuration.

To identify the major/minor/patch version of the vendor specific implementation of the module the following numbers shall be provided:

- ▶ <MODULE>\_SW\_MAJOR\_VERSION
- ▶ <MODULE>\_SW\_MINOR\_VERSION
- ▶ <MODULE>\_SW\_PATCH\_VERSION

In the SCM system the following syntax for a version number is defined: <Alternative>.<Major>.<Minor>.<Bugfix>\_<Userdefined>. Major, Minor, Bugfix numbers are used for the definitions above.

The version numbers shall be enumerated according to the following rules:

- ▶ Increasing a more significant digit of a version number resets all less significant digits

- ▶ The PATCH\_VERSION is incremented if the module is still upwards and downwards compatible (e.g. bug fixed)
- ▶ The MINOR\_VERSION is incremented if the module is still downwards compatible (e.g. new functionality added)
- ▶ The MAJOR\_VERSION is incremented if the module is not compatible anymore (e.g. existing API changed)

Example:

Eep module with version 1.14.2:

- Versions 1.14.2 and 1.14.9 are exchangeable. 1.14.2 may contain bugs
- Version 1.14.2 can be used instead of 1.12.0, but not vice versa
- Version 1.14.2 cannot be used instead of 1.15.4 or 2.0.0

Next information is the major/minor/revision release version number of the AUTOSAR specification which the appropriate implementation is based on.

- ▶ <MODULE>\_AR\_RELEASE\_MAJOR\_VERSION
- ▶ <MODULE>\_AR\_RELEASE\_MINOR\_VERSION
- ▶ <MODULE>\_AR\_RELEASE\_REVISION\_VERSION

Entire example:

ADC vendor module version 1.14.9; implemented according to the AUTOSAR Release 4.0, Revision 2

```
#define ADC_SW_MAJOR_VERSION 1
#define ADC_SW_MINOR_VERSION 14
#define ADC_SW_PATCH_VERSION 9
#define ADC_AR_RELEASE_MAJOR_VERSION 4
#define ADC_AR_RELEASE_MINOR_VERSION 0
#define ADC_AR_RELEASE_REVISION_VERSION 2
```

**Hint** There are similar rules to provide version information also in the BSWMD file: Rule [\[BSWMD\\_Impl\\_006\]](#) and [\[BSWMD\\_Impl\\_007\]](#). Please consider that the version information given in the module header is consistent to the version information given in the BSWMD file.

## Rule BSW\_VersionInfo\_005: Provision of GetVersionInfo Interface

### Instruction

**Scope:** Software based on an AUTOSAR SWS

Each AUTOSAR based BSW module shall provide a function <Module>\_GetVersionInfo to read out published parameters.

Each BSW module shall provide a function to read out the version information of a dedicated module implementation.

The prototype of this API is defined as followed:

```
void <Module>_GetVersionInfo(Std_VersionInfoType* VersionInfo);
```

The type of the parameter is defined in [\[CCode\\_Symbols\\_004\]](#). To implement this API identifier defined in [\[BSW\\_VersionInfo\\_002\]](#), [\[BSW\\_VersionInfo\\_003\]](#) and [\[BSW\\_VersionInfo\\_004\]](#) can be used.

Following example shows the implementation for module "Adc" based on AUTOSAR R4.0:

```
#include "Adc.h"
...

#define ADC_START_SEC_CODE
#include "Adc_MemMap.h"
void ADC_GetVersionInfo(Std_VersionInfoType* versionInfo)
{
```

```

versionInfo->vendorID = ADC_VENDOR_ID;
versionInfo->moduleID = ADC_MODULE_ID;
versionInfo->sw_major_version = ADC_SW_MAJOR_VERSION;
versionInfo->sw_minor_version = ADC_SW_MINOR_VERSION;
versionInfo->sw_patch_version = ADC_SW_PATCH_VERSION;
}
#define ADC_STOP_SEC_CODE
#include "Adc_MemMap.h"

```

## Rule BSW\_VersionInfo\_006: Usage of GetVersionInfo Interface

**Instruction** The function `<Module>_GetVersionInfo` shall not be called within any SW of an ECU (maybe to check if another module exists or to get the version of another module). These functions are only provided to be called from offline tools.

How the offline tools are working and how they call the `<Module>_GetVersionInfo` services is not finally specified from AUTOSAR. But there is a clear intention that these services are not called from any software which runs on an ECU.

## 3.5.9 BSW Scheduler and Exclusive Areas

### Rule BSW\_Sched\_001:

**Instruction** Direct access of interrupt suspend/resume functions is not allowed. Instead the AUTOSAR Standard Interfaces `SchM_Enter_<Module>_<Name>` and `SchM_Exit_<Module>_<Name>` shall be used.

When using exclusive areas, no implicit assumptions about task priority or scheduling context are allowed. Even at highest task priority the locks shall be present to protect against concurrent accesses on multicore systems. Exclusive areas entered in a function shall be exited before end of the function. Calls to `SchM_Enter` and `SchM_Exit` can be nested as long as exclusive areas are exited in the reverse order they were entered. Nesting is not allowed, if a "NoNest" exclusive area was entered.

According to AUTOSAR this lock API is provided by `SchM_xxx.h` via the *RTE*/SchM generator. However, the complexity of automatic lock configuration currently cannot be generated by *RTE*. Also spinlocks are not supported. Therefore the locks are provided by a separate configuration header named `<Module>_Cfg_SchM.h`. So, the integrator is responsible for configuration.

A description for the integrator has to be provided in `<Module>_Cfg_SchM.h`. Different tradeoffs between number of locks and maximum lock time can be supported by `<Module>_Cfg_SchM.h` and selected by integrator according to different timing needs. For e.g. DGS 2us lock time has to be supported.

Convention for locks provided by `<Module>_Cfg_SchM.h`:

- ▶ `SchM_Enter_<Module>_<Name>NoNest` – No other locks are allowed when this lock is set. Shall not be used if external functions are called. Reason: to improve parallelism for future systems by mapping to different NoNest locks.
- ▶ `SchM_Enter_<Module>_<Name>` – Nesting with other locks is allowed.

To support maximum backward compatibility the previous interfaces for interrupt locks are mapped to a common lock. This also applies for locks provided by AUTOSAR OS. For new AUTOSAR Software only the `SchM_Enter/Exit_<Module>_<Name>` APIs shall be used.

For efficient multicore support and migration the following new lock APIs are available for integrators:

```

extern void rba_BswSrv_GetLockCommon(void);
extern void rba_BswSrv_ReleaseLockCommon(void);

```

Implements busy waiting for "common" resource + interrupt lock on multicore systems. For single core systems it implements an interrupt lock. Nesting is allowed.

```
extern void rba_BswSrv_GetLockNoNest(rba_BswSrv_Lock_to* Lock_pst);
extern void rba_BswSrv_ReleaseLockNoNest(rba_BswSrv_Lock_to* Lock_pst);
```

Implements busy waiting for specified resource + interrupt lock on multicore systems. For single core systems it implements an interrupt lock. Nesting is not allowed.

```
extern void rba_BswSrv_SuspendCoreLocalInt(void);
extern void rba_BswSrv_ResumeCoreLocalInt(void);
```

Implements interrupt lock.

This APIs shall only be used in `<Module>_Cfg_SchM.h` or for mapping other APIs.

## 3.6 Rule Set: Style Guide

Handling of code is easier if the style of the code is harmonized. If the optical structure of the code is well-defined the main focus can be set to functionality of the code. It is then easier to understand the code and thus makes a review simpler. This rule set covers rules for style of code sequences, comments and documentation topics. The style guide is valid for C code and code of script languages.

**Hint** Templates for c and header files are available on the delivery folders of the coding guidelines. The templates contain all comment blocks which are defined in this rule set.

### 3.6.1 Common File Style

#### Rule CCode\_Style\_001: Unique Style

**Instruction** Each C file, header and script source file shall follow common settings to get an unique style.

This rule will ensure that a common set of file-base style guidelines are fulfilled.

#### Rule CCode\_Style\_002: Standard File Header

**Instruction** Every C and header source file shall be headed by a standard file header. Excluded are generated C and header files which are not stored in the SCM.

In [Figure 15](#) the file header for C files and header files is shown.

Figure 15 Module Header for C Files and Header Files

```
/*<BASDKey>
*****
*
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2014. All rights reserved.
* The reproduction, distribution and utilization of this document as well as the communication of its contents to
* others without explicit authorization is prohibited. Offenders will be held liable for the payment of damages.
* All rights reserved in the event of the grant of a patent, utility model or design.
*
*****
* Administrative Information (automatically filled in)
* $Domain____:$
* $Namespace__$
* $Class______$
* $Name______$
* $Variant____$
* $Revision__$
*****
</BASDKey>*/
```

The year in the file header represents the creation respectively the first release of the corresponding file. Therefore the year has not to be updated when an existing file is changed. But for new files the year shall be set to the current one.

Headers for other file types (script files, ARXML) are defined in the corresponding chapters and rule sets.

## Rule CCode\_Style\_003: Special Comment Blocks

**Instruction** To structure source files special comment blocks may be used to cluster header includes, definitions, declaration and code.

As shown in the following figure *Figure 16* the defined comment blocks will help to create a common file style for c and h files. The given order of the comment block shall be used. If a category is not needed the corresponding comment block shall be removed.

Figure 16 Comment blocks to structure source files

```
/*
*****
* Includes
*****
*/
...
/*
*****
* Defines/Macros
*****
*/
...
/*
*****
* Type definitions
*****
*/
...
/*
*****
* Variables
*****
*/
...
/*
*****
* Extern declarations
*****
*/
...
```

## Rule CCode\_Style\_004: Line Length

**Instruction** The line length shall not exceed 120 characters.

With a length of 120 characters a line can be displayed entirely on current monitors. Additionally a printout of the code fits perfectly on a sheet of paper.

## Rule CCode\_Style\_005: Indentation

**Instruction** Each new instruction block shall be indented by 4 whitespaces within C and header files. The "Tab" character (ASCII code 09) is not permitted in the code and in scripts. Instead of a tabulator 4 whitespaces have to be used.

The editor in the development environment shall be configured in such a way that a tabulator is replaced by 4 whitespaces.

## Rule CCode\_Style\_006: Definition of Variables

**Instruction** Only one variable shall be defined in a single line. A multi line or comma separated definition is not allowed.

Definitions of variables are located in C files and script files. This rule will help to keep lines with definitions short and clear. The usage of comma operator is generally restricted, see [\[CCode\\_Expr\\_010\]](#).

Example:

Not allowed:  
uint16 a\_ul6, c\_ul6;

Allowed:  
uint16 a\_ul6;  
uint16 c\_ul6;

## Rule CCode\_Style\_007: Initialization of Arrays and Structures

**Instruction** Braces shall be used to indicate and match the structure in a non-zero initialization of arrays and structures.

ISO C requires initializer lists for arrays, structures and union types to be enclosed in a single pair of braces (though the behaviour if this is not done is undefined). The rule given here improves ISO C by requiring the use of additional braces to indicate nested structures. This forces the programmer to explicitly consider and demonstrate the order in which elements of complex data types are initialised (e.g. multi-dimensional arrays). This principle applies to structures, nested combination of structures, arrays and other complex types.

The elements of arrays or structures can be initialized by giving an explicit initializer for the first element only. If this method of initialisation is chosen then the first element should be initialized to zero and nested braces need not be used. All other non-zero initialisation of arrays and structures shall require an explicit initializer for each element.

Examples:

```
uint16 x[3][2] = { 1, 2, 3, 4, 5, 6 };          /* Not OK, braces are missing */
uint16 y[3][2] = {{1, 2}, {3, 4}, {5, 6}};      /* OK */
uint16 z[3][2] = { 0 };                        /* OK, zero initialisation */
uint16 p[5]    = { 1, 2, 3 };                  /* Not OK, incomplete initialisation */
```

## Rule CCode\_Style\_008: Operators and Whitespaces

**Instruction** Mathematical, logical, comparison and conditional operators shall be embedded in whitespaces. This rule does not apply to brackets and operators like !, ++ and --.

This rule will ensure a better readability for mathematical and logical terms.

Examples:

```
result_ul6 = (a_u8 * b_u8) + c_ul6;

if (var1 && var2)
...

if (!var1)
...

result = (value1 >= value2) ? value1: value2;

result ^= value1;

result++;
```

## Rule CCode\_Style\_009: Nested Preprocessor Commands

**Instruction** Nested preprocessor commands should use a special form of indentation.

It could be confusing to use many preprocessor conditions which introduce new indentation levels. Each new level shall have an additional whitespace as indentation between the # and the command. The # character shall always be the first character of a pre-processor command line. Following example will show the principle.

```
#if (...)          <-- First level
...
```

```

05  # if (...)                <-- Second level (one whitespace)
    ...
    # ifdef (...)           <-- Third level (two whitespaces)
    ...
    # endif                <-- Third level (two whitespaces)

10  # else                  <-- Second level (one whitespace)
    ...
    # endif                <-- Second level (one whitespace)
    #endif                <-- First level

15  ^ # is on the same line

```

## Rule CCode\_Style\_010: Display History Information

**Instruction** To display history information from the SCM system a special comment block shall be placed at end of a file.

The SCM system BASD provides history information which shall be displayed in a C or header file. Because the history information could be long (because every version of a file can have its own history information) it shall be placed at the end of a file. To do that use the comment block shown in [Figure 17](#).

Figure 17 Comment block for SCM History information

```

/*<BASDKey>
*****
* $History__:$
*****
</BASDKey>*/

```

## Rule CCode\_Style\_011: Display Header File Name

**Instruction** The name of a header file should be named within a comment at the end of each header file.

For an easier handling of processed files the name of a header file should be placed at end of a header file with a special comment. This will help to find the end of a header in a processed file listing. Than it is clear from which point a header starts (can be identified by the module header of [\[CCode\\_Style\\_002\]](#)) and a header ends (identified by the comment of this rule).

Figure 18 Comment block for header name

```

/*<BASDKey>
*****
* End of header file: $Name____:$
*****
</BASDKey>*/

```

## Rule CCode\_Style\_012: New Line at End of File

**Instruction** Each c file and header shall end with a new line to avoid compiler warnings.

Some compilers are sensible in case that a c or header file ends with no new line. A compiler warning (e.g. "no newline at end of file") can occur. To avoid this problem a new line shall be placed explicitly.

## Rule CCode\_Style\_013: Avoiding of trailing spaces

**Instruction** Trailing spaces shall be avoided.

Trailing spaces are white spaces at the end of a line. These white spaces cannot be seen and have no technical effect. Some editors have the feature to suppress trailing spaces. If possible that feature shall be activated.



## 3.6.2 Block Style

### Rule CCode\_BlockStyle\_001: Common Block Style Layout

**Instruction** A block style layout shall be used for all functions and type definitions and all instruction blocks like for-loop, if-else, do-while, while and switch-case. The statement forming the body of an instruction shall be a compound statement.

An instruction block is a list of commands which are enclosed with curly brackets and which are in context of a function or type definition or language keywords like if-else, do-while, while and switch-case. To make the borders and the hierarchy of such blocks visible this rule and the following sub rules are specified. An 'else if' needs no own braces but the following else path needs to have a compound statement (see [\[CCode\\_CntrFlow\\_007\]](#)).

Examples:

<b>Function:</b> <pre>void MyModule_Func(void) {     ... }</pre>	<b>for loop:</b> <pre>for (I = 0; I &lt; 10; I++) {     ... }</pre>	<b>switch-case:</b> <pre>switch (var) {     case 1:     {         ...     }     break;      case 2:     {         ...     }     break;      default:     {         ...     }     break; }</pre>
<b>Typedef definition:</b> <pre>typedef struct {     uint8 a_u8;     uint8 b_u8; } MyStruct_tst;</pre>	<b>do-while:</b> <pre>do {     ... } while (a &gt; b);</pre>	
<b>if-else:</b> <pre>if (a &gt; b) {     ... } else if (b &gt; c) {     ... } else {     ... }</pre>	<b>while:</b> <pre>while (a &gt; b) {     ... }</pre>	

### Rule CCode\_BlockStyle\_002: Block Brackets are Single Characters in a Line

**Instruction** The beginning of a block "{" and the end of a block "}" shall be at a new line and shall be the only character in this line (Except the character indicates the end of a structure definition or is followed by a while command. Here the name of the structure or the while command can stand after the end of block bracket. Additional comments are allowed.) After a block a new line should be entered (Except the block ends before an else or break command.).

Example:

```
void MyModule_Func(void)
{
    <-- Single character in a new line (or a comment is allowed)
    /* Instructions */
}
<-- Single character in a new line (or a comment is allowed)
<-- New line after block
```

It is not allowed to write it in following way:

```
void MyModule_Func(void) {
```

```
/* Instructions */
}
```

### Rule CCode\_BlockStyle\_003: Horizontal Position of Brackets of Blocks

**Instruction** "{" and "}" shall be placed at the same horizontal position.

Example:

```
void MyModule_Func(void)
{
    /* Instructions */
}
```

*^ Same horizontal position*

### Rule CCode\_BlockStyle\_004: Instruction Block with One Instruction is a Block

**Instruction** If an instruction block only contains one instruction, the block shall nevertheless be embraced with "{" and "}".

Example:

```
if (a > b)
{
    a = b;    <-- Only one instruction is still enclosed with curly brackets.
}
```

It is not allowed to write it in following way:

```
if (a > b) { a = b }
```

## 3.6.3 Comments

### Rule CCode\_Comments\_001: Usage of Comments

**Instruction** Code which is not self-explanatory shall be extended by a meaningful comment. Also workarounds and rule violations shall be commented, too.

Comments are necessary to understand a section of code. But they should not be a novel, a comment is only helpful if it is precise and easily understandable. All code sequences which are not obviously self-explaining shall be commented generally to help others to understand the code. Backgrounds, properties, assumptions and invariant sections shall be explained by a helpful comment too.

The developer shall not blindly repeat what the code says. Such comments can be deleted. A comment should be essential and meaningful.

### Rule CCode\_Comments\_002: Allowed Comment Marker

**Instruction** It is allowed to use "/\* ... \*/" or "///" as comment marker styles to write a comment.

/\* ... \*/ is the standard comment style defined in ISO C90 standard. // is an extension but it is supported by most compilers.

### Rule CCode\_Comments\_003: Non-Nested Comments

**Instruction** Comments shall not be nested and comment styles shall not be mixed.

C does not support the nesting of comments even though some compilers support this as a language extension. A comment begins with `/*` and continues until the first `*/` is encountered. Any `/*` or `//` occurring inside such a comment is a violation. Single line comments starting with `//` shall not contain additional `//` comment styles or sequences of `/* ... */`.

## Rule CCode\_Comments\_004: Sections of Code Should Not be Commented Out

**Instruction** Sections of code should not be commented out.

Where it is required for sections of source code not to be compiled then this should be achieved by use of conditional compilation (e.g. `#if` or `#ifdef` construct with a comment). Using comment markers for this purpose is dangerous because any comment already existing in the section of code would create a nested comment which is not allowed (see [CCode\_Comments\_003]).

## Rule CCode\_Comments\_005: Language of Comments is English

**Instruction** Comments shall be written in English.

English is the preferred language for all code documentation. English words shall be used in their native meaning, a dictionary or domain textbook can be consulted if necessary.

## Rule CCode\_Comments\_006: Position of Comments

**Instruction** It is allowed to write a comment behind code but it is not allowed to write code behind a comment.

This rule supports also the look and the readability of code. Additional comments behind several code lines should start at the same horizontal position to get a nice look. If a comment is written before a line of code it should use the same alignment as the code line. Then it is easy to recognize to which code line a comment belongs.

Examples:

```
/* Not allowed example comment */ valueC = valueA + valueB;

valueC = valueA + valueB;    /* Allowed example comment */
valueD = valueC;             /* Allowed example comment */

if (valueD > 0)
{
    /* Do something */  <-- comment has same alignment as the next code line
    valueD = 0;
    ...
}
```

## Rule CCode\_Comments\_007: Markers for Requirements Tracing

**Instruction** A special marker may be used within comments to trace requirements from AUTOSAR: `TRACE[<AUTOSAR SWS Trace ID>]`.

Stating of trace elements from the AUTOSAR specs in the source code is allowed (but not mandatory). This should be made in a consistent way, and additionally it should be possible to use Eclipse's task tag feature to display all those trace references in the source code. If the markers are used following suggested format shall be used: `TRACE[<AUTOSAR SWS Trace ID>]`.

Example (Flash driver, AUTOSAR\_SWS\_FlashDriver):

```
/* TRACE[FLS165] Implementation of the Fls_GetVersionInfo interface. */
```

## Rule CCode\_Comments\_008: Publishable Contents of Comments

**Instruction** All contents of comments shall be publishable. That means that comments ...

- ▶ shall not contain customer names, product names or customer specific wordings
- ▶ shall not contain names of persons, names of departments or Bosch specific wordings
- ▶ shall not contain four-letter words or other inappropriate phrases like "dirty hack", "bug", "quick and dirty" or "this can be a reason for a big recall"

The reason for that rule is the publication of CUBAS software in context of freeCUBAS / COMASSO. Because it is often not probable at the time of software creation that software will be published. Therefore this rule shall be kept in advance.

## 4 Rule Set: ECU Configuration

### 4.1 Introduction to ECUConfig

AUTOSAR ECU configuration defines a concept of code generation based on abstract configuration values. The following issues are addressed by this concept:

- ▶ Allow to define configuration parameters on a higher abstraction level compared to the resulting generated configuration code
- ▶ Facilitate greatly enhanced validation possibilities compared to configuration via C code
- ▶ Define uniform configuration formats, even if the configured SW components are created by different vendors (e.g. Bosch, Conti, Elektrobit, Etas, Vector)

The term ECUC has also been chosen to separate this concept from other configuration concepts such as CM (Configuration Management) or variant mechanisms such as configuration via system constants.

In a very general view, there are four categories of artifacts in the realm of ECU configuration:

- ▶ **ECU Configuration Values:** specifies (typically project-specific) configuration settings, e.g. length of CAN messages, resolution of analog input channels, or the frequency of PWM outputs. From the process perspective, creating or changing ECU configuration values is almost equivalent to creating or changing code.
- ▶ **ECU Configuration Parameter Definition:** defines the structure and constraints of configuration settings, e.g. that the A/D converter supports only the resolutions 8 bit, 10 bit, and 12 bit. This artifact defines the configuration interface of the corresponding SW component.
- ▶ **ECU Configuration Processor:** consists of configuration value checkers and code generators which transform the abstract configuration values into code which implements this configuration information on the ECU. Additionally, some ECU configuration processors also contain so called "forwarders" which generate dependant ECU configuration values from their input ECU configuration values.
- ▶ **Build Action Manifest:** specifies the interface of the ECU configuration processor to the configuration framework which controls the processing of ECUC values (see also chapter 4.1.2 ).

Each SW component falls into one of the following three categories with respect to ECU configuration:

- ▶ **Not affected:** The SW component neither contains ECUC values nor ECUC parameter definitions or configuration processors.
- ▶ **Configuring:** The SW component contains ECUC values, but does not contain ECUC parameter definitions or configuration processors.
- ▶ **Configurable:** The SW component contains ECUC values as well as ECUC parameter definitions and a configuration processor.

In the AUTOSAR ECUC methodology, only BSW components are defined to be configurable<sup>1</sup>. Because there is no composition concept for BSW in AUTOSAR, only atomic BSW components (also known as BSW modules) can be configurable.

Each configurable SW component shall provide configuration documentation which describes all available configuration parameters. This documentation is part of the ECUC parameter definition files and can be viewed during editing with the appropriate configuration editors.

<sup>1</sup> Applying the ECUC methodology to application software or complex drivers is explicitly allowed by AUTOSAR, however.

## 4.1.1 Authoring of ECUC artifacts

For each of the types of ECU Configuration artifacts defined above, the following chapters define the procedures for authoring activities (i.e. creation and maintenance) of these artifacts. These procedure definitions provide all conventions and best practices to be followed during the authoring activities.

### Rule ECUC\_001: AUTOSAR conformance

#### Instruction

Within BSW, only the ECUC formats defined by AUTOSAR are allowed to be used. In particular, MSR Conf is not to be used within BSW (but is allowed in legacy application software and adapters to legacy software).

All AUTOSAR ECUC artifacts shall be conforming to the corresponding AUTOSAR specifications.

### Rule ECUC\_002: AUTOSAR version

#### Instruction

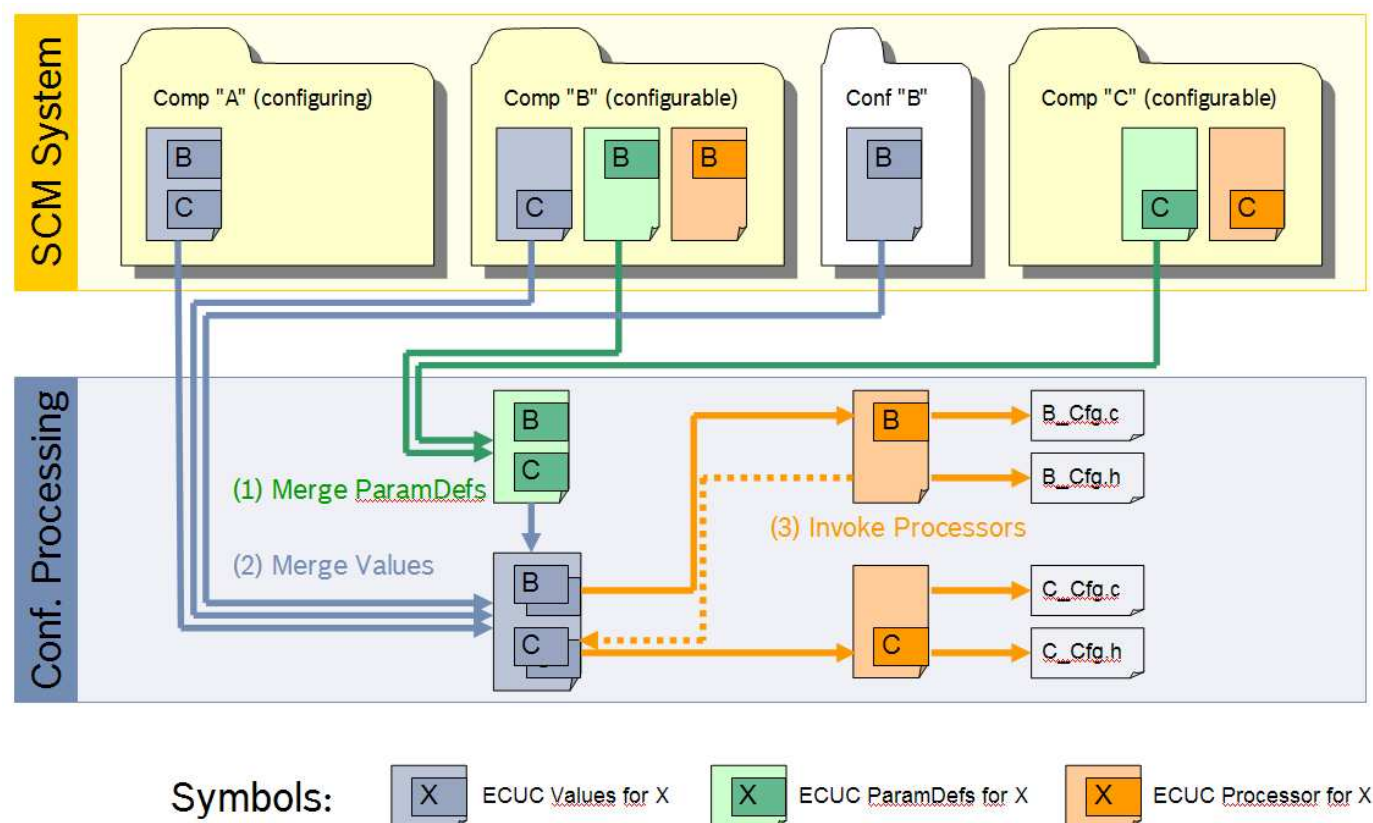
On the head of the main development branch, all AUTOSAR ECUC artifacts shall be based on AUTOSAR release 4.0.2 or higher.

## 4.1.2 How ECUC Values Are Processed

All ECUC values are transformed to the corresponding code by ECUC processors which are delivered together with the configurable SW components. Configuration processing always takes place in project context as depicted in figure 19. In this figure, the code generated e.g. by SW component B is shown as B\_Cfg.c and B\_Cfg.h, but in reality often many more files (e.g. reports or meta data) are generated by the ECUC processor<sup>2</sup> of a SW component. Forwarding of ECUC values from SW component B to SW component C is shown as a dotted line from B's ECUC processor to the merged ECUC values of C.

<sup>2</sup> In some AUTOSAR documents, ECUC processors are also called "BSW module generators" or simply "configuration tools".

Figure 19 Processing of ECUC Values



Whenever possible, configuration processing is part of the SW build, and the output of configuration processing is not stored in the SCM system (just like object code generated by compilers which is also not stored in the SCM system for the same reasons). This processing concept is also known as **"online configuration"** as opposed to **"offline configuration"** (where configuration processing takes place e.g. in a configuration editor and the results are stored in the SCM system). There may be circumstances, however, which prohibit the online configuration approach, e.g. some SW sharing scenarios or situations where the generated files shall be reviewed.

## 4.2 ECUC Values

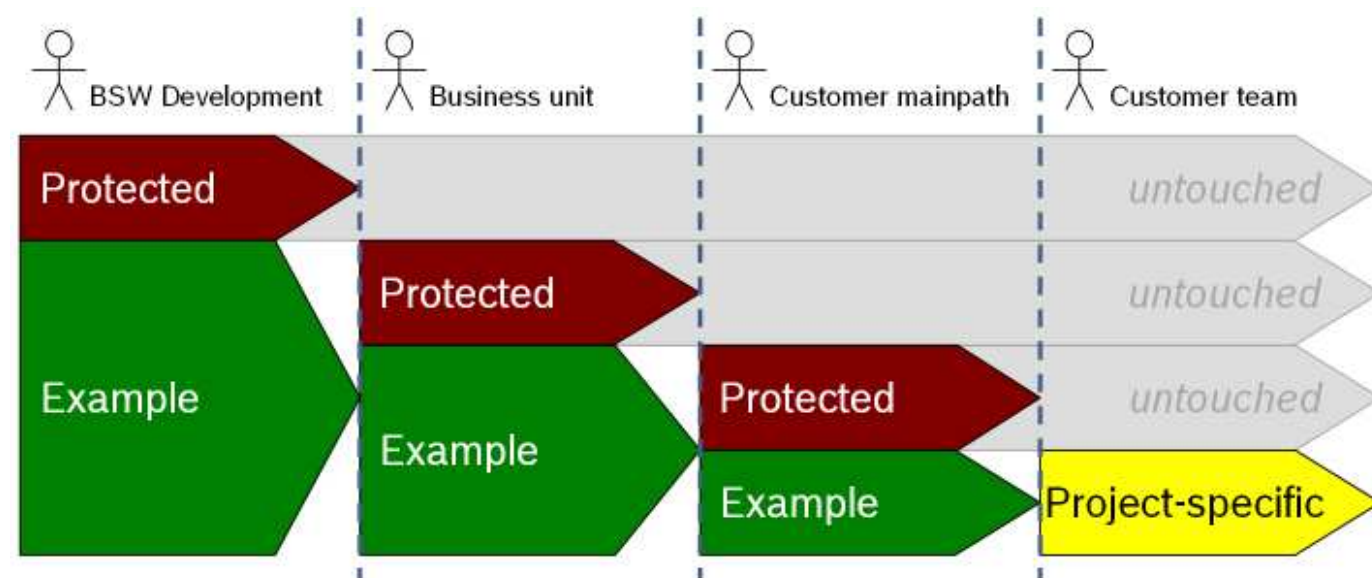
### 4.2.1 ECUC Values: Content Responsibility

By nature, ECUC values are generally project-specific. In consequence, the customer projects are generally responsible for these values and hence also for the delivery of the files which contain them.

In some cases, however, customer projects may delegate responsibility for ECUC values (see also figure 20):

- ▶ If some configuration aspects are identified to be identical for **all** customer projects **regardless of the business unit**, then these aspects are separated into **protected ECUC values** files. This protected configuration content is delivered together with the SW components which are most closely related to this content.
- ▶ If some configuration aspects are identified to be identical for **all** customer projects **within a specific business unit**, then these aspects are also separated into **protected ECUC values** files. This protected configuration content is delivered by business unit responsables.
- ▶ This concept may be generalized for even longer delivery chains: each contributor along the delivery chain freezes all configuration values which are identical for **all of his customers** and puts them into **protected ECUC values** files.

Figure 20 Distribution of ECUC Values



SW components which are configurable via the ECU configuration concept shall also deliver (non-protected) **example ECUC values**. The only exception are cases where this example file would only contain ECUC values which are identical to their default values. No customer project may take over these example values without the full set of quality measures (check, review, test) executed in their specific project context as described below. The responsibilities of configurable SW components shall formally check their example ECUC values in an arbitrarily chosen project context, but no other quality measures are prescribed for these example files because the results of this effort could not be re-used by the customer projects.

### Rule ECUC\_V001: Content responsibility

#### Instruction

Customer projects are responsible for ECUC values files unless this responsibility has explicitly been delegated.

Project-specific ECUC values files are named `<Comp>_EcucValues.arxml`<sup>3</sup>.

### Rule ECUC\_V002: Protected configuration

#### Instruction

Protected ECUC values shall not be changed by any developer who receives these files as a deliverable<sup>4</sup>.

Protected ECUC values files are named `<Comp>_Prot_EcucValues.arxml`.

### Rule ECUC\_V003: Configuration examples

#### Instruction

Configurable SW components shall also deliver example ECUC values files (except where these files would only contain values which are identical to their default values).

Example ECUC values files are named `<Comp>_EcucValues.arxml`.

<sup>3</sup> See also the guideline on file naming rules for an overview of all file naming conventions.

<sup>4</sup> Protected ECUC values are a concept which go beyond of what is currently defined by AUTOSAR itself. But this concept does also not violate any definition imposed by AUTOSAR.



## Rule ECUC\_V004: Release conditions

### Instruction

ECUC values may only be released if they have been successfully checked, reviewed, and tested (see below for details).  
ECUC value examples need only be formally checked but not tested.

## 4.2.2 ECUC Values: Editing, Checking, and Reviewing

ECUC values are to be edited with BCT which is part of the SW development IDE. You may only fall back to generic text or XML editors if an ECUC Values file is corrupted up to a level where fixing this error with BCT would be more time-consuming or even impossible. BCT requires ECUC parameter definitions for properly presenting and validating entered configuration values. These parameter definitions are only available in project context, so editing of configuration values always requires such a project context. This is not only a tool-related requirement, but is also necessary from process perspective because a SW component with the need for ECUC values can only be compiled and tested inside a project context (otherwise the code generated from the ECUC values is not available and hence compilation and test is not possible).

### Rule ECUC\_V005: Editing

#### Instruction

ECUC values shall be edited with BCT.

### Rule ECUC\_V006: Formal checks

#### Instruction

ECUC values have to be formally checked before delivery by processing it in a suitable project context.

This check is automatically performed by the ECU configuration framework (typically invoked from SW build) and the ECUC processors. Only ECUC values which can be processed error-free in such a context are allowed to be released. Any warnings raised during processing shall be assessed before a release.

### Rule ECUC\_V007: Reviewing

#### Instruction

Protected ECUC values shall be reviewed before releasing the SW component version which contains these values.  
Project-specific ECUC values shall be reviewed by a project expert before releasing the project version which contains these values.

## 4.2.3 ECUC Values: Testing

### Rule ECUC\_V008: Testing

#### Instruction

Protected ECUC values shall be tested before SW component delivery as part of the unit test for this SW component.  
Project-specific ECUC values are tested in the context of the corresponding project.

For details on project-specific tests, please refer to the corresponding process descriptions because the related processes are business unit specific and typically there are several test types addressing dedicated functionality aspects of the entire ECU SW.

## 4.2.4 ECUC Values: Content-related Procedures

### Rule ECUC\_V010: ArPackage usage

#### Instruction

All ECU configuration values shall be within the ArPackage hierarchy

**/RB/UBK/Project/EcucModuleConfigurationValues.**

See the guideline rules for ArPackage usage for details.

### Rule ECUC\_V011: EcucValueCollection usage

#### Instruction

No BSW module specific ECUC values file shall contain a **<ECUC-VALUE-COLLECTION>**. Not more than one centrally maintained ECUC values file per project shall contain this collection.

### Rule ECUC\_V009: Container short names

#### Instruction

The short name of each container in the ECUC values files shall be identical to the short name of the corresponding container definition **if** it is not allowed to exist more than once. E.g., the short name of the "DioGeneral" container shall be "DioGeneral". No ECUC processor may rely on this convention, though.

For containers which may exist more than once according to the ECUC ParamDef (i.e. UpperMultiplicity > 1), no general conventions apply.

This convention is only for removing a major ambiguity inherent to the AUTOSAR specification regarding ECUC value containers.

## 4.3 ECUC Parameter Definitions

### 4.3.1 ECUC ParamDefs: Content Responsibility

#### Rule ECUC\_PD001: Content responsibility

#### Instruction

ECUC parameter definitions are in the sole responsibility of the developer whose SW component is configurable by means of the ECU configuration methodology. They shall be named **<Comp>\_EcucParamDef.xml**.

### 4.3.2 ECUC ParamDefs: Editing, Checking, and Reviewing

Currently, there is no dedicated editing support for ECUC parameter definitions available. Hence, any XML or text editor may be used. You may also try out the [\[Document Excel-based ParamDef generator / URL: file:///bosch.com/dfsrb/Dfs-DE/DIV/CDG/Prj/CUBAS/70\\_EcuConfiguration/80\\_Tools/Converters/Xls2EcucParamDef\]](file:///bosch.com/dfsrb/Dfs-DE/DIV/CDG/Prj/CUBAS/70_EcuConfiguration/80_Tools/Converters/Xls2EcucParamDef). Additionally, a generic editor for many types of AUTOSAR XML documents is available ("Axe") which is also usable for editing ECUC parameter definitions.

#### Rule ECUC\_PD002: Editing

#### Instruction

ECUC parameter definitions shall be changed in a backwards-compatible way whenever possible. Incompatible changes shall be documented.

Since ECUC parameter definitions define the ECUC "interface" of the corresponding SW component, the author of these definitions shall make sure that any changes are backwards compatible with existing ECUC values. This typically means that

- ▶ no existing parameter definition may be removed and that
- ▶ new parameter definitions are made optional.

If backwards compatibility cannot be maintained or is mutually agreed to be given up, this shall be documented in the release notes of the SW component which owns these parameter definitions.

### Rule ECUC\_PD003: Formal checks

#### Instruction

ECUC parameter definitions shall be formally checked for validity against the corresponding version of the AUTOSAR schema before delivery.

Dedicated checker tools are not available for ECUC parameter definitions, but a formal validation against the related AUTOSAR XML Schema shall take place before delivery. This can be done by any validating XML editor. With the standard Eclipse XML editor, the procedure is as follows:

1. Open the file to be validated with the standard Eclipse XML editor (Open With: XML Editor)
2. In the XML editor, select the "Design" page (on the bottom of the editor)
3. Make sure that XML Schema assistance is available by checking that possible content is shown for each XML element (in pale green, e.g. (ADMIN-DATA?, INTRODUCTION?, AR-PACKAGES?) for the top level AUTOSAR element)
4. Select the "Source" page (again on the bottom of the editor)
5. Make sure that no error markers are shown in the overview ruler (which is located to the right of the vertical scroll bar)

### Rule ECUC\_PD004: Reviewing

#### Instruction

After a successful formal check, the parameter definitions shall be reviewed against the requirements which triggered their creation or update.

## 4.3.3 ECUC ParamDefs: Testing

The reviewing of ECUC parameter definitions is currently regarded as being a sufficient quality measure. Hence, no dedicated tests are prescribed for these artifacts.

## 4.3.4 ECUC ParamDefs: Content-related Procedures

### Rule ECUC\_PD005: Vendor-specific derivation

#### Instruction

If the AUTOSAR standard specifies ECUC parameters for the particular SW component, then the ECUC parameter definition file for this SW component shall adhere to the "vendor-specific derivation rules" provided in the AUTOSAR ECUC specification in chapter 5.

In particular, the following derivation rules shall get special attention:

- ▶ Vendor-specific ECUC parameter definition files shall contain a complete tree of parameter definitions, not only the add-ons to parameters already specified by AUTOSAR. In contrast to the parameter definitions delivered by AUTOSAR itself, there is no single huge vendor-specific ECUC parameter definition file but rather a single parameter definition file for each configurable SW component.
- ▶ Standardized parameters which are not supported by a module implementation shall not be removed from the vendor-specific ECUC parameter definition but shall rather be set to a multiplicity range of [0; 0]. In cases where this is not allowed due to chapter 5.1 in the ECUC spec, this deviation shall be documented in the corresponding chapter of the module documentation.

### Rule ECUC\_PD006: One module definition per file

#### Instruction

No ECUC parameter definition file shall contain the definition of more than one **<ECUC-MODULE-DEF>**.

### Rule ECUC\_PD007: English language

#### Instruction

The only language to be used is English.

### Rule ECUC\_PD014: ArPackage usage

#### Instruction

All ECUC parameter definitions shall be either within the ArPackage hierarchy

**/AUTOSAR\_<Module>/EcucModuleDefs** (for BSW components where AUTOSAR defines a standardized ECUC parameter definition) or

**/RB/RBA/<Module>/EcucModuleDefs** (for all other BSW components).

See the guideline rules for ArPackage usage for details.

### Rule ECUC\_PD008: EcucDefinitionCollection usage

#### Instruction

No BSW module specific ECUC parameter definition file shall contain a **<ECUC-DEFINITION-COLLECTION>**. Not more than one centrally maintained ECUC parameter definition file per project shall contain this collection.

### Rule ECUC\_PD009: Naming conventions for short names

#### Instruction

The contents of the **<SHORT-NAME>** of each parameter definition (incl. containers and references) shall adhere to the AUTOSAR naming convention of ECUC parameters: upper camel case, starting with the owning SW component's name (e.g. "DioGeneral"). If the component's name contains underscores, the actual parameter name shall be separated from the SW component name by another underscore (e.g. "rba\_loSigDio\_General"). Bosch-specific parameters in SW components specified by AUTOSAR shall be marked by an "Rb" infix directly after the SW component name, e.g. "DioRbLegacyApi".

### Rule ECUC\_PD010: Order of parameters

#### Instruction

The **<SHORT-NAME>**s of all items inside a container shall be in the following order:

1. all parameters in alphabetical order
2. all references in alphabetical order
3. all sub-containers in alphabetical order.

## Rule ECUC\_PD011: Long name of parameters

### Instruction

Each ECUC parameter (incl. containers and references) shall provide a human-readable **<LONG-NAME>**. This is typically used as a display text in configuration editors. If providing a unit is helpful for the user, then it shall be appended behind the actual parameter long name, e.g. "Maximum delay [us]".

## Rule ECUC\_PD012: Description of parameters

### Instruction

Each parameter (incl. containers and references) shall provide a proper description of this parameter in the **<DESC>** element. If one paragraph of description is not sufficient, it shall be continued in the **<INTRODUCTION>** element.

## Rule ECUC\_PD013: Specification of parameter origin

### Instruction

The **<ORIGIN>** of any parameter not defined by AUTOSAR shall be set to "RB", possibly followed by a version and/or date specifier.

Example of a properly defined parameter:

```
<ECUC-ENUMERATION-PARAM-DEF>
  <SHORT-NAME>rba_IoSigDio_InitState</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Init state</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">
      State of this signal after initialization.
      This setting is only relevant for output signals.
    </L-2>
  </DESC>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <IMPLEMENTATION-CONFIG-CLASSES>
    <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
      <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
      <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
    </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
  </IMPLEMENTATION-CONFIG-CLASSES>
  <ORIGIN>RB:0.1.0:2011-05-09</ORIGIN>
  <SYMBOLIC-NAME-VALUE>>false</SYMBOLIC-NAME-VALUE>
  <DEFAULT-VALUE>Idle</DEFAULT-VALUE>
  <LITERALS>
    <ECUC-ENUMERATION-LITERAL-DEF>
      <SHORT-NAME>Idle</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Idle</L-4>
      </LONG-NAME>
      <ORIGIN>RB:0.1.0:2012-01-16</ORIGIN>
    </ECUC-ENUMERATION-LITERAL-DEF>
  </ECUC-ENUMERATION-LITERAL-DEF>
```

```
<SHORT-NAME>Active</SHORT-NAME>
<LONG-NAME>
  <L-4 L="EN">Active</L-4>
</LONG-NAME>
<ORIGIN>RB:0.1.0:2012-01-16</ORIGIN>
</ECUC-ENUMERATION-LITERAL-DEF>
</LITERALS>
</ECUC-ENUMERATION-PARAM-DEF>
```

## Rule ECUC\_PD016: Multiplicity handling

### Instruction

If the multiplicities of a standardized ECUC parameter shall be changed in a vendor-specific derivation, the following rules apply:

- ▶ The multiplicity interval defined in a vendor-specific ECUC parameter definition can be made smaller than its standardized counterpart without violating the standard as long as the standardized interval limits are not crossed. Nevertheless, this fact shall be documented as a deviation from the AUTOSAR specification.
- ▶ If the multiplicity interval defined in a vendor-specific ECUC parameter definition goes beyond at least one interval limit defined in the standardized ECUC parameter definition, then this vendor-specific derivation violates the AUTOSAR standard. As such, it must be justified and well documented if it is not avoidable.

A typical example of the first case is if a parameter which has been defined as optional in AUTOSAR (0..1) shall be made mandatory (1..1) in the vendor-specific version of the ECUC parameter definition.

A typical example of the second case is if a parameter which has been defined as mandatory in AUTOSAR (1..1) shall be made optional (0..1) in the vendor-specific version of the ECUC parameter definition.

## Rule ECUC\_PD017: Default values for mandatory parameters

### Instruction

Default values for mandatory parameters are a mere editing aid with the semantics of a "typical value". The existence of a default value for a parameter does not make this parameter implicitly optional.

## Rule ECUC\_PD015: Anticipation of parameters from future AUTOSAR releases

Sometimes it is necessary to introduce an ECUC parameter which does not exist in the currently supported AUTOSAR release yet, but a later AUTOSAR release already defines this parameter. With a strict interpretation of the AUTOSAR standard, this parameter would have to be defined as a purely vendor-specific parameter. According to rule [\[ECUC\\_PD009\]](#), this anticipated parameter would then require to have a vendor-specific name infix, e.g. NvMRbBlockUseSetRamBlockStatus (instead of NvMBlockUseSetRamBlockStatus as defined by the later AUTOSAR release). Once the affected BSW module is promoted to this later AUTOSAR release, however, two parameters with the same meaning would exist then. To address this issue, an exception is defined for this special case:

### Instruction

If an ECUC parameter shall be anticipated from a future AUTOSAR release, then the short name of this parameter shall be taken exactly as defined by this future AUTOSAR release. The **<ORIGIN>** of this parameter shall also be set to AUTOSAR\_ECUC, not to a vendor-specific string. The BSW module's documentation shall mention this parameter anticipation as a temporary deviation from the AUTOSAR standard (until the entire BSW module is upgraded to the future AUTOSAR release which defines this parameter). Other rules for deriving vendor-specific ECUC parameters from standardized ECUC parameters still apply.

## 4.4 ECUC Processors

### 4.4.1 ECUC Processors: Content Responsibility and Language

#### Rule ECUC\_P001: Content responsibility

##### Instruction

ECUC processors and the related helper files (libraries, templates, etc.) are in the sole responsibility of the SW developer whose SW component is configurable by means of the ECU configuration methodology.

#### Rule ECUC\_P002: Language

##### Instruction

The only allowed languages for ECUC processors are oAW (openArchitectureWare) and Perl. No other languages are allowed.

The preferred language is oAW. Inside CDG, this language is to be used for all SW components for which no explicit allowance of using Perl has been defined by CDG management. This exception is granted for all SW components belonging to:

- ▶ MemStack
- ▶ IoStack
- ▶ MCAL layer except MCAL SW components belonging to the ComStack
- ▶ Calibration Software
- ▶ ECUSec
- ▶ Boot Control
- ▶ BSW Library.

CDG packages shall not contain a mix of ECUC processor languages.

### 4.4.2 ECUC Processors: Editing, Checking, and Reviewing

No tools are prescribed for ECUC processor editing or checking.

#### Rule ECUC\_P003: Reviewing

##### Instruction

Before delivery, ECUC processors shall be reviewed against the corresponding requirements and existing content--related procedures (see below).

### 4.4.3 ECUC Processors: Testing

#### Rule ECUC\_P004: Testing

##### Instruction

ECUC processors are tested as part of the SW component's unit test. Ideally, all branches of the ECUC processors should be covered by these tests.

It shall be tested that correct ECUC value input produces correct ECUC processor output (including logs, reports, and ECUC processor messages) according to the requirements.

To address missing checks for existence of optional parameters, there shall be at least one test case where all optional parameters are not existing in the ECUC values.

## 4.4.4 ECUC Processors: Documentation

### Rule ECUC\_P005: Documenting

#### Instruction

Apart from line-oriented documentation inside the ECUC processor code itself, also the overall ECUC processor concept shall be documented unless it is a straightforward 1:1 implementation of the AUTOSAR specification of the related BSW module.

This documentation is part of the module's concept/design documentation.

## 4.4.5 ECUC Processors: General Content-related Procedures

### Rule ECUC\_P006: Allowed output

#### Instruction

Only artifacts of the own component (this includes AUTOSAR interface definitions) are allowed to be generated. Exceptions from this rule must be approved before implementation and they must be documented by the process responsible for ECU configuration<sup>5</sup>.

In addition to code artifacts and meta information (such as fragments of BSW module descriptions or SW component descriptions), only documentation fragments, configuration reports and configuration exports may be generated.

For generated code and meta information, the same quality demands and coding guidelines apply as for manually created code.

### Rule ECUC\_P007: Stable output

#### Instruction

In order to provide reproducible output and to allow the easy comparison of the files generated by an ECUC processor, the following rules shall be obeyed:

- ▶ Generated files are not allowed to contain path, date, time, or user information.
- ▶ No ECUC processor may rely on a particular order of non-ordered source data (in AUTOSAR, all multiple instances of the same container, parameter, or reference are non-ordered). It may even happen that the order of these elements as seen by the ECUC processor varies from processor run to processor run, even with unchanged source data. The ECUC processor shall establish its own reproducible order in these cases, e.g. by alphabetical sorting of the container instances by the short name.
- ▶ Generated files need not contain any SCM header because they are typically not checked in.

<sup>5</sup> Within CDG, the only approved exception are the components rba\_Wdg\* which are allowed to generate artifacts of components Wdg\_6\*.



## Rule ECUC\_P008: Allowed input

### Instruction

Main input of each ECUC processor shall be the ECUC values belonging to the ECUC ParamDefs delivered in the same SW component as the ECUC processor.

Additional input from other SW components is allowed if and only if this is technically necessary. In case of ECUC parameters standardized by AUTOSAR, only parameters marked with scope "global" or "ECU" in the AUTOSAR specification of these other SW components are allowed to be taken as additional input.

Due to maintenance issues, a global configuration values section (also known as GLOBDATA) is not allowed.

## Rule ECUC\_P009: Independence from input origin

### Instruction

ECUC processors are not allowed to react differently in case of

- ▶ manually created input ECUC values resp.
- ▶ forwarded ECUC values.

## Rule ECUC\_P010: Independence from container short names in ECUC Values

### Instruction

The short names of containers in ECUC Values shall only be used for identification purposes. No actual SW functionality may be influenced depending on short name contents except for the creation of #defines corresponding to ECUC parameters with the SymbolicNameValue attribute set to true.

## Rule ECUC\_P011: (Removed)

### Instruction

Removed.

## Rule ECUC\_P012: Naming conventions for input files

### Instruction

There shall be a clearly evident relation from input templates to the corresponding output files. E.g., if a generated file is called Hugo\_Cfg.h, then

- ▶ the corresponding oAW template shall be called Hugo\_Cfg\_h.xpt and
- ▶ the corresponding Perl template shall be called Hugo\_Cfg.ht.

## Rule ECUC\_P013: Naming conventions for output files

### Instruction

The following naming conventions apply for files generated by ECUC processors:

- ▶ Files containing PreCompile-time information shall be named `<Comp>_Cfg[<Sub>].<ext>`
- ▶ Files containing link time information shall be named `<Comp>_Lcg[<Sub>].<ext>`

- ▶ Files containing PostBuild-time information shall be named `<Comp>_PBcfg[_<Sub>].<ext>`
- ▶ Files defining AUTOSAR interfaces to ASW shall be named `<Comp>_Cfg[_<Sub>]_SWCD.arxml`
- ▶ Files defining (fragments of) BSW module descriptions shall be named `<Comp>_Cfg[_<Sub>]_BSWMD.arxml`
- ▶ Report files shall be named `<Comp>_Report.txt`
- ▶ Export files shall be named `<Comp>_Export.xml`

Please note that the `_auto_` infix in generated names is obsolete and shall not be used for AUTOSAR components anymore.

## Rule ECUC\_P014: Clear separation of actions

### Instruction

ECUC processors fulfil the following tasks (not necessarily all of them):

- ▶ Validation: checks input ECUC values for potential semantic problems.
- ▶ File generation: creates output files depending on input ECUC values.
- ▶ Forwarding: creates output ECUC values depending on input ECUC values.

These aspects shall be clearly separated in different processor actions. In particular, forwarders and generators shall not be mixed in one processor action. In oAW, also the validators shall be separated from the forwarders and generators. In Perl, separating the validators is often not a feasible approach, and hence validation is typically part of the forwarder and/or generator actions.

## Rule ECUC\_P015: Validation responsibility

### Instruction

No assumption about the correctness of input ECUC values shall be made in the ECUC processors. If a forwarder or generator action only produces valid output if some conditions on the input side are met, then the fulfillment of these conditions must be verified before actually forwarding or generating output in this action.

To meet the general single maintenance goals, also each validation code shall only be written once if technically feasible. In particular, validations centrally carried out by the configuration framework shall not be repeated in SW component-specific code.

## Rule ECUC\_P016: Problem reporting

### Instruction

Problem reporting in ECUC processors shall meet the following expectations:

- ▶ If no error is reported, then the configured SW component works according to its specification.
- ▶ Suspicious configurations which are typically caused by wrong ECUC values but still result in error-free operation (see above) shall trigger a warning.
- ▶ Merely informational items shall be provided via report files and logs only.

## Rule ECUC\_P017: Logging

### Instruction

Logging shall only be used for analyzing the trace of operations in the ECUC processors. The intended audience of

log files are the developers of ECUC processors, not ECUC users. Hence, customer-relevant information shall not be logged but issued via reports and/or generated documentation instead.

## Rule ECUC\_P018: Generating configuration documentation and reports

### Instruction

Currently, the effective configuration of a SW component is documented by plain text files (aka report files). No report-specific guidelines are defined yet.

Eventually, these reports will be replaced with generated documentation fragments according to the standard AUTOSAR documentation concepts.

## Rule ECUC\_P019: Handling of PostBuild data sets

### Instruction

The following conventions apply for multiple configuration containers (used for post-build selectable configuration, marked as **<MULTIPLE-CONFIGURATION-CONTAINER>** set to *true* in the ECUC parameter definition):

#### ► Configuration:

- The short name of the multiple configuration container shall start with the name of the configuration container itself plus an optional suffix identifying the data set, separated by an underscore. For example, if the multiple configuration container for SW component "Hugo" is called "HugoConfig", then "HugoConfig" or "HugoConfig\_4Cyl" or "HugoConfig\_8Cyl" are valid short names of these container values while "Hugo\_4Cyl" or "HugoConfig8Cyl" are not. This convention shall be checked by the corresponding ECUC processor.
- If forwarding takes place to a forwarding target which has no post-build capabilities at all, then a technically feasible superset of all data sets of the forwarding origin shall be pushed (e.g. when forwarding data from powerstage drivers to the OS).

#### ► Implementation:

- For each variant-specific configuration data set (e.g. "HugoConfig\_4Cyl", "HugoConfig\_8Cyl"), a corresponding C structure instance with the same name as this configuration data set shall be generated by the ECUC processor.
- These C structures shall contain or reference **all** configuration information belonging to the corresponding post-build data set.
- All these C structure instances shall be of the same type which shall have the name *<Comp>\_ConfigType* (e.g. "Hugo\_ConfigType").
- All post-build dependent C data shall be instantiated in *<Comp>\_PBcfg.c*. This file shall contain **only** post-build configuration data, and **no** other file belonging to this SW component shall contain information which depends on the post-build selectable contents of the post-build configuration data sets. The number of post-build data sets may also influence the contents of other files, though (e.g. if the number of post-build data sets is stored in a #define in *<Comp>\_Cfg.h*).
- All post-build dependent C data shall be located in a separate MemMap section *<COMP>\_START/STOP\_PBCFG\_<SIZE>*. No pointer inside this section should point to something outside of this section.
- The interface to this post-build configuration data shall be located in *<Comp>\_PBcfg.h*.

## 4.4.6 ECUC Processors: oAW-specific Procedures

The following pages describe oAW specific rules

## Rule ECUC\_OP\_001: File naming conventions

**Instruction** A oAW processor for a component *<Comp>* shall be a set of oAW scripts with the following naming conventions:

Workflow Control (Model Workflow Environment) files:

- for configuration forwarding named *<Comp>\_Forward.mwe*
- for configuration validation named *<Comp>\_Validate.mwe*
- for code generator named *<Comp>\_Generate.mwe*
- for ID generator named *<Comp>\_Generateld.mwe*

Forwarder scripts:

- configuration forwarder files: *<Comp>\_Forward\_<Sub>.ext*

Validator scripts:

- configuration validation files: *<Comp>\_<Sub>.chk*
- configuration validation extension scripts: *<Comp>\_<Sub>.ext*

Generator scripts:

- configuration generator template (Xpand) files: *<Comp>\_Generate<Sub>.xpt*
- configuration generator extension (Xtend) files: *<Comp>\_Generate<Sub>.ext*
- C code file generator template (Xpand) files: *<Comp>\_<Sub>\_Cfg\_c.xpt*
- C header file generator template (Xpand) files: *<Comp>\_<Sub>\_Cfg\_h.xpt*
- RTE configuration generator template (Xpand) files: *<Comp>\_<Sub>\_Cfg\_Bswmd\_<Suffix>.xpt* and *<Comp>\_<Sub>\_Cfg\_Swcd\_<Suffix>.xpt*

Generate ID scripts:

- ID generator files: *<Comp>\_Generateld\_<Sub>.chk*
- ID generator extension files: *<Comp>\_Generateld\_<Sub>.ext*

The file name parts *\_<Sub>* and *\_<Suffix>* are optional.

Examples: BswM\_Generate.mwe, Dem\_Events.chk, WgdM\_Genutils.ext.

**Caution** Please see the naming conventions of input and output data and artifacts in the chapter ECUC Processors: Content-related Procedures.

**Tip** As rule of thumb Xpand scripts that generates a file shall be named exactly like the generated filename where the point in its extension is replaced by *\_*.

Example: Dem\_Cfg\_h.xpt generates the file Dem\_Cfg.h

**Hint** The file extension for MWE files used to be *.oaw* in the past, it has changed to *.mwe* when oAW moved to Eclipse. However, *.oaw* is still accepted as extension.

## Rule ECUC\_OP\_002: Subdirectories

**Instruction** oAW scripts shall be placed in the *scripts* folder of the respective component.

If Subdirectories cannot be avoided, e.g. if the number of files is too large to have a good overview, the following folder structures shall be used:

- for generate ID and configuration forwarders: *scripts\forwarder*
- for code and other file generators: *scripts\generator*
- for configuration validations: *scripts\validator*
- for utility extensions: *scripts\util*

## Rule ECUC\_OP\_003: Forward scripts need separate action

**Instruction** Forward scripts shall have a separate action defined with complete input and output data declaration in BAMF.

This is required in order to provide only the necessary data to the forwarder, and to validate the forwarded data afterwards.

## Rule ECUC\_OP\_004: Id Generator scripts need separate action

**Instruction** ID Generators shall have separate actions defined with complete input and output data declaration in BAMF.

This is required in order to provide only the necessary data to the generator, and to validate the forwarded data afterwards.

## Rule ECUC\_OP\_005: Generator scripts need separate action

**Instruction** Generator scripts shall have a separate actions defined with complete input data and output artifact declaration in BAMF.

This is required in order to provide only the necessary data to the generator, and to verify if all artifacts have been created afterwards.

# 4.4.7 ECUC Processors: Perl-specific Procedures

## Rule ECUC\_PP001: File naming conventions

**Instruction**

A Perl processor shall be a Perl module named `<Comp>_Process.pm`. Additional helper Perl modules (if required) shall be named `<Comp>[_<Add>]_Ext.pm`.

Examples: `Fee_Process.pm`, `Fee_Helpers_Ext.pm`, `rba_IoSigDio_Process.pm`.

## Rule ECUC\_PP005: Action naming conventions

**Instruction**

Perl subroutines corresponding to processor actions shall be named according to the following scheme:

- ▶ subroutines preparing the configuration input for further processing<sup>6</sup> shall be called Prepare[<Suffix>]
  - ▶ subroutines generating files (model to text transformations) shall be called Generate[<Suffix>]
  - ▶ subroutines generating configuration data (model to model transformations) shall be called Forward[<Suffix>].
- The <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders in a Perl processor.

## Rule ECUC\_PP008: No WhoAml or Register in AUTOSAR BSW

### Instruction

The legacy Perl subroutines WhoAml() and Register() shall not be used in AUTOSAR BSW.

## Rule ECUC\_PP006: Restriction of output file locations and names

### Instruction

Perl subroutines corresponding to processor actions shall generate files only within the paths provided by the configuration framework. Subfolders of these paths shall neither be created nor used by Perl processor actions. The names of the generated files shall strictly adhere to the names provided in the corresponding build action manifest file.

Each Perl action gets the following parameters provided by BCT:

[1]: Array of paths for several types of generated files:

- [0]: C source files
- [1]: C header files
- [2]: Assembler source files
- [3]: Documentation files
- [4]: MSR PaVaSt files (not to be used in AUTOSAR BSW)
- [5]: Log files
- [6]: Report files
- [7]: Dump files
- [8]: MSR PaCoIn files (not to be used in AUTOSAR BSW)
- [9]: Hex files
- [10]: Export files
- [11]: Message files (not to be used in AUTOSAR BSW)
- [12]: Scheduling files (only to be used by OS)
- [13]: Post-processing files (only to be used by OS)
- [14]: MSR PaVaSt variant files (not to be used in AUTOSAR BSW)
- [15]: MSR PaVaLa files (not to be used in AUTOSAR BSW)
- [16]: AUTOSAR SWC description files
- [17]: XDI data files (not to be used in AUTOSAR BSW)
- [18]: XDI documentation files (not to be used in AUTOSAR BSW)
- [19]: EPS files

<sup>6</sup> Typical use cases for such prepare actions are the automatic calculation of ID values or centralized validations. This involves the reading and writing of these actions to the same configuration subtree.

[20]: PDF files  
[21]: SVG files  
[22]: MSR FS files  
[23]: ADX files

[2]: Path to the Perl processor itself (also known as the "selfpath").

There may be more arguments provided to the Perl actions (depending on the version of the configuration framework), but the values of these arguments shall not be used by the action's Perl code.

## Rule ECUC\_PP002: Usage of the conf\_process API

### Instruction

Wherever there is an API method of the configuration framework "conf\_process" available for a particular task, this method shall be used by all Perl processors instead of low-level Perl operations. Only the APIs documented below from the Perl modules documented below are allowed to be used.

The following API methods are available (alphabetically ordered within each package):

conf\_process:: CreateDumpFile:

Description: This routine creates a dump of the complete hash CONFHash or a part of it (depending on parameters) and writes it to the specified file. This should only be used for debugging purposes. Whether the output file is really created or not can be specified by a global conf\_process option.

Parameters: [0] Name or complete path of the dump file to be generated.

[1] Name of the top-level branch of the CONFHash to be dumped. If undefined, the entire CONFHash is dumped.

[2] If defined, the generated dump is appended to the specified file instead of overwriting its contents.

[3] If equal to 1, and if [1] is not defined, then the dump file contents are generated in "terse" mode, i.e. without specifying a variable name.

Returns: -

CreateDumpFileLocalHash:

Description: This routine creates a dump of a hash provided as an argument and writes it to the specified file. This should only be used for debugging purposes. Whether the output file is really created or not can be specified by a global conf\_process option.

Parameters: [0] Name or complete path of the dump file to be generated.

[1] Hash or array to be dumped.

[2] Name of the variable to be used in the dump (defaults to "Var" if missing).

[3] If defined, the generated dump is appended to the specified file instead of overwriting its contents.

Returns: -

Exit: Description: This routine assembles an entry containing the passed over error message plus some context information in the logfile, and then 'dies' with this message.

Parameters: [0] String to be put into the log file. Leading or trailing line breaks ("\n") are stripped from the log string before sending it to the log file.

Returns: - (never returns)

**GenerateFile:**  
 Description: This routine writes an arbitrary string to the specified file. If this file already exists and even contains exactly the contents provided, this routine avoids a rewriting of this file. This behaviour is necessary to get more efficient make behaviour (incremental make). If writing the specified file fails, this subroutine dies with an error message.

Parameters: [0] Path of the file to be written.  
 [1] Contents of the file to be written as a '\n' separated string.

Returns: -

**GetBoolean:**  
 Description: This routine converts a boolean parameter value into the corresponding numeric value for further Perl processing.

Parameters: [0] Boolean parameter value or string.

Returns: 1 if the input parameter is '1' or 'true', 0 if the input parameter is '0' or 'false'.

**GetCallerInfo:**  
 Description: This routine returns an array with the elements returned from the Perl caller() interface. The filename is made relative and the subroutine information is enriched in case it is an (eval).

Parameters: [0] Caller depth (normally 0).

Returns: Enriched caller list.

**GetCDefine:**  
 Description: This routine generates a pretty-printed list of C #defines from the provided parameters.

For example, by writing

```
conf_process::GetCDefine(
    "Input signals",
    ["Name", "Id"],
    [
        ["CLUTCH", "1"],
        ["BRAKE", "12"],
        ["ACCELERATION", "123"]
    ]
);
```

you get

```
/* Input signals */
#define CLUTCH      1
#define BRAKE      12
#define ACCELERATION 123
```

Parameters: [0] Headline of the list (put into a C comment).

[1] Array of column headers (currently ignored): the first array element describes the symbol and second describes the semantics of its value.

[2] Array of definitions. Each definition is again an array where the first array element defines the symbol and the second defines its value.

Returns: Pretty-printed list of C #defines as a string.

**GetCEnum:**  
 Description: This routine generates a pretty-printed C enum from the provided parameters.

For example, by writing

```
conf_process::GetCEnum(
    "Input signals",
    ["Name"],
    [
        ["Clutch"],
        ["Brake = 12"],
    ]
);
```



```

        ["Acceleration"]
    });

```

you get

```

/* Input signals */
enum
{
    Clutch,
    Brake = 12,
    Acceleration
};

```

Parameters: [0] Headline of the enum (put into a C comment).

[1] Array of column headers (currently ignored): shall contain only one element describing the semantics of the enum values.

[2] Array of enumeration literals. Each literal definition is again an array with exactly one element defining the C literal name.

Returns: Pretty-printed C enum as a string.

**GetCStruct:**

Description: This routine generates a pretty-printed C typedef struct from the provided parameters.

For example, by writing

```

conf_process::GetCStruct(
    "signals",
    ["Type", "Property"],
    [
        ["uint8", "Clutch"],
        ["uint16", "Brake"],
        ["uint32", "Acceleration"]
    ]
);

```

you get

```

typedef struct
{
    uint8 Clutch;
    uint16 Brake;
    uint32 Acceleration;
}signals;

```

Parameters: [0] Defines the name of the generated type.

[1] Array of column headers (currently ignored): the first array element describes the type and second describes the properties.

[2] Array of structure elements. Each element definition is again an array where the first array element defines the struct member type and the second defines its name.

Returns: Pretty-printed C typedef struct as a string.

**GetCStructInit:**

Description: This routine generates a pretty-printed C struct (or array) initializer from the provided parameters.

For example, by writing

```

conf_process::GetCStructInit(
    "const Hugo_ConfigType Hugo_Config_Left",
    "",
    ["Initializer"],
    [
        ["47"],
        ["&Hugo_SignalInits"],
        ["&Hugo_SignalProps"]
    ]
);

```

you get

```
const Hugo_ConfigType Hugo_Config_Left =
{
    47,
    &Hugo_SignalInits,
    &Hugo_SignalProps
};
```

Parameters: [0] Defines the type and name of the struct or array to be initialized.

[1] Shall always be "".

[2] Array of column headers (currently ignored): shall contain only one element describing the semantics of the initializers.

[3] Array of initializers. Each initializer is again an array with exactly one element defining the initial value of the current struct or array element.

Returns: Pretty-printed C struct or array initializer as a string.

GetCTable:

Description: This routine generates a pretty-printed C initializer for an array of structs from the provided parameters. The number of columns must be the same in all rows (including the column headers specification).

For example, by writing

```
conf_process::GetCTable(
    3,
    "const Hugo_SignalProps Hugo_SignalPropsTable",
    ["Resolution", "Trigger", "Inversion"],
    [
        [8, "Cyclic", 0],
        [12, "Event1", 1],
        [10, "Cyclic", 0]
    ]
);
```

you get

```
const Hugo_SignalProps Hugo_SignalPropsTable[3] =
{
    // Resolution, Trigger, Inversion
    { 8,          Cyclic, 0 },
    { 12,         Event1, 1 },
    { 10,         Cyclic, 0 }
};
```

Parameters: [0] Number of array elements (shall match the number of struct initializers given in argument [3]).

[1] Defines the type and name of the array to be initialized.

[2] Array of column headers describing the struct elements.

[3] Array of structure initializers. Each initializer is again an array containing the initializers of each struct element.

Returns: Pretty-printed C array of structs initializer as a string.

GetFileContent:

Description: This routine offers a method for opening files and returns the file content as a '\n' separated string. If opening the specified file fails, this subroutine dies with an error message.

Parameters: [0] Name of the SW component to which the file to be opened belongs (e.g. "Xyz").

[1] Full path of the file to be opened.

Returns: Contents of the file as a '\n' separated string.

**GetInteger32Bit:**

Description: This routine returns a decimal value of an integer given in decimal or hex notation.

Parameters: [0] Integer number or string.

Returns: Decimal value of the input parameter, undef if an invalid decimal or hexadecimal value has been provided or if this value exceeds the 32bit range.

**GetInterfaceVersion:**

Description: This routine allows to query the current version of the Perl interface of the configuration framework. This version will change if the interface incompatibly changes.

Parameters: –

Returns: Current interface version of the Perl interface of the configuration framework.

**IsModuleExistent:**

Description: This routine queries whether the specified SW component exists.

Parameters: [0] Name of the SW component to be queried for existence.

Returns: 1 if the specified SW component exists, 0 otherwise.

**Log:** Description: This routine creates an entry in the log file.

Parameters: [0] String to be put into the log file. Leading or trailing line breaks ("\n") are stripped from the log string before sending it to the log file.

Returns: –

**LogEmpty:**

Description: This routine creates an empty line in the log file.

Parameters: –

Returns: –

**MakePathPortable:**

Description: This routine converts a path specification in a portable form by converting all platform-specific end-of-line characters to "\n" and all backslashes "\" to forward slashes "/" (which work on both Windows and Unix).

Parameters: [0] Path to be made portable.

Returns: String containing a portable path, i.e. with forward slashes as path separators.

**MakePathRelative:**

Description: This routine converts a given absolute file path specification (e.g. "C:\Work\MyCurrent-Project\MyPackage\MyComponent\MyFile.ext") to a portable path specification which is relative to the project root folder (e.g. "MyPackage/MyComponent/MyFile.ext").

Parameters: [0] Path to be made relative to the project root.

Returns: Relative path.

**Pop:**

Description: This routine reads data from the configuration repository. The usage of this routine is obligatory when a configuration Perl module has to read from this repository.

Parameters: [0] Name of the SW component whose configuration data shall be read.

Returns: Configuration data of the specified SW component if existent, undef otherwise.

**Push:**

Description: This routine writes new data or additional data to the configuration repository. The usage of this routine is obligatory when a configuration Perl module has to write to this repository. Please note that calls to this routine are potentially time-consuming. Therefore it is recommended to do only one big Push call per action rather than many small Push calls.

Parameters: [0] Name of the SW component which initiates the push operation.

[1] Array of data destinations. For example, if the array (a, b, c) is supplied, then the data is pushed to container c inside container b in SW component a. Please note that if the destination is specified using AUTOSAR ECUC methodology, only a single array entry corresponding to the destination module shall be provided. This is due to the fact that each ECUC container requires a \*\_\_KEY value in AUTOSAR, even if its value is not evaluated by any ECUC processor at all.

[2] Data to be pushed. Can be a reference to a hash or an array.

[3] ArPackage of the push destination. The ArPackage of the destination's EcucParam-Def is expected here, e.g. /AUTOSAR\_Dio/EcucModuleDefs or /RB/RBA/rba\_loSigDio/EcucModuleDefs. For pushes to MSR structures, this parameter shall be left undefined.

Returns: -

RegisterDynamicArtifact:

Description: This routine registers a dynamically generated file in the build framework. Please relate to rule [\[ECUC\\_PP007\]](#) for the conditions when it is allowed to use this routine.

For Perl processors which shall run unchanged in both legacy (BCT-less) configuration environments and within BCT, then the legacy calling convention is still available: RegisterDynamicArtifact(role, name, category, parent, parent\_category). Perl processors without this backwards compatibility constraint (which is the vast majority of Perl processors) shall use the single parameter form instead.

Parameters: [0] Name of the generated file (no path specifications allowed).

Returns: -

## Rule ECUC\_PP003: Usage of global variables

### Instruction

Global variables in Perl (i.e. variables in file scope) are evil. Try to avoid them whenever somehow possible.

Global variables are problematic for several reasons:

- ▶ They make a Perl module very hard to maintain.
- ▶ They prohibit that Perl actions can be invoked independently from each other (important in particular for incremental build behaviour).
- ▶ They increase the effort for porting Perl code to oAW.

A good measure for avoiding global variables is creating helper subroutines which derive the desired information from the configuration repository as the only global information source.

## Rule ECUC\_PP004: Conventions for injection markers

### Instruction

Injection markers shall always have the form `</Identifier/>`.

To be consistent with existing Perl processors, the identifiers used for these injection markers shall be in all uppercase using "\_" as word separators. The first word in these injection markers shall be the name of the SW component which owns this Perl processor.

Examples: `</FEE_INCLUDES/>`, `</RBA_IOSIGDIO_CFG_DEV_ERROR_DETECT/>`

## Rule ECUC\_PP007: Dynamic registration of generated files

### Instruction

Whenever it is not possible to specify the name of a generated file statically in the module's BAMF file (and hence,

a wildcard operator is used in the related BAMF section), each generated file with file name determined at Perl processor runtime

- ▶ shall be registered using the RegisterDynamicArtifact API of conf\_process and
- ▶ shall match exactly one BAMF specification of created artifacts which uses wildcard operators.

Whenever the *name* of a generated file can already be known at BAMF specification time, dynamic registration of generated files is not permitted.

## 4.5 Build Action Manifests

The Build Action Manifest Format (BAMF) defines the following Objects which rules can apply to:

- ▶ BuildActions
- ▶ Input/Output-Elements (IO-Elements)
  - Artefacts
  - Model References

For each of these objects, different constraints exist. In addition, there are also some constraints which apply to specific types of actions (that is Perl/oAW... other types will be added in the future).

### 4.5.1 BuildAction Declarations

BuildActions are the topmost building block of the BAMF file format. They basically correspond to one node in the Build graph which is executed during a software build. All other object definitions in BAMF always relate to BuildActions and are therefore used during build time. BuildActions contain instance specific invocation parameters which are then used to configure how they work internally. Also they contain a textual identifier for the environment they require to be executed properly.

#### Rule ECUC\_B001: BuildActions names shall be unique

##### Instruction

All BuildActions to be executed within a given context (that is in a full program stand at max) shall have unique names.

As a BuildActions name is its sole identifier during software build, it has to have a unique name so the underlying frameworks don't get mixed up with them. The chances for such a misconfiguration are especially high as BuildActions can reside in multiple files scattered throughout the project. In case there are two or more BuildActions with the same name, they can't be processed properly as they can have invocation parameters and IO-Elements which contradict.

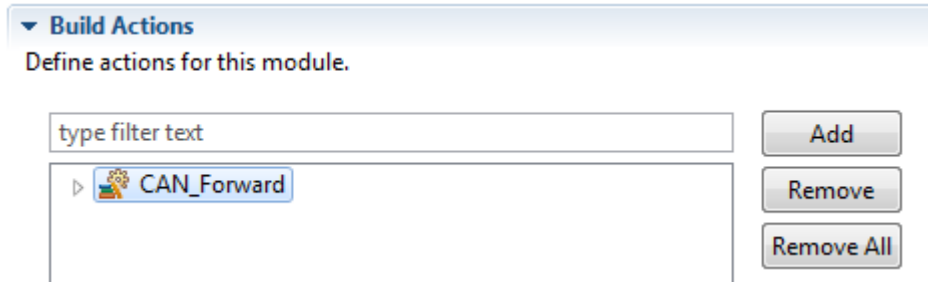
#### Rule ECUC\_B002: Naming conventions for BuildActions

##### Instruction

All BuildActions should comply with the following naming convention: <ModuleName>\_<Function> where the <ModuleName> corresponds to the BSW module shortname the script was developed for and the <Function> is either Generate or Forward. For legacy MSR modules, the possible values for <Function> are Register, SetGlobals, Forward and Generate.

The following shows a correctly named BuildAction which forwards data for the BSW module CAN:

Figure 21 Properly named forwarder BuildAction for the BSW module CAN



## Rule ECUC\_B003: Declaring Startup-/Teardown Action References

### Instruction

Startup-/Teardown Action References should be declared within just one BAMF file.

Theoretically it is easily possible to distribute Startup-/Teardown Action References over lots of different BAMF files. However this can easily lead to duplicate/contradicting declarations (for instance it doesn't make much sense to declare one BuildAction as Startup and Teardown Action). To minimize the risk for possible misconfiguration, it is highly recommended to have all Startup-/Teardown Action References declared within one central BAMF file.

## Rule ECUC\_B004: Usage of Startup-/Teardown-/Followup and Predecessor Action References

### Instruction

Before introducing any declaration such as Startup-/Teardown-/Followup- or Predecessor Action References it shall always be made sure that the corresponding relationship to other declarations can't be achieved with regular IO-Element declarations with justifiable effort.

Theoretically, it would be possible to statically configure Action dependencies through Followup-/Predecessor Action References and/or Startup-/Teardown Action References. However this would not comply with the basic paradigms introduced with the Build Action Manifest concept. Ideally, each BuildAction shall declare its inputs and outputs (be it model references or Artefact declarations) and therefore the toolchain can calculate the optimum execution order for all BuildActions in the current context. Therefore instead of statically defining that an Action should be executed before/after another one, they should declare their inputs/outputs and let the toolchain step in. The simple reason for that is that the BAMF concept is decentralized. That means: Module developers create their own BAMF file containing their Build-Actions locally. They will be executed later in a larger context (eg. in full program stand context). Explicite dependency declarations to other BuildActions would potentially need to be altered upon integration. Declared inputs/outputs are equally valid in full program stand and smaller contexts. Therefore this should always be the preferred solution.

Figure 22 The following two actions have a well defined execution order no matter in which context they're executed

Build Actions

Define actions for this module.

Add
Remove
Remove All

Dio\_Generate

PROCESSOR: Dio\_Process.pm [CUBAS:CC

/AUTOSAR\_Dio/EcucModuleDefs/Dio

Dio\_Prepare

PROCESSOR: Dio\_Process.pm [CUBAS:CC

/AUTOSAR\_Dio/EcucModuleDefs/Dio

Build Action Details

Set the properties of the selected action. Required fields are denoted by "\*".

Short Name\*: Dio\_Generate
Category\*: GENERATOR
Required Environment\*: /RB\_Manifest/com\_bosch

Invocation parameters

Follow-Up Actions

The following actions shall run after the selected action. Remark: Use follow-up actions with extreme caution.

Add
Remove
Remove All

Predecessor Actions

The following actions shall run before the selected action. Remark: Use predecessor actions with extreme caution.

Add
Remove
Remove All

## Rule ECUC\_B005: Invocation Parameter keys

### Instruction

A BuildAction instance shall not have two or more invocation parameters with the same key.

Invocation parameters are used to locally configure a BuildAction instance. As invocation parameters are key value pairs, and individual invocation parameters are identified by the parameter key it is a technical constraint that there is never more than one invocation parameter for a given BuildAction with one and the same key. Otherwise the framework would have to choose between two potentially contradicting values for the parameter.

## 4.5.2 Input/Output Declarations

Every BuildAction should declare its input and output relationships with its environment by attaching Input/Output--Elements (or IO-Elements for short). These can either be some kind of model reference or artefacts which correspond to files. These IO declarations are used to determine the possible execution order of the actions and accordingly model data/file content is passed from one BuildAction to another during build time.

### 4.5.2.1 Artefact Declarations

Artefact declarations in a BAMF file correspond to files which are required/read by a BuildAction instance or created/written by a BuildAction instance. These declarations are necessary so that files can be registered throughout the

build process, passed to other actions, etc. Artefact declarations are also used to determine a valid execution order of BuildActions present.

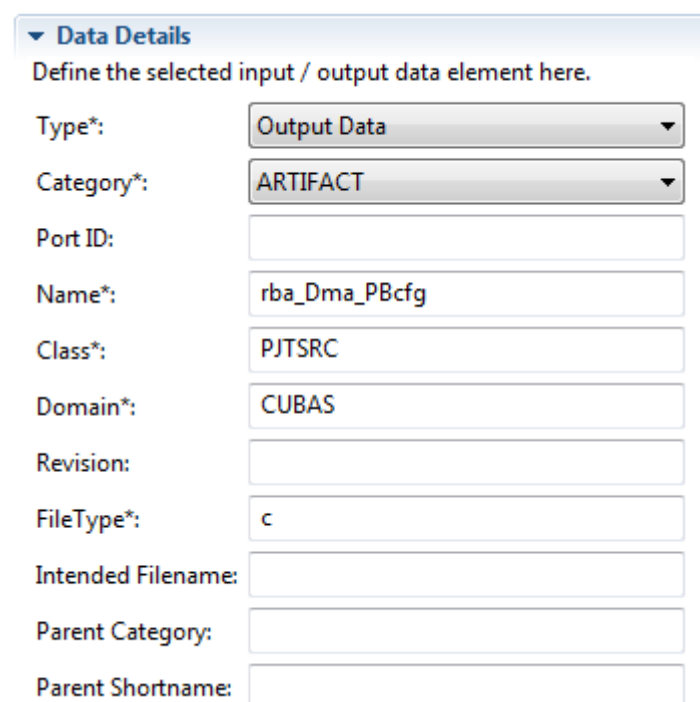
## Rule ECUC\_BIOA001: Artefact Declaration Composition

### Instruction

The AUTOSAR standard defines an artefact declared in a BAMF file by the following four attributes: Artefact name, Extension, Class and Domain. Every artefact declaration shall contain a valid value for all of them.

An Artefact is well defined when all of the attributes are matching to exactly one String. That is: even though AUTOSAR allows the usage of wildcards for some of the attributes, none of these are used. The following picture shows a valid declaration of an Artefact:

Figure 23 A well defined artefact



▼ Data Details

Define the selected input / output data element here.

Type\*:

Category\*:

Port ID:

Name\*:

Class\*:

Domain\*:

Revision:

FileType\*:

Intended Filename:

Parent Category:

Parent Shortname:

## Rule ECUC\_BIOA002: Artefact Declaration Characteristics

### Instruction

As defined by the rule above, there are four attributes which clearly define an artefact. The characters which are allowed to be used for each of them can be found below and no declaration shall contain any other character but the ones mentioned in this rule.

Artefact Name: [A-Z][a-z][0-9]\_

Filetype: [A-Z][a-z][0-9]\_

Class: [A-Z][a-z][0-9]\_

Domain: [A-Z][a-z][0-9]\_

This rule will change slightly in the future as the Bosch toolchain will also support usage of the \* operator as a wildcard for Artefact Name and Filetype. However at this point in time this is not supported. Therefore only alphanumerical characters as well as the underscore character are valid. Please note that artefact declarations are case sensitive and shall match exactly to generated/read artefacts in the project.



In general every Class is theoretically valid. But since the Class will define the content of a file and other tools will collect files of the same class for further processing it is useful to take care of the following list of commonly used Classes for Output file artifacts:

Table 27 Commonly used classes for generated files

File Type	File extension	Class
C source files	c	PJTSRC
C header files	h	PJTHDR
BSWMD files	arxml	PJTBSWMD
SWCD files	arxml	PJTSCWD
Measurement/Calibration Support files	arxml	PJTMCSUPP
Report files	txt	PJTREP
BSW Documentation (FS files)	xml	PJTCOREFS

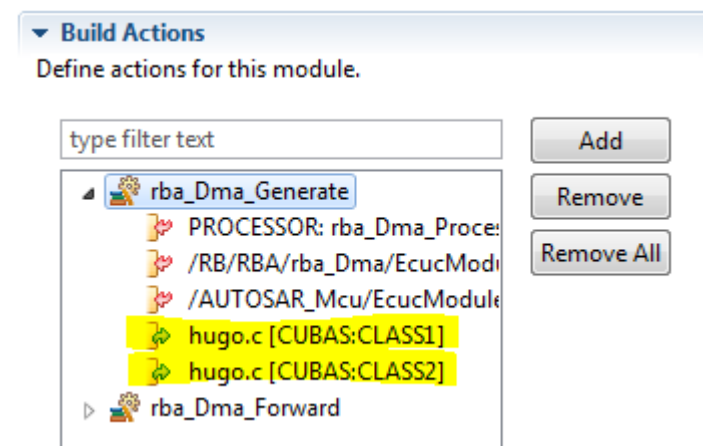
## Rule ECUC\_BIOA003: Uniqueness of Output Declarations

### Instruction

One BuildAction instance shall never contain duplicate output element declarations which will be applied to one and the same artefact.

Under one BuildAction, it is theoretically possible to declare one artefact twice but with different attributes. As once a code generator script has done its work it is not always given that meta information for said artefact is already existing then (say in case of an LWS project, registration still needs to take place before the file can be identified by its artefact name, class, etc. Therefore the following output declaration would be invalid as it would want to register one and the same artefact with different classes at the same point in time:

Figure 24 Invalid Output Artefact Declaration



### 4.5.2.2 Model Reference Declarations

Model References in a BAMF file correspond to BSW modules defined in AUTOSAR/MSR format as of today. This means: BuildActions need to contain IO-Element declarations which in turn correspond to Module Definitions. That is: a BuildAction script processing configuration data from a given module (say CAN) needs to explicitly declare it does so. If forwarding is involved in the scripts, they also need to declare the modules they forward to in BAMF. The declarations are used to calculate a valid processing order for the BuildActions during build.

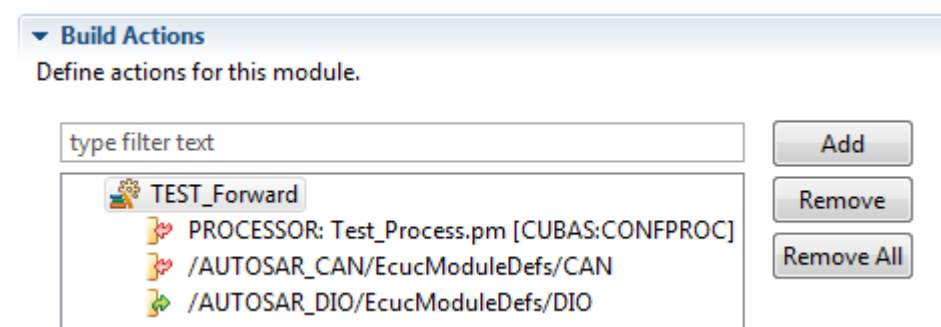
## Rule ECUC\_BIOM001: Required declaration of accessed Model Data

### Instruction

Build scripts of any technical nature (Perl or oAW) shall only access model elements they also explicitly declared in their corresponding Build Action. That means, a script is only allowed to read data from /AUTOSAR\_CAN/EcucModuleDefs/CAN if it also declared this AR\_OBJECT to be an input IO-element under the same BuildAction it is encapsulated by.

The following is an example of a BuildAction which declares its input and output properly: It can access data from the AUTOSAR standardized module CAN and forwards data to the AUTOSAR standardized module named DIO.

Figure 25 Valid input/output declarations for a BuildAction



## Rule ECUC\_BIOM002: Model Reference Identifiers

### Instruction

For any reference to a BSW module, the following reference pattern shall be used:

For an AUTOSAR standardized module named <BSWM>: /AUTOSAR\_<BSWM>/EcucModuleDefs/<BSWM>

For a Non-Standardized module in AUTOSAR format named <BSWM>: /RB/RBA/<BSWM>/EcucModuleDefs/<BSWM>

For a legacy MSR module named <BSWM>: /MEDC17/<BSWM>

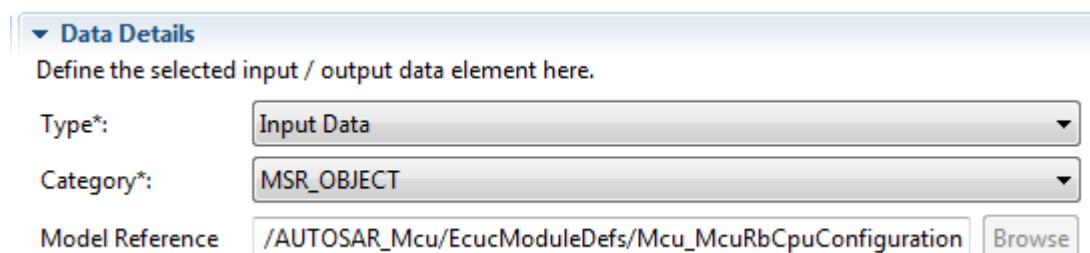
## Rule ECUC\_BIOM003: Model Reference Category

### Instruction

Whenever a model object is referenced by an IO-Element, the corresponding category needs to be used. That is in case of an MSR reference, the category shall be MSR\_OBJECT, in case of an AUTOSAR reference, an AR\_OBJECT shall be used. The model reference text needs to correspond to the rule mentioned above.

The following screenshot illustrates an invalid configuration of a model referencing IO-element which mixes syntactical reference to an AUTOSAR standardized module but declares that the content is an MSR\_OBJECT.

Figure 26 Invalid mixture of MSR and AUTOSAR model reference declaration



## 4.5.3 Action Specific Declarations

This chapter contains some rules which only apply for a given type of BuildAction. AUTOSAR defines that BuildActions themselves don't have a type per se. Instead they only require a given execution environment which can be identified by a string. In order to make explanations easier understandable we use the following mapping so we can actually talk about different action types instead of actions referring to a given execution environment (even though this would be the formally correct way):

Table 28 Action Type – Required Environment Mappings

Required Environment used	Action Type
/RB_Manifest/com_bosch_configfw_bfwaction_setup	Setup Action
/RB_Manifest/com_bosch_configfw_bfwaction_teardown	Teardown Action
/RB_Manifest/com_bosch_configfw_bfwaction_processstep	Perl Action
/RB_Manifest/com_bosch_configfw_bfwaction_oaw	oAW Action

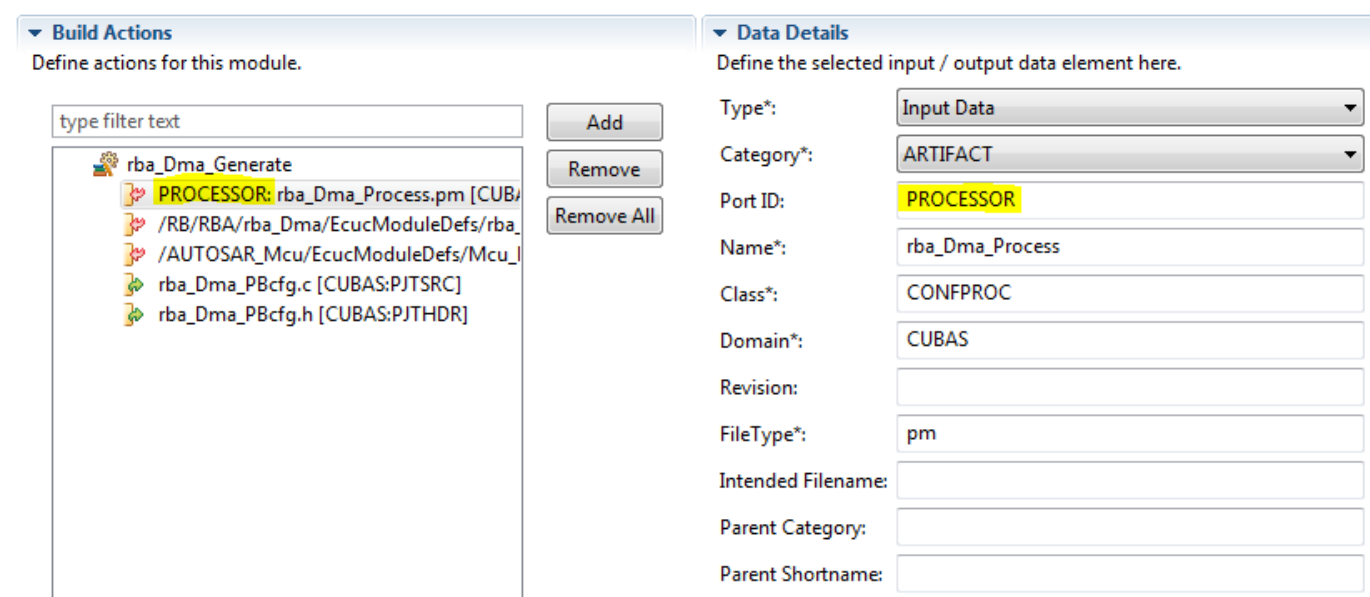
### Rule ECUC\_BAS001: Artefact marked as processor

#### Instruction

Every Perl and oAW Action shall declare exactly one input artefact which is well defined (that is: no usage of wildcards in the defining attributes allowed) and marked with the Port-ID PROCESSOR.

As it is theoretically possible to declare any number of input artefacts for any BuildAction, Perl and oAW Actions need to mark the script to be executed in their context as the processor. This can be achieved by assigning the Port-ID PROCESSOR. The mark should occur exactly once per oAW/Perl BuildAction. Otherwise the toolchain wouldn't know which script to trigger at buildtime. Also a Perl/oAW Action without any such declaration will not be executable for the same reasons. The following image shows a valid declaration of a Perl Action containing exactly one well defined processor artefact:

Figure 27 Valid Processor declaration for a Perl Action



The screenshot displays the 'Build Actions' configuration interface. On the left, under 'Build Actions', a list of artefacts is shown, with 'PROCESSOR: rba\_Dma\_Process.pm [CUBAS:PJT SRC]' highlighted. On the right, the 'Data Details' panel shows the configuration for the selected artefact:

- Type\*: Input Data
- Category\*: ARTIFACT
- Port ID: PROCESSOR
- Name\*: rba\_Dma\_Process
- Class\*: CONFPROC
- Domain\*: CUBAS
- Revision: (empty)
- FileType\*: pm
- Intended Filename: (empty)
- Parent Category: (empty)
- Parent Shortname: (empty)

### Rule ECUC\_BAS002: Generated Artefacts for Perl Actions

#### Instruction

Output and log directory can be centrally configured for all Perl Actions in the single Setup Action which must exist in

any Perl processing project. Any perl script creating output artefacts which are also declared in their corresponding BAMF files shall to stick to these centrally configured directories.

As the build toolchain can't monitor the complete filesystem, it only expects output artefacts to be created within the centrally declared output/log folders configured in the Setup Action. Every artefact declared but generated elsewhere will lead to possible build failure. Similarly, any undeclared artefact which is generated can't be further processed within the build toolchain.

## Rule ECUC\_BAS003: Generated Artefacts for oAW Actions

### Instruction

Output artefacts which should be processed in the build toolchain need to be configured via the appropriate means for oAW (outlets defined in the workflow file or referenced centrally). Due to limitations in the toolchain, the output folder should be \_out directly underneath the project root at the moment. At a later release it will be possible to configure the output folder for each oAW Action in its corresponding BAMF file. Once this is the case, this rule will have to be altered.

Just like with the Perl Actions, the build toolchain expects all output which should be processed/declared in BAMF files to be present in a well defined folder. Due to technical limitations this is currently restricted to the folder \_out under the project root.

## Rule ECUC\_BAS004: Perl Actions Required Invocation Parameters

### Instruction

The following parameters shall be present for any Perl Action and be filled with a valid value:

MODULE\_NAME

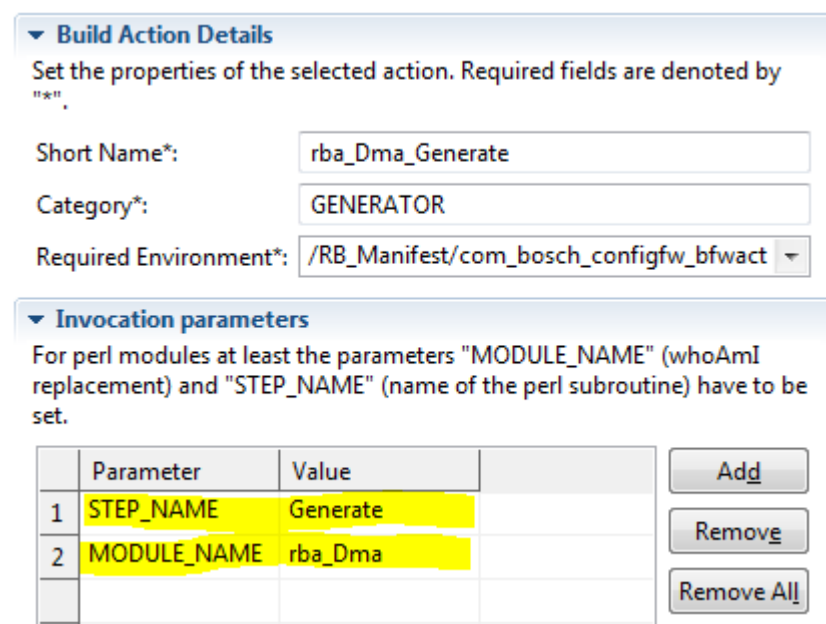
STEP\_NAME

The value for the parameter MODULE\_NAME shall be the same as the first part of the BuildAction name

The value for the parameter STEP\_NAME shall be the same as the last part of the BuildAction name

Example for a Perl Action containing all required invocation parameters:

Figure 28 Perl Action equipped with all required Parameters



The screenshot shows a 'Build Action Details' dialog box. The 'Short Name\*' field is 'rba\_Dma\_Generate', 'Category\*' is 'GENERATOR', and 'Required Environment\*' is '/RB\_Manifest/com\_bosch\_configfw\_bfwact'. Below this is the 'Invocation parameters' section, which contains a table with two rows: 'STEP\_NAME' with value 'Generate' and 'MODULE\_NAME' with value 'rba\_Dma'. To the right of the table are buttons for 'Add', 'Remove', and 'Remove All'.

	Parameter	Value
1	STEP_NAME	Generate
2	MODULE_NAME	rba_Dma

## Rule ECUC\_BAS005: oAW Actions Required Invocation Parameters

### Instruction

There are no mandatory invocation parameters for an oAW Action as of today. As this will change in the future, they'll be added to this rule.

## Rule ECUC\_BAS006: Setup Action Required Invocation Parameters

### Instruction

There are no mandatory invocation parameters for the Setup Action today. As this will change in the future, they'll be added to this rule.

## 5 Rule Set: Perl Coding Rules

### 5.1 Code Layout and Comments

#### Rule Perl\_001: Sections of Perl files

**Instruction** Every source file contains the following sections

- ▶ a file header (with copyright information and a short description)
- ▶ the actual source code

The file header provides important information about the global contents of a file, and has the following layout:

Figure 29 Perl Module Header

```
#!/usr/bin/perl -w
#
#####
# COPYRIGHT RESERVED, Robert Bosch GmbH, 2014. All rights reserved.
# The reproduction, distribution and utilization of this document as well as the communication of its contents to
# others without explicit authorization is prohibited. Offenders will be held liable for the payment of damages.
# All rights reserved in the event of the grant of a patent, utility model or design.
#####
#
```

The year in the file header represents the creation respectively the first release of the corresponding file. Therefore the year has not to be updated when an existing file is changed. But for new files the year shall be set to the current one.

#### Rule Perl\_002: Comments

**Instruction** Comments in the code shall be written in English

- ▶ The code you write might one day be useful to some other programmer who doesn't speak your native language.
- ▶ Comments should only be used when they clarify the code. They explain the "why", not the "how".
- ▶ Comment entire blocks, not single lines.

#### Rule Perl\_003: Comment of a subroutine

**Instruction** Every function in the Perl source is preceded by a comment block, specifying a synopsis of the defined function

The function header provides a small summary of what this function does and describes each parameter and return value:

```
#-----
#
# FooFunction
#
# This is an (optional) comment for the defined function, targeted at
# the code developer. This section typically explains the data
# structure and algorithm that were used.
#
# parameters: 0: description of parameter 0
#             1: description of parameter 1
# return value:
#             0: description of return value 0
#             1: description of return value 1
#
#-----
sub FooFunction
```

```
05 {  
    ...  
}
```

## Rule Perl\_004: Indentation and spacing style

**Instruction** Use the following indentation style:

- ▶ use 2-column indentation; don't use hard tabs
- ▶ opening curly bracket below to the keyword in the next line at the column as the keyword ("open braces left")
- ▶ closing curly bracket lines up with the keyword that started the block
- ▶ blank lines between groups of code that do different things
- ▶ no space between function name and its opening parenthesis
- ▶ space after each comma
- ▶ space between a keyword and opening parenthesis
- ▶ space around operators
- ▶ line up corresponding items vertically
- ▶ use parenthesis to indicate evaluation order
- ▶ use spaces where it improves readability

```
35 # open braces left  
if ( $flag_s eq "h" )  
{  
    $headers_s = 0;  
}  
40  
# function call  
$returnValue_s = FooFunction( $input1_s, $input2_s, $output1_s, \%output2_h );  
  
# use spaces  
$res = (($a + $b) / $c) * 10;  
  
# line up corresponding items  
$line_s      = 1;  
$up_s        = 2;  
$things_s    = 3;  
50 $vertically_s = 4;  
  
my $filename_s = $args_h{PATHNAME};  
my @names_a    = @{ $args_h{FIELDNAMES} };  
my $tab_s      = $args_h{SEPARATOR};  
55  
socket(SERVER, PF_UNIX, SOCK_STREAM, 0) || die "socket $sockname_s: $!";  
bind  (SERVER, $uaddr_s)                 || die "bind $sockname_s: $!";  
listen(SERVER, SOMAXCONN)                || die "listen $sockname_s: $!";  
  
60 $rot13_s =~ tr[a-mn-z]  
          [n-za-m];
```

## Rule Perl\_005: Line length

**Instruction** The line length of 120 characters shall not be exceeded.

## 5.2 Naming Conventions

### Rule Perl\_006: Packages and filenames

**Instruction** Perl script associated files shall start with the module name and are written in camel case (e.g. mixed upper/lower case).

Valid filenames:

```
# valid filename

MyModule.pm           # Part of module MyModule
BootCtrl_Process.pm   # Part of module BootCtrl
rba_IoSigDio_Process.pm # Part of module rba_IoSigDio
```

### Rule Perl\_007: Constants

**Instruction** Constants are written with uppercase characters. They begin with letters, followed by word characters and underscores used to separate the components in a long name. Constants are preferably defined with the "use constant" pragma.

```
# pragma-style: the recommended way to define constants
use constant PI_CONST => 3.14159;
use constant A_CONST => ( 11, 22, 33 );

# compile-time error!
PI_CONST = 4711; # error!
(A_CONST)[0] = 88; # error!
```

Please note:

- ▶ Pragma-style constants are real constants, i. e. the Perl interpreter ensures that you cannot assign a new value to them. On the other side, it's more difficult to interpolate pragma-style constants into strings.
- ▶ Pragma-style constants work for scalars and arrays, not hashes.

```
# examples of usage constants
print "This is the number of PI: PI_CONST\n"; # will not work!
printf "This is the number of PI: %f\n", PI_CONST;
print "This is the 0. element of A_CONST: " . (A_CONST)[0] . "\n";

$comp1_s = 3 * PI_CONST;
$comp2_s = 3 * (A_CONST)[0];

printf "This is the number of 3*PI: %f\n", $comp1_s;
printf "This is the number of 3*(A_CONST)[0]: %f\n", $comp2_s;
```

### Rule Perl\_008: Naming of global and local variables

#### Instruction

- ▶ Variable names are written in mixed upper/lower case, and start with a lower-case letter
- ▶ All variables should be lexical (defined with my) and global variables should be avoided.
- ▶ Prefixes may specify the kind of usage of a local variable.



- ▶ The names of variables are suffixed with:
  - "\_h" for hash variables
  - "\_a" for array variables
  - "\_s" for scalar variables
- ▶ Suffixes are not used for variables with a very limited scope (e. g. loop indices).

```
my $verbose_s = 0;
my @stateTable_a = ( 1, 2, 3 );

foreach my $value_s ( @stateTable_a )
{
    my $currentLine_s;
    $currentLine_s = &getLine( $value_s );
    ...
}
```

Please note:

- ▶ Please have look also at the Rule Set "Naming conventions" of this document
- ▶ Please use abbreviations from the table appendix of this document.
- ▶ Local and global variables should have descriptive names, when they are used for more than 2 or 3 nearby lines.
- ▶ For variables that have a very limited scope (loop indices, block-local variables), it often improves legibility when you use short names. So don't write:

```
for ( my $index_s = 0; $index_s < $#table_a; $index_s++ )
{
    $table_a[$index_s] += 2;
}
```

when you should be writing this:

```
for ( my $i = 0; $i < $#table_s; $i++ )
{
    $table_s[$i] += 2;
}
```

- ▶ Please use prefixes from the table appendix of this document.

```
# Prefix 'opt'
$result_s = GetOptions ( "length=i" => \$optLength_s,      # numeric
                        "file=s"   => \$optData_s,         # string
                        "verbose"  => \$optVerbose_s);      # flag

# Prefix 'nr'
$nrElems_s = @elems_a;

# Prefix 'adr'
$adrStartup_s = 0x80018000;

# Prefix 'cntr'
for $person_s (keys %persons_h)
{
    if ($persons_h{$person_s}{"sex"} eq "female")
    {
        $cntrFemale_s++;
    }
    else
    {
        $cntrMale_s++;
    }
}
```

```
}

# Prefix 'st'
$stRevers_s = GetReversibility();

if ($stRevers_s) # success case
{
    ...
}
```

## Rule Perl\_009: References

**Instruction** The names of reference variables are suffixed with

- ▶ "\_ph" for references to hash variables
- ▶ "\_pa" for references to array variables
- ▶ "\_ps" for references to scalar variables
- ▶ "\_pg" for references to typeglobs
- ▶ "\_pf" for references to functions

```
# Global variables
my @names_a = qw( John Jane);
my %address_s = ( "address" => "Park Ave",
                  "city" => "Baltimore" );
# Global references (referencing)
my $names_pa = \@names_a;

sub Foo
{
    ...
}

{
    # Local variables
    my $state = "completed";

    # Local references (referencing)
    my $address_ph = \%Address;
    my $state_ps = \$state;
    my $stdout_pg = \*STDOUT;
    my $foo_pf = \&foo;

    # Dereferencing
    print &$foo_pf(1);
    print $stdout_pg 'Hello world';
    print $$state_ps;
    print %$address_ph;
    print @$Names_pa;
}
```

## Rule Perl\_010: Names of subroutines

**Instruction** Subroutines names are written in camel case, and start with an upper-case letter.

```
sub IsReady
{
    ...
}

sub PrintLog
```

```
{  
    ...  
}
```

Please note:

- ▶ The main difference between procedures and functions is that a function returns a result, a procedure does not. A procedure only can return a state value or an error code.
- ▶ Procedure names should reflect what they do; function names should reflect what they return
- ▶ Predicate functions should usually be named with 'is', 'does', 'can' or 'has'. Thus, `&IsReady` is better than `&Ready` for the same function
- ▶ Therefore, `&Canonize` as a void function (procedure), `&CanonicalVersion` as a value-returning function, and `&IsCanonical` for a boolean check.

## Rule Perl\_0101: File handles

**Instruction** File handles (typeglobs) are written in upper case letters.

```
open FILE, ">", "filename.txt" or die $!  
$firstLine_s = <FILE>;  
doSomethingWithFile(\*FILE);  
close FILE;
```

Since Perl 5.6, indirect file handles are supported. A file can be opened and its file handle can be stored directly in a lexical variable. This variable contains a reference to this file handle and can also be passed to a subroutine.

```
my $file_s;  
  
open $file_s, ">", "filename.txt" or die $!  
$firstLine_s = <$file_s>;  
doSomethingWithFile($file_s);  
close $file_s;
```

## Rule Perl\_011: Names for hashes and arrays

**Instruction** Name of arrays may be in plural and hashes in singular.

- ▶ Because hash entries are typically accessed individually, it makes sense for the hash itself to be named in the singular.
- ▶ Arrays are usually ordered sequences of multiple values, and are most commonly processed collectively or iteratively in loops or in map or grep operations. So it makes sense to name them in the plural, after the group of items they store.

```
# Usual case  
my @numbers_a = qw(4 5 6);  
my @persons_a = <FILE>;  
my %person_h = ( "name" => "Fritz", "age" => 19 );  
  
# Exceptions  
my %computers_h = ( "aix"      => "134.94.100.100",  
                   "solaris" => "134.94.100.24",  
                   "www"     => "134.94.100.51",  
                   );  
  
my @time_a=localtime(time());  
$hour_s  = $time_a[2];  
$month_s = $time_a[4];
```

## 5.3 Coding Conventions

### Rule Perl\_012: Function Calls

**Instruction** The first thing to do in a function is to fetch its parameters

► Never directly use @\_ or \$\_[0],...,\$\_[n]

► All parameter variables should instead be assigned to local variables first.

```
my $result_s = RaiseToPower( $value, $power );
my @len_s = GetLengthList( \@list1, \@list2 );
```

...

```
sub RaiseToPower
{
    # Formal parameters
    my ( $value_s, $power_s ) = @_;

    return $value_s ** $power_s;
}
```

...

```
sub GetLengthList
{
    # Formal parameters
    my $list1_pa = $_[0];
    my $list2_pa = $_[1];

    # Local parameters
    my $numList1_s;
    my $numList2_s;

    $numList1_s = @$list1_pa;
    $numList1_s = @$list1_pa;

    return ( $numList1_s, $numList2_s );
}
```

### Rule Perl\_020: Reading Configuration Data of Other Modules

**Instruction** The function `conf_process::IsModuleExistent()` shall be used to determine the existence of an optional Module (i.e. a Module which is not necessarily part of a project environment). To actually get access to its data, an appropriate \*bamf file entry is required, see rule set "ECU Configuration".

**Hint** Even if `IsModuleExistent` returns true, a `conf_process::Pop` may return undef if no ECUC values exist for this (existing) module.

Application example:

```
# CalWup data is only required when EtkEnable = YES
if($EcucGeneral_ph{EtkEnable} eq 'YES')
{
    if( conf_process::IsModuleExistent('rba_CalWup') )
    {
        # Get the ECUC values for rba_CalWup
        my $EcucCalWup_ph = conf_process::Pop("rba_CalWup");

        .....
    }
}
```

```
05     }  
    else  
    .....  
}
```

## 5.4 Programming Tips

### Rule Perl\_014: Defensive Programming

**Instruction** Make a habit of defensive programming

- ▶ always use the `-w` option when you call the perl interpreter
- ▶ the "use strict" and the "use warning" should always be used in modules
- ▶ always check function return values
- ▶ watch for external program failures in `$?`
- ▶ always check your input (including command line arguments)
- ▶ always have an else after a chain of elsif's (even when the else case is empty)
- ▶ always have a default after a chain of given's (even when the default case is empty)
- ▶ put commas at the end of lists so your program won't break if someone inserts another item at the end of the list.

Designers of a perl script generally assume that users will always supply reasonable input data. However, this is far from reality. A misunderstanding or typing error can often cause the user to enter something that was never intended. The programmer is responsible for taking all reasonable precautions. This technique is known as **defensive programming**. A defensive programmer will e.g.

1. Validate input wherever possible so that the user will be shown a useful error message when an error is made and be given the opportunity to re-enter the input.
2. The use of parameters instead of global variables is a form of defensive programming, as it helps to reduce side effects of changes in variables.

```
45 # An else after a chain of elsif-s  
if ( $number_s == 1 )  
{  
    &DoSomething( $number_s );  
}  
50 elsif ( $number_s > 1 )  
{  
    &DoSomethingElse( $number_s );  
}  
else  
55 {  
    #-- EMPTY  
}  
  
# Comma at the end of an array or hash  
60 my %computers_h = ( "aix"      => "134.94.100.100",  
                    "solaris" => "134.94.100.24",  
                    "www"     => "134.94.100.51",  
                    );
```

## Rule Perl\_015: Make regular expressions readable

### Instruction

- ▶ You can use comments and spaces in regular expressions to make them more readable, if you use the pattern modifier /x
- ▶ You can split a complex regular expression in parts and store them in variables packaged by qr (since Perl 5.6).

```
# OK
m/\w+: (\s+\w+)\s*\d+;/

# BETTER
m/\w+: (\s+ \w+) \s* \d+/x;

# PERFECT
m{
    \w+:      # Match a word and a colon.
    (        # (begin group)
        \s+   # Match one or more spaces
        \w+   # Match another word
    )        # (end group)
    \s*      # Match zero or more spaces
    \d+      # Match some digits
}x

# Usage of quoted regular expressions
my $gerZipCode_s = qr/[0-9]{5}/;
my $space_s      = qr/[ \t]+/;
my $city_s       = qr/\w+/;

my $example1_s = "70199 Stuttgart";
my $example2_s = "7019 Nowhere";
if ($example1_s =~ /$gerZipCode_s$space_s$city_s/)
{
    print "--> 1: STRIKE!\n";
}
if ($example2_s =~ /$gerZipCode_s$space_s$city_s/)
{
    print "--> 2: STRIKE!\n";
}

my $gerLocation_s = qr/$gerZipCode_s$space_s$city_s/;
if ($example1_s =~ /$gerLocation_s/)
{
    print "--> 3: STRIKE!\n";
}
```

## Rule Perl\_016: Make dereferenciation to references readable

### Instruction

- ▶ The usage of the arrow operator allows a more compact notation of dereferenciation.
- ▶ The storage of intermediate results in blocks makes complicate pointer constructs more readable.

Here are some examples of references of single arrays and hashes:

```
# References to single arrays and hashes
$hugo_pa = [ aa, bb, cc ];
```

```

05 $hugo_ph = {
    'Adam' => 'Eve',
    'Clyde' => 'Bonnie',
    };

10 # Dereferencing of single array elements (ok)
$elem_s = $$hugo_pa[0];

# Dereferencing of single array elements (better)
$elem_s = ${$hugo_pa}[0];
15 $elem_s = $hugo_pa->[0];

# Dereferencing of an array (ok)
@hugo_a = @$hugo_pa;

# Dereferencing of an array (better)
20 @hugo_a = @{$hugo_pa};

# Dereferencing of single hash elements (ok)
$elem_s = $$hugo_ph{"Clyde"};

# Dereferencing of single array elements (better)
25 $elem_s = ${$hugo_ph}{"Clyde"};
$elem_s = $hugo_ph->{"Clyde"};

# Dereferencing of a hash (ok)
30 %hugo_h = %$hugo_ph;

# Dereferencing of a hash (better)
%hugo_h = %{$hugo_ph};

35 Here are some examples of references of more complicate structures:

# References to hashes of hashes
%hoh_h = (
    "John" => {
40         "EMAIL" => "john@doe.com",
        "LOCATION" => "L.A.",
    },
    "Jane" => {
45         "MAID_NAME" => "Mueller",
        "EMAIL" => "jane@doe.com",
    }
);

$hoh_ph = {
    "John" => {
50         "EMAIL" => "john@doe.com",
        "LOCATION" => "L.A.",
    },
    "Jane" => {
55         "MAID_NAME" => "Mueller",
        "EMAIL" => "jane@doe.com",
    }
};

# Access to single hash element (ok, the only way)
$elem_s = $hoh_h{Jane}{MAID_NAME}; # Mueller

60 # Dereferencing of a single sub-hash
$elem_h = %{ $hoh_h{Jane} };

# Access to single hash element (ok)
$elem_s = ${$hoh_ph{Jane}}{MAID_NAME}; # Mueller
65 # Access to single hash element (better)

```

```

05 $elem_s = $hoh_ph->{John}->{LOCATION}; # L.A.
   $elem_s = $hoh_ph->{John}{LOCATION}; # L.A.

   # Dereferencing of a single sub-hash (ok)
   %elem_h = %{ $$hoh_ph{John} }; # %elem_h = ( EMAIL => "john@doe.com",
10                                     # "LOCATION" => "L.A.", )

   # Dereferencing of a single sub-array (better)
   %elem_h = %{ ${ $hoh_ph }{John} }; # ditto
   %elem_h = %{ $hoh_ph->{John} }; # ditto

15 # References to arrays of arrays
   @lol_a = (
           [1, 2],
           [3, 4]
           );

20 $lol_pa = [
           [1, 2],
           [3, 4]
           ];

25 # Access to a single array element
   $elem_s = $lol_a[1][0]; # '3'

   # Dereferencing of a single sub-array
30 @elem_a = @{ $lol_a[1] }; # @elem_a = (3, 4)

   # Access to a single array element (ok)
   $elem_s = ${ $$lol_pa[1] }[0]; # '3'

   # Access to array hash element (better)
35 $elem_s = $lol_pa->[1]->[0]; # '3'
   $elem_s = $lol_pa->[1][0]; # '3'

   # Dereferencing of a single sub-array (ok)
40 @elem_a = @{ $$lol_pa[1] }; # @elem_a = (3, 4)

   # Dereferencing of a single sub-array (better)
   @elem_a = @{ ${ $lol_pa }[1] }; # @elem_a = (3, 4)
   @elem_a = @{ $lol_pa->[1] }; # @elem_a = (3, 4)

```

45

## Rule Perl\_017: Handling of multi-line strings

**Instruction** For the definition of multi-line strings should preferred here documents.

50

55

60

65

70

```

$price_s = '$100';

$here1_s = << "EOF";
Hey
    The price is $price_s.
Bye
EOF

$here2_s = << 'EOF';
Hey
    The price is $price_s.
Bye
EOF

print $here1_s;
#
# Hey
#    The price is $100.

```



```
# Bye
#

print $here2_s;
#
# Hey
#   The price is $price_s.
# Bye
#
```

## Rule Perl\_018: Local overwriting of global variables

**Instruction** Avoid the overwriting of global variables and furthermore of Perl system variables.

If the overwriting is necessary, use a local declaration within its own block:

```
# Enabling whole-file mode
{
    local $/;
    $file_s = <FH>;
}
```

## 5.5 Templates

**Rule Perl\_019:**Removed, template file is available

## 5.6 Usage of perltidy

To format an existing perl script or module the tool perltidy can be used. With the following options, the indentation, the maximum line length and some style corrections can be set. The name of the generated formatted script is specified with option `-outfile`:

```
perltidy --indent-columns=2 --maximum-line-length=100 tplModule.pm --outfile=tplModule2.pm
```

The used options can also be summarized in a command file named `.perltidycrc`:

```
# This is a simple example of a .perltidycrc configuration file
--indent-columns=4
--maximum-line-length=120
```

Some options which can be useful are the following:

```
# This is a simple of a .perltidycrc configuration file
--indent-columns=4
--maximum-line-length=120

# Choice between "open braces left" and "open braces right"
--opening-brace-on-new-line
#--noopening-brace-on-new-line

# Choice between space before and after semicolons in for loop
--nospace-for-semicolon
#--space-for-semicolon

# Closing Side Comments
# could be helpful in some cases (debugging, maintenance,...)
--closing-side-comments
--closing-side-comment-interval=16
```

For a complete list of all possible options please have a look in [\[Document The Perltidy Home Page / Name: Perltidy / URL: http://perltidy.sourceforge.net/\]](http://perltidy.sourceforge.net/)

## 6 Rule Set: oAW Rules

Coding Guidelines for oAW scripts

### 6.1 General oAW rules

This chapter contains general rules for oAW scripting, they are applicable for all oAW file types.

#### Rule OAW\_General\_001: Line length

**Instruction** The line length of 120 characters shall not be exceeded.

#### Rule OAW\_General\_002: Comments

**Instruction** Comments shall be written in English language. Each function shall have a comment. Comments shall be used where needed to clarify blocks of scripting.

### 6.2 Model Workflow Environment

This chapter covers Model Workflow Environment (MWE) related topics, including properties files.

#### Rule OAW\_MWE\_001: Project global settings

**Instruction** .oaw files shall use globally defined values instead of local settings.

A few typesystem names are available as global properties which can be used commonly in all .oaw files. This will help to avoid rework of all files if the tool changes.

##### Global properties:

```
CheckComponent = com.bosch.oawtypesystem.actions.BCTCheckComponent
CodeGenerator = com.bosch.oawtypesystem.actions.BCTCodeGenerator
ForwardComponent = com.bosch.oawtypesystem.actions.BCTXtendComponent
GenerateIdComponent = com.bosch.oawtypesystem.actions.BCTXtendComponent
MetaModel = com.bosch.oawtypesystem.metamodel.cubec.CubecConfMetaModel
```

These properties used to be located in the file *bct.properties*, but this content will be provided by the tools in future. Hence the inclusion of *bct.properties* in the workflow will not be necessary anymore then.

##### Path settings:

In former rules, it was told to put path settings to a file like *oaw.properties*.

This file can be discarded and shall be removed from workflow as well, as the default path settings are now preconfigured by the tools.

The setting `<outlet path='_out'/>` is not needed anymore, and it is advised to remove it, as the default output folder might change in future.

Starting with tool release end of 2013 it will be possible to write logfiles to the default folder `_log/bct`.

To do this the outlet path needs to be configured as follows:

```
<outlet path='${LogDir}' />
```

Example using the above properties:

```
<workflow>

  <!-- Code generator for C header file -->
  <component class='${CodeGenerator}'>
    <metaModel class='${MetaModel}' />
    <!-- Name of file::function -->
    <expand value="BswM_Cfg_h::Generate FOR model" />
    <!-- output path is default -->
  </component>

  <!-- Code generator for C file -->
  <component class='${CodeGenerator}'>
    <metaModel class='${MetaModel}' />
    <!-- Name of file::function -->
    <expand value="BswM_Cfg_c::Generate FOR model" />
    <!-- output path is default -->
  </component>

  <!-- BSWMD generator -->
  <component class='${CodeGenerator}'>
    <metaModel class='${MetaModel}' />
    <!-- Name of file::function -->
    <expand value="BswM_Cfg_BSWMD_arxml::Generate FOR model" />
    <!-- output path is default -->
  </component>

  <!-- SWCD generator -->
  <component class='${CodeGenerator}'>
    <metaModel class='${MetaModel}' />
    <!-- Name of file::function -->
    <expand value="BswM_Cfg_SWCD_arxml::Generate FOR model" />
    <!-- output path is default -->
  </component>

  <!-- Report generator -->
  <component class='${CodeGenerator}'>
    <metaModel class='${MetaModel}' />
    <!-- Name of file::function -->
    <expand value="BswM_GenerateReport::Generate FOR model" />
    <!-- output path is log -->
    <outlet path='${LogDir}' />
  </component>
</workflow>
```

## Rule OAW\_MWE\_002: Properties

**Instruction** Component specific constant properties shall be stored in a file named <Comp>[\_<Sub>].properties

Properties can be used as kind of global setting across all oAW scripts of the component.

## 6.3 Configuration Validator files (\*.chk)

### Rule OAW\_Validator\_001: Validator

**Instruction** Each component shall validate its relevant configuration with a validator script after all data has been loaded and forwarded to it.

**Hint** Use BAMF to have the correct order of the workflow.

**Caution** Special rules for validator scripts will be defined in a future release of the guidelines document

## Rule OAW\_Validator\_002: Validator scripts follow Ext rules

**Instruction** Validation scripts are very similar to Xtend scripts and therefore they shall be written according to the same rules.

See rules of Extensions.

## 6.4 Extensions (Xtend files, \*.ext)

Extension files (\*.ext) can be used to keep extensions used by different generator templates and validator scripts.

### Rule OAW\_Xtend\_008: Do not use native Java Extensions

**Instruction** Java extensions shall not be used. There is no general support for native Java in code generators.

Since BCT is already using Java, and the configuration build process encapsulates the code generators, it is not allowed to use native Java code to avoid issues here.

Please contact the tool developers if you have special requirements where you don't find a proper solution with Xtend only.

### Rule OAW\_Xtend\_001: Standard file header

**Instruction** Every Xtend file shall be enclosed by a standard file header and footer comment for configuration management.

Xtend files shall have the same comments as C files:

Figure 30 Module Header for Xtend Files

```
/*<BASDKey>
*****
*
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2014. All rights reserved.
* The reproduction, distribution and utilization of this document as well as the communication of its contents to
* others without explicit authorization is prohibited. Offenders will be held liable for the payment of damages.
* All rights reserved in the event of the grant of a patent, utility model or design.
*
*****
* Administrative Information (automatically filled in)
* $Domain____:$
* $Namespace_:$
* $Class____:$
* $Name____:$
* $Variant____:$
* $Revision____:$
*****
</BASDKey>*/
```

The year in the file header represents the creation respectively the first release of the corresponding file. Therefore the year has not to be updated when an existing file is changed. But for new files the year shall be set to the current one.

Additional information that has to be inserted at the end of each file:

Figure 31 Comment block for SCM History Information

```
/*<BASDKey>
*****
* $History____:$
*****
</BASDKey>*/
```

## Rule OAW\_Xtend\_002: Includes and imports

**Instruction** Each ext and chk file which has functions that work on configuration data shall import first the bsw and then the module to allow access to its methods, object types and properties.

Example:

```
import bsw;  
import bsw::BswMModule;
```

## Rule OAW\_Xtend\_003: Aspect orientation

**Instruction** Whenever a function is using aspect orientation and may be overloaded by an aspect function, it shall have the string "Aspect" in its function name: <ModuleName>\_Aspect\_<FunctionName>.

Examples:

```
Boolean Dem_Aspect_IsPresent (CanSM this) :  
    false  
;
```

```
around *Dem_Aspect_IsPresent (CanSM this) :  
    ((Ecu)this.parent).dem != null  
;
```

**Caution** If Dem\_Aspect\_IsPresent changes (e.g. name change) in the original file, the whole feature will not work anymore unless the name is changed in all other components that use the aspect feature!

## Rule OAW\_Xtend\_004: Indentation

**Instruction** The first character of a function declaration and the closing character ; shall be placed at the begin of a new line. Script in between shall be indented by 4 blanks.

Indentation helps to enhance readability.

Example:

```
String GetString() :  
    "Hello World!"  
;
```

Not allowed:

```
String GetString() : "Hello World!";
```

## Rule OAW\_Xtend\_009: Line length

**Instruction** Long lines shall be avoided or split to enhance readability.

Extension lines can be split after a concatenation point (".") or using the "+" sign.

Example:

```
/*  
 * return a list of Mode Conditions with SchM Generic Mode Request Port.  
 */  
List BswM_Genutils_getAllSchMConditions (List conditions) :  
    conditions.select (e | e.bswMConditionMode.bswMModeRequestSource.bswMGenericRequest != null) .  
        select (e | e.bswMConditionValue.bswMModeDeclaration != null) .
```

```
select (e|e.bswMConditionValue.bswMModeDeclaration.bswMModeValueRef != null)
```

```
;
```

Not allowed:

```
/*
```

```
* return a list of Mode Conditions with SchM Generic Mode Request Port.
```

```
*/
```

```
List BswM_Genutils_getAllSchMConditions(List conditions) :
```

```
conditions.select(e| e.bswMConditionMode.bswMModeRequestSource.bswMGenericRequest != null).select(e
```

```
;
```

## Rule OAW\_Xtend\_005: Private functions

**Instruction** Functions that are not meant to be visible outside and that are only being used inside one ext file shall be declared private.

Example:

```
private String GetString() :
```

```
"Hello World!"
```

```
;
```

## Rule OAW\_Xtend\_006: Function names

**Instruction** Similar to the global visible elements in C-code the public functions in ext files shall have the prefix of their context (namespace).

Example: A generator extension defined for Dem that retrieves a part of the configuration:

```
pDemConfigSet Dem_getDemConfigSet(Dem this) :
```

```
this.demConfigSet
```

```
;
```

Example: A generator extension defined for non-Autosar component DemFifo:

```
String rba_DemFifo_getString() :
```

```
"Hello"
```

```
;
```

## Rule OAW\_Xtend\_007: ID Generator scripts

**Instruction** This rule has been superseded by **Rule OAW\_XpandIdGen\_001**: ID Generator follows Xpand and XTend rules

See the other rules of Xtend in the same chapter

# 6.5 Generator templates (Xpand, \*.xpt)

Xpand scripts are also known as templates.

## Rule OAW\_Xpand\_001: Standard file header

**Instruction** Every Xpand file shall be headed by a standard file header and footer.

Standard file header for Xpand templates:

Figure 32 Module Header for Xpand Files

```

«REM»
<BASDKey>
*****
*
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2014. All rights reserved.
* The reproduction, distribution and utilization of this document as well as the communication of its contents to
* others without explicit authorization is prohibited. Offenders will be held liable for the payment of damages.
* All rights reserved in the event of the grant of a patent, utility model or design.
*
*****
* Administrative Information (automatically filled in)
* $Domain____:$
* $Namespace_:$
* $Class____:$
* $Name____:$
* $Variant__:$
* $Revision__:$
*****
</BASDKey>
«ENDREM»

```

The year in the file header represents the creation respectively the first release of the corresponding file. Therefore the year has not to be updated when an existing file is changed. But for new files the year shall be set to the current one.

Additional information to be inserted at the end of each file:

Figure 33 Comment block for SCM History Information

```

«REM»
<BASDKey>
*****
* Administrative Information (automatically filled in)
* $History____:$
*****
</BASDKey>
«ENDREM»

```

## Rule OAW\_Xpand\_002: Includes and imports

**Instruction** Each Xpand template shall import first the bsw and then the module configuration itself to allow access to its methods, object types and properties:

```
«IMPORT bsw»
```

```
«IMPORT bsw::<<Modulename>Module»
```

Example:

```

«IMPORT bsw»
«IMPORT bsw::BswModule»

```

## Rule OAW\_Xpand\_003: Indentation

**Instruction** Paired keywords shall be indented at the same level, the starting « sign has to be at the same column position.

This is valid for all oAW Expand keywords like IF – ELSE – ENDIF, FOREACH – ENDFOREACH, LET – ENDLET, DEFINE – ENDDEFINE, FILE – ENDFILE, REM – ENDREM, PROTECT – ENDPROTECT, AROUND – ENдарOUND, etc.

Additional indentation shall be used, with 4 spaces (Tabs shall be replaced by 4 spaces) per indentation level, except where the indentation has negative effect on generated code.

Example:

```

«FOREACH ...»
  «IF ...»
    ...
  «ELSE»
    ...
  «ENDIF»
«ENDFOREACH»

```

**Hint** Generated files should be post-processed using beautifiers if available, then there is no need to take care for indentation and absolutely exact formatting in the generator template.

## Rule OAW\_Xpand\_004: Generated files rules

**Instruction** File generators shall follow the rules of the respective generated file types

E.g. generated C/H files shall follow the rules of C coding guidelines.

## Rule OAW\_Xpand\_005: Generated files formatting using beautifiers

**Instruction** Generated files should be post-processed by beautifiers if available

Currently code beautifiers exist for Java, XML and C++ code. Usage is under investigation and might become mandatory in a later step.

## Rule OAW\_Xpand\_006: One generated file per generator

**Instruction** In a Xpand template only one file shall be generated. The template shall not contain different file generators. However, if a component needs to generate several similar files, the code generator is allowed to generate more than one file if they are based on the same script, e.g. using a loop.

Example: for BswM\_Cfg.c there shall be BswM\_Cfg\_c.xpt and for BswM\_Cfg.h there shall be BswM\_Cfg\_h.xpt generator.

## Rule OAW\_Xpand\_007: Main generator function

**Instruction** The actual main generator function shall be named Generate.

Example:

```
«DEFINE Generate for Autosar »
```

## Rule OAW\_Xpand\_008: Removal of undesired newlines using – sign

**Instruction** Xpand allows to prevent the generation of a blank new line when a – is inserted at the end of the command bracket.

A space (blank) shall be added before the hyphen for better readability.

Example:

```
«IF this.varType == 0 -»  
#define MYMOD_VARTYPE (0)  
#define MYMOD_VARS_DISABLED  
«ELSE -»  
#define MYMOD_VARTYPE («this.varType.toInteger()»)  
#define MYMOD_VARS_ENABLED  
«ENDIF -»
```

## Rule OAW\_Xpand\_009: No time and no date in generated files

**Instruction** Time and date information is not allowed in generated files



Software integrators and project teams use to compare configurations and the output of different projects files. If the content of generated files is identical, date and time information would still lead to unnecessary and undesired differences.

## Rule OAW\_Xpand\_010: Generic copyright header for generated files

**Instruction** Generated files shall use a generic copyright information header without configuration management information.

For oAW scripting, a general copyright header has been defined. Previously it was available in the file *Copyright.ext*, this will be provided by the tools in future.

Usage:

```
«EXTENSION Copyright»  
«FILE "BswM_Cfg.c"-»  
«Copyright_getComment () »  
«ENDFILE»
```

## 6.6 ID Generator templates

ID Generator templates have additional rules to the Generator templates above

### Rule OAW\_XpandIdGen\_001: ID Generator follow Xpand and XTend rules

**Instruction** Id generators shall follow the rules of the respective templates rules for Xtend and XPand.

### Rule OAW\_XpandIdGen\_002: Separate Action

**Instruction** Id generators shall have a separate build action in the respective BAMF of the component

ID Generator shall run before code generation, therefore a separate action has to be defined and used in BAMF.

**Hint** GenerateId actions need to make proper usage of input and output data declarations in BAMF.

### Rule OAW\_XpandIdGen\_003: Avoidance of Set\* Methods

**Instruction** Ids shall be added as new items using forwarding mechanism. Set\* methods shall not be used anymore.

Set\* methods might lead to unintentional changes of the data. Forwarded data can be validated by the tool, e.g. multiplicity check.

Instead the new forwarding API (.fwd methods) shall be used.

## 6.7 Auxilliary rules

### Rule OAW\_Auxilliary\_001: Check for availability of a module

**Instruction** Before using data of a module or forwarding data to it, the presence of a module shall be checked.

This can be done e.g. with the following extension IsModuleExistent.

If a module has a valid EcucParamDef and a valid configuration (ECU Conf Navigator does not show [?] in front of the module, the extension will return true, otherwise it will return false.

```
/* * returns true if the specified module name is present in configuration tree (both EcucParamdef and o
Boolean IsModuleExistent(Autosar this, String moduleName):
    this.getAllModules().select(e|e.name == moduleName).name.size > 0
;
```

Usage in ext:

```
if (False != IsModuleExistent(Autosar, "FrTp") then { // do some stuff }
```

Usage in xpt:

```
<IF (IsModuleExistent(this, "FrTp")) != false) ->
#define USE_F RTP (1)
<ELSE ->
#define USE_F RTP (0)
<ENDIF ->
```

## Rule OAW\_Auxilliary\_002: Usage of Protected regions

**Instruction** Usage of protected regions is not recommended.

With CUBAS BSW, most of the code is either static or generated. User callouts and integration code have to reside in separate project specific files, that are not overwritten by code generation anyways.

Therefore it is not necessary to use the PROTECT and ENDPROTECT keywords in Xpand scripts.

## 7 Rule Set: Data Description

One important feature of embedded automotive software is the possibility to calibrate data while the software is running on the *ECU*. At that time, code compilation and linking is already finished. The fine-tuning of different values, *curves* and *maps* is done in real-time e.g. while the engine is running. In order to retrieve your *Calibration* and *Measurement data* in the *hex file* of a program, data specification in the earlier phases of software development is necessary. Data specification is inevitable if you want to calibrate and measure data at program runtime.

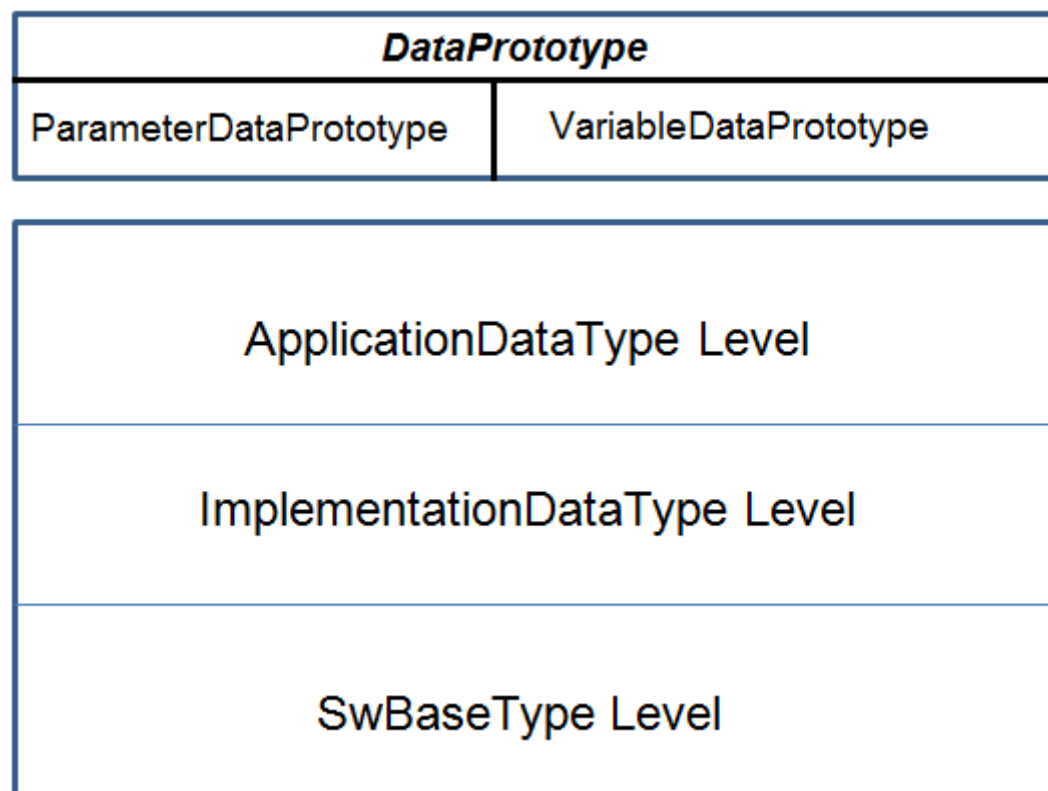
Data description of a BSW module is done preferably with a Basis Software Module Description ARXML file (BSWMD). As described in rule [\[BSW\\_Files\\_001\]](#) p. 88 the BSWMD file is one of the mandatory files of a BSW module.

For some specific cases a Software Component Template ARXML file (SWCD) is needed which is normally used in application software ASW (e.g. to describe AUTOSAR interfaces and to do the mapping of runnable entities). But this file is not in the main focus of a BSW module and is not described within the BSW coding guidelines. For more details about the SWCD ARXML file and the handling of its use cases take a look at the ArCaDe guideline (AUTOSAR Coding and Data Description Guideline) which describes all use cases of the application software ASW.

### 7.1 Data Types

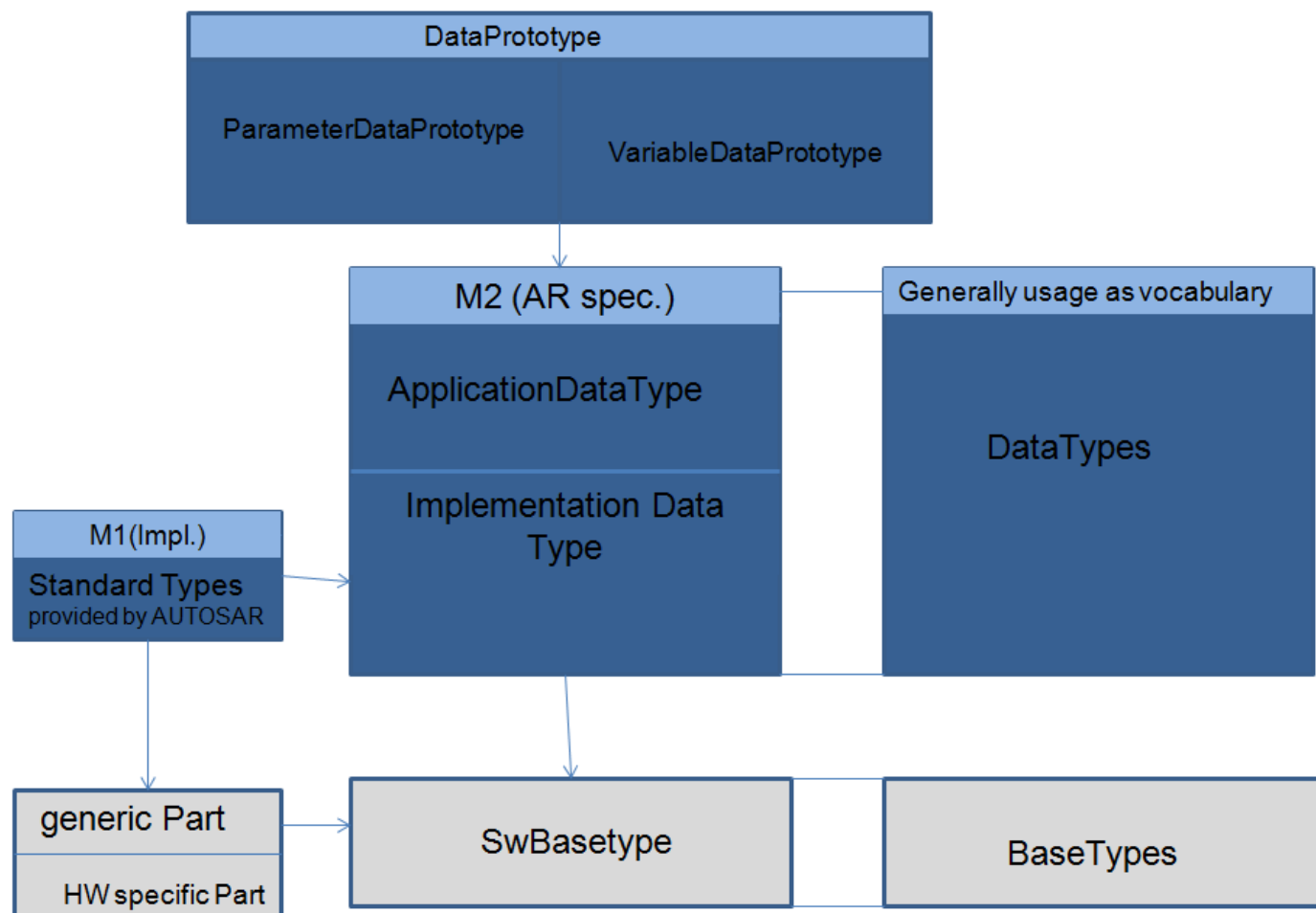
In AUTOSAR the meaning of data type is widely extended. It includes not only the C-data types but may also contain attributes which define a semantical meaning. The (C-)language implementation properties shall be widely hidden from the functional developers. Therefore three layers of data type abstraction were introduced in AUTOSAR R4.0.

Figure 34 Data Types 4.x



The following diagram illustrates the relation of types between AUTOSAR specification (M2 definitions which provides the meta model) and its instantiation, the implementation level for platform and standard types.

Figure 35 Overview of AUTOSAR Types



### 7.1.1 ApplicationDataTypes

*ApplicationDataType* defines a data type from the application point of view. This level is the common level at which application SW components specify a data type or prototype. It should be used whenever something "physical" is at stake. An *ApplicationDataType* represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianness, etc. It should be possible to model the application level aspects of a VFB system by using *ApplicationDataTypes* only.

The Application Data Level includes among other things the numerical range of values, the data structure as well as the physical semantics. The data semantics is important for a unique interpretation of data in the application software and in Measurement and Calibration systems.

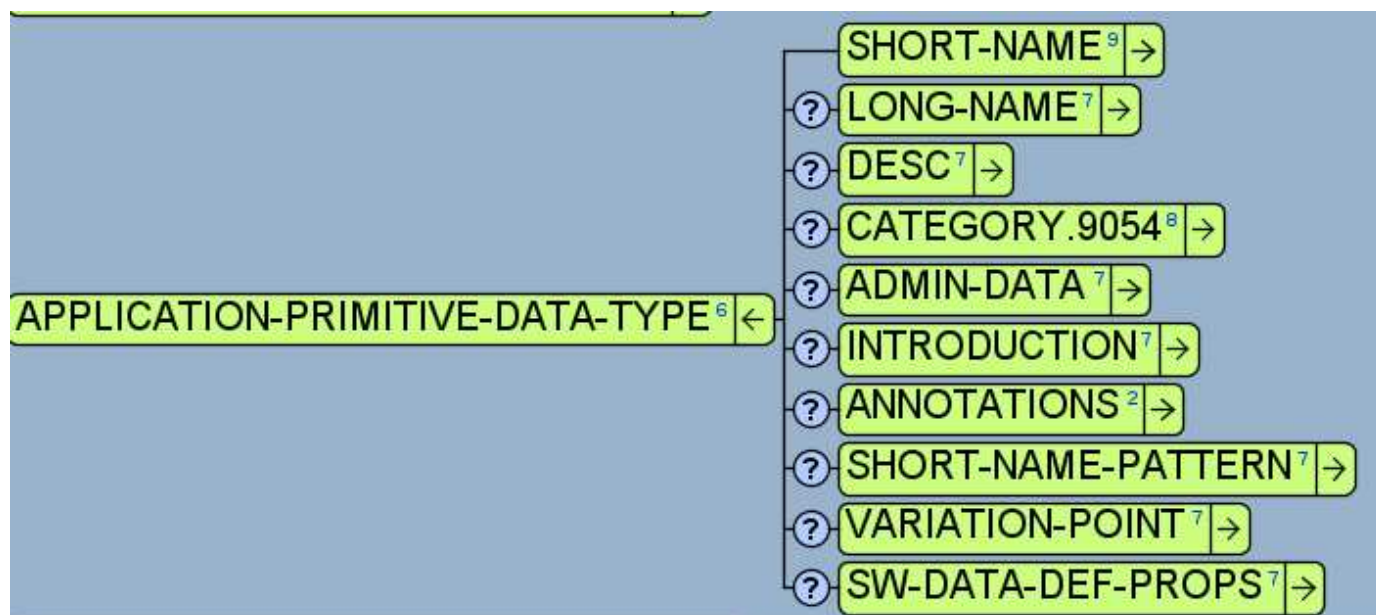
**Instruction** If two or more data prototypes have the same set of values at their application model, then a reuse of respective *ApplicationDataType* is allowed.

The *ApplicationDataTypes* are characterized by their categories and their data definition properties. For a given category only a limited set of attributes of the SW data definition properties makes sense.

*ApplicationDataType* can be standardised by AUTOSAR. Developer shall use this pre-defined *ApplicationDataType* and create the *ApplicationDataType* if and only if one is not defined. The Primitive *ApplicationDataType* can be used for basis for complex types.

Figure 6 gives the schematic representation of the Primitive *ApplicationDataType*. *ShortName* and *Category* **Category** are mandatory attributes.

Figure 36 Schematic Representation of ApplicationDataType\_Primitive



*ApplicationDataTypes* can be classified as

- Primitive *ApplicationDataType*
  - 1. VALUE
  - 2. BOOLEAN
  - 3. STRING
  - ASW specific Primitive *ApplicationDataType*
    - 1. CURVE
    - 2. MAP
    - 3. COM\_AXIS
    - 4. RES\_AXIS
- Complex *ApplicationDataType*
  - 1. ARRAY
  - 2. STRUCTURE

**Hint** The categories listed above are supported by Ara2L Gen

### 7.1.1.1 Primitive *ApplicationDataTypes*

#### 7.1.1.1.1 *ApplicationDataType\_Value*

**Example:**

```
<AR-PACKAGE>
  <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-PRIMITIVE-DATA-TYPE>
      <SHORT-NAME>ApplicationDataType_Primitive</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
```

```

05      <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
          <COMPU-METHOD-REF DEST="COMPU-METHOD">
10            <!-- Refer to a CompuMethod -->
          </COMPU-METHOD-REF>
          <DATA-CONSTR-REF DEST="DATA-CONSTR">
            <!-- Refer to a DataConstraint -->
          </DATA-CONSTR-REF>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
  </APPLICATION-PRIMITIVE-DATA-TYPE>
</ELEMENTS>
20 </AR-PACKAGE>

```

### 7.1.1.1.1 Rules

Rule Data\_003: Removed.

### Rule Data\_114:Data type for Application Primitive Data Type

#### Instruction

**Scope:**AUTOSAR 4.x

Primitive ApplicationDataTypes shall be mapped to Autosar specified ImplementationDataTypes (uint 8, sint 16 etc.) which are available as central elements.

**Hint** This would lead to creation of very less number of Implementation Data Type as Implementation Data Type can be reused. It would be compatible with the data types used in Service-Libs routines

#### Hint

**Scope:** DGS

This was also defined by ARAP as the DP0050A-2

### Rule Data\_004a: Defining *ApplicationDataType*

#### Instruction

**Scope:** AUTOSAR 4.x

**1.** Some *ApplicationDataType* are standardized by AUTOSAR and are provided as central elements.

**2.** All other *ApplicationDataType* are Module/Component specific.

Component Developer should reuse the *ApplicationDataType* defined for the component as far as possible. Referring to *ApplicationDataType* defined by other Software Component shall not be used.

This rule is a more detailed definition of the rule *CEL\_061*.

Finding of Approval Process: Product-line specific, centrally defined Application Data Types (not standardized by AUTOSAR) must be permissible, too. Finding will be solved for next version of Data Description.(V1.6)

Rule Data\_119: DataConstr are mandatory for *ApplicationDataType* that are mapped to Float

#### Instruction

**Scope:**AUTOSAR 4.X

ApplicationDataType shall have a DataConstr, if it is mapped to ImplementationDataType of type float.

This is because A2L-Generator<sup>7</sup> needs the lower limit and upper limit. It is possible to fetch DataConstr from Implementation Data Type of type Integer, if DataConstr are not available at *ApplicationDataType* level. This is not possible in case of Implementation Data Type of float type.

Rule Data\_142: Definition of *SwDataDefProps* for *ApplicationPrimitiveDataType*

**Instruction** *SwDataDefProps* shall be a mandatory element for *ApplicationPrimitiveDataType* as mandatory element similar to *Short-Name* and *Category*.

### 7.1.1.1.2 ApplicationDataType\_Boolean

#### Example:

```
<AR-PACKAGE>
  <SHORT-NAME>ApplicationDataTypes_Boolean</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-PRIMITIVE-DATA-TYPE>
      <SHORT-NAME>ApplicationDataType_Primitive</SHORT-NAME>
      <CATEGORY>BOOLEAN</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
            <COMPU-METHOD-REF DEST="COMPU-METHOD">
              <!-- Refer to a CompuMethod -->
            </COMPU-METHOD-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </APPLICATION-PRIMITIVE-DATA-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

#### 7.1.1.1.2.1 Rules

tbd

### 7.1.1.1.3 ApplicationDataType\_String

tbd

#### 7.1.1.1.3.1 Rules

Rule Data\_130:Support for *ApplicationDataType* of Category STRING

#### Instruction

**Scope:**AUTOSAR 4.X

<sup>7</sup> tool for creating A2Lsnippet from McSupport file



Support for *ApplicationDataType* Category *STRING* is not yet planned in AR\_A2L Gen. As such developer should not use these categories in their development.

## 7.1.1.2 Complex Application DataType

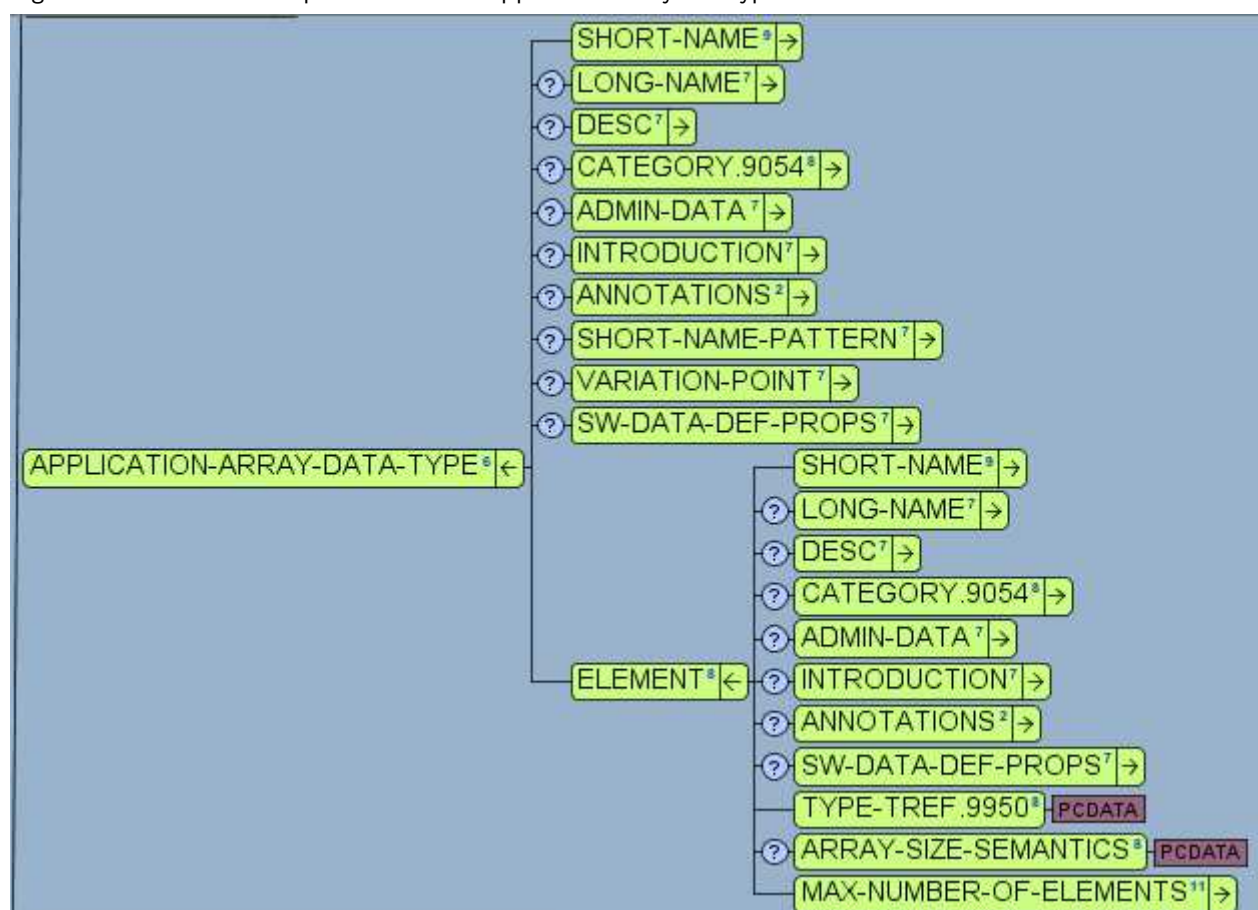
### 7.1.1.2.1 ApplicationDataType\_Array

For describing the type of an array you shall define an *ApplicationArrayDataType*. An *ApplicationArrayDataType* describes the whole array as "parent" and the subelements of the *ARRAY* often known as "children" are enclosed with in *Element*. The *Category* of the parent object shall be set to *ARRAY*. The *Category* of the child element is set to *VALUE* if it is not a nested array. All the child elements within an *ARRAY* shall belong to same *Category*. It is also possible to describe a nested array. A nested array may be *ARRAY* of *ARRAY*s or *ARRAY* of *STRUCTURES*.

Surprisingly the array size has to be described as property of the child, using *MaxNumberOfElements*.

*ArraySizeSemantics* and *MaxNumberOfElements* as mandatory attribute. *Array-Size-Semantics* defines whether the array is fixed size *FIXED-SIZE* or can be varied *VARIABLE-SIZE* and *MaxNumberOfElements* defines the maximum elements that an Array can contain. The properties defined within *Element* attribute are applicable to all the children of an Array while the properties defined for *ApplicationArrayDataType* are applicable for whole Array

Figure 37 Schematic Representation of ApplicationArrayDataType



#### Example:

```
<APPLICATION-ARRAY-DATA-TYPE>
  <SHORT-NAME>ApplicationDataType_Array</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
```



```

    </SW-DATA-DEF-PROPS-CONDITIONAL>
  </SW-DATA-DEF-PROPS-VARIANTS>
</SW-DATA-DEF-PROPS>
<ELEMENT>
  <SHORT-NAME>ApplicationDataType_Array_Element</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    <!--Refer to Application Primitive Data Type -->
  </TREF>
  <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
  <MAX-NUMBER-OF-ELEMENTS>3</MAX-NUMBER-OF-ELEMENTS>
</ELEMENT>
</APPLICATION-ARRAY-DATA-TYPE>

```

### 7.1.1.2.1.1 Rules

#### Rule Data\_140: Defining Array Size

##### Instruction

**Scope:**AUTOSAR 4.X

*MaxNumberOfElements* describes the size of ARRAY, as such the attribute *MaxNumberOfElements* shall be defined for *Application-Array-Data-Type*

*ArraySizeSemantics* should not be defined in the *ApplicationArrayDataType* (Could be defined in the *Implementation-DataType* )

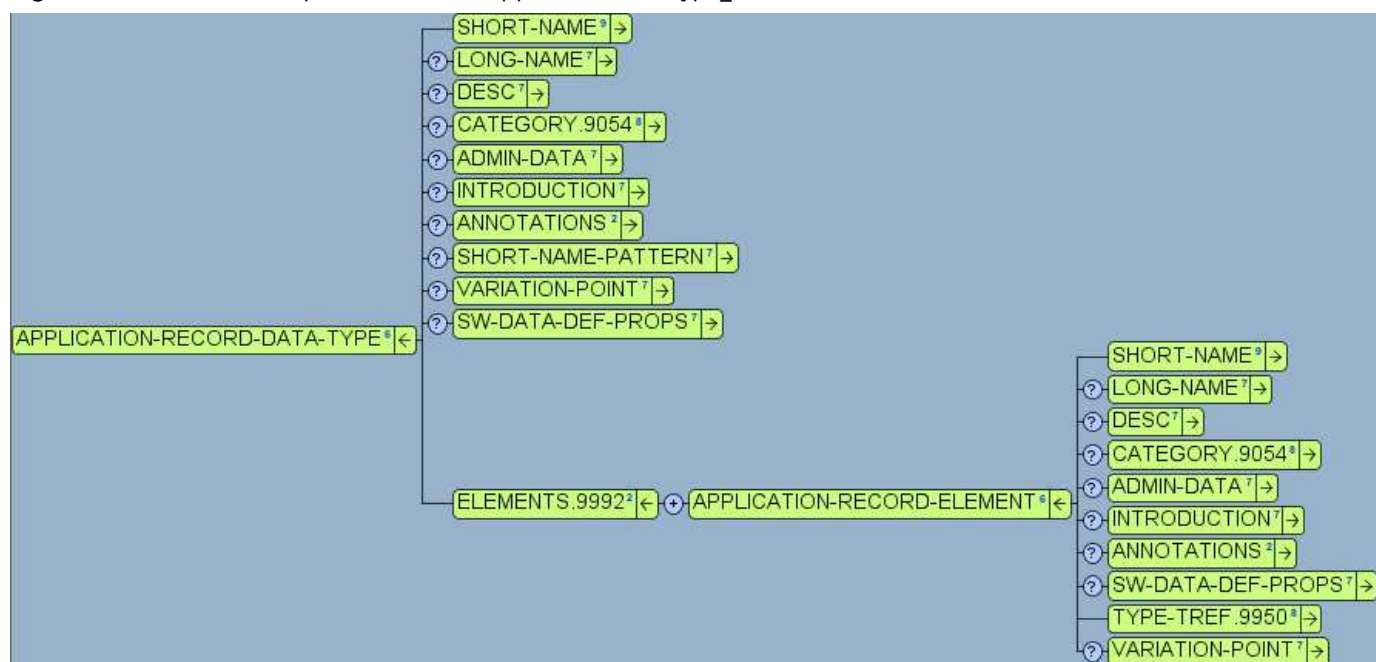
*MaxNumberOfElements* and *ArraySizeSemantics* are defined within the *Element* attribute and is applicable to all the children of an array.

### 7.1.1.2.2 ApplicationDataType\_Structure

*ApplicationRecordDataType* " is the Element used to represent a Structure. The attributes *Category* and *Element* are mandatory. Category shall be set to *STRUCTURE* The attribute *Element* is used to represent children of Structure. An Element attribute shall contain *ApplicationRecordElement* . Each *ApplicationRecordElement* shall represent a child of the structure.

**Instruction** The order of defining the child elements of the *STRUCTURE* also matters.

Figure 38 Schematic Representation of ApplicationDataType\_Structure

**Example:**

```

<APPLICATION-RECORD-DATA-TYPE>
  <SHORT-NAME>ApplicationDataType_Struct</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <ELEMENTS>
    <APPLICATION-RECORD-ELEMENT>
      <SHORT-NAME>ApplicationRecordElement_01</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
        <!-- Refer to the Snippet of an ApplicationPrimitiveDataType -->
      </TYPE-TREF>
    </APPLICATION-RECORD-ELEMENT>
  </ELEMENTS>
</APPLICATION-RECORD-DATA-TYPE>

```

**Hint** Structures can be nested. Nesting shall not only contain Primitive ApplicationDataTypes but also Complex ApplicationDataTypes.

**7.1.1.2.3 Rules**

**Rule Data\_123:** SubElements in Structures shall have the same names in ApplicationDataType and ImplementationDataType

**Instruction**

**Scope:**AUTOSAR 4.X

SubElements in Structures shall have the same names in ApplicationDataType and ImplementationDataType

This is required since RTEgen uses the SubElement names of implementation type for creating .c/.h but the SubElements of *ApplicationDataType* for creating McSupport. A2Lgen takes the names in McSupport for searching the address in .elf – and will fail if the names differ.

## Rule Data\_141: Definition of *SwDataDefProps* at SubElement Level

**Instruction** The *SwDataDefProps* of the *Elements* of an *ApplicationArrayDataType* or *ApplicationRecordDataType* shall be defined by the *ApplicationDataType* referenced by the *Element* and not by defining *SwDataDefProps* on *Elements* level itself.

The definition of *SwDataDefProps* at parent element shall contain *SwCalibrationAccess*.

### 7.1.2 *ImplementationDataType*

The concept of an *ImplementationDataType* has been introduced to optimize the formal support for data type handling on the implementation level. That is, an *ImplementationDataType* conceptually corresponds to the level of (C) source code. For example, *ImplementationDataTypes* have a direct impact on the contract of a software-component and the RTE.

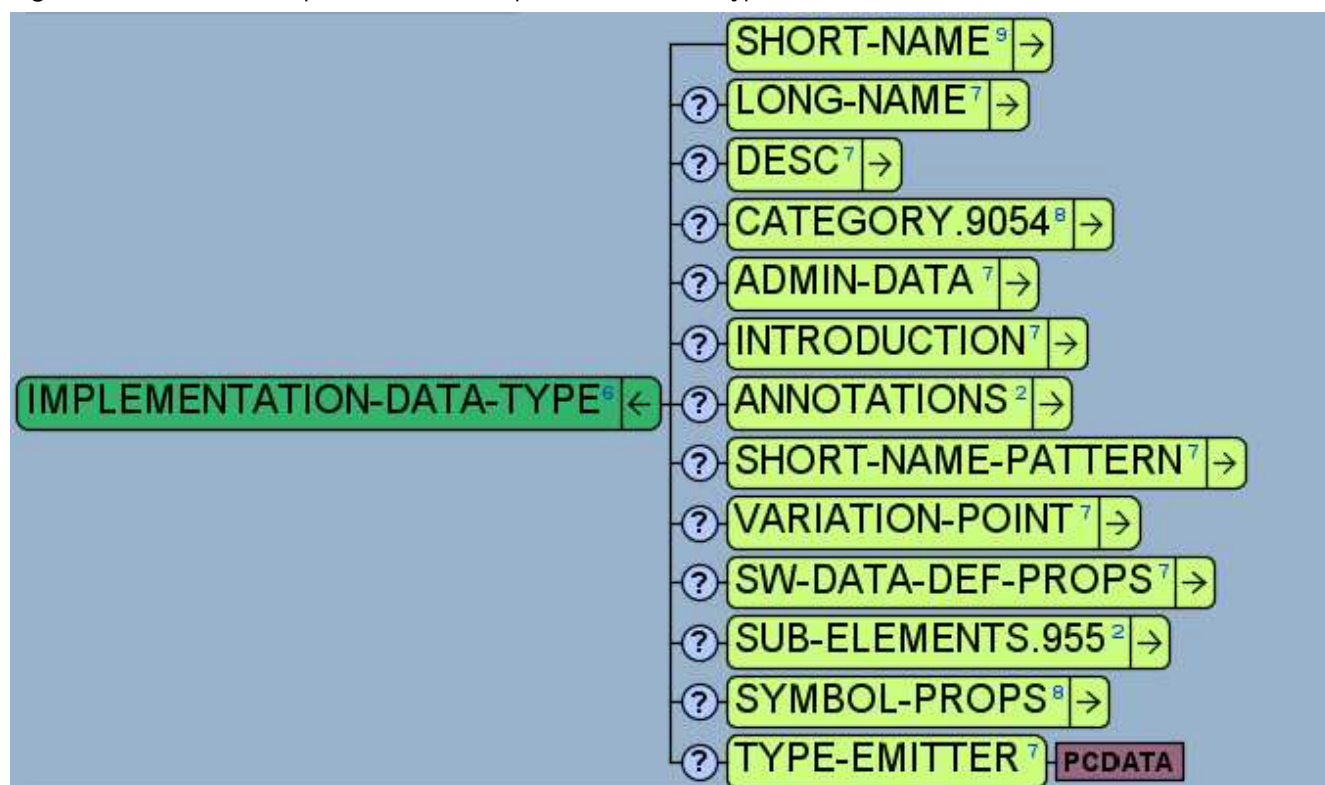
There are several use cases for this level in AUTOSAR:

1. First of all, the Implementation Data level can be used in the description of interfaces, and data (e.g. debug data) within the basic software.
2. *ImplementationDataTypes* should also be used to describe the interfaces of libraries which operate on a purely numerical level.
3. *ImplementationDataType* is also used for the description of interfaces between software-components and the basic software (namely AUTOSAR Services), because these typically cover implementation aspects only.

Implementation Data Types can have the following categories.

1. VALUE (Primitive)
2. ARRAY(Composite)
3. STRUCTURE(Composite)

Figure 39 Schematic Representation of ImplementationDataType



### 7.1.2.1 ImplementationDataType\_Value

#### Example:

```

<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>ImplementationDataType_Value</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
          <!-- Refer to a PlatformImplementationDataType (eg.,uint8, sint16..) -->
        </IMPLEMENTATION-DATA-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>

```

#### 7.1.2.1.1 Rules

tbd

#### Rule Data\_004b: ImplementationDataType

##### Instruction

**Scope:** AUTOSAR 4.x

1. Primitive Implementation data types (e.g. sint16) shall not be defined locally as they are always central and Standardized and delivered by AUTOSAR.
2. Complex Implementation data types for Maps, Curves etc, are always Product Line specific and are delivered as Central Elements. They are delivered as Central Elements of this Product Line. It is planned to have an AUTOSAR

Standardized implementation data types for these objects in a future version. For further details refer to [Document "Central Elements" \[A\\_CEL\]](#)

3. Other Complex Implementation data types (e.g. Structures) are defined as Module/Component specific in BSWM-D/SWCD.

4. ImplementationDataTypes referencing TEXTTABLE CompuMethods shall be module specific.

5. ---This point was removed based on the finding of Approval process.---

Finding of Approval Process: product-line specific, centrally defined enums should be permissible, too. Finding will be solved for next version of Data Description.(V1.6)

## Rule Data\_050: Reuse of ImplementationDataTypes

### Instruction

**Scope:** AR4.x

The reuse of ImplementationDataTypes is encouraged. Developer shall reuse the *ImplementationDataType* that are provided centrally or create module specific *ImplementationDataType*. *ImplementationDataType* defined for one specific module shall not be reused.

This implies that there should be no 1:1 mapping between application and implementation data types.

## Rule Data\_051: Reference to an ImplementationDataType

### Instruction

**Scope:** AR4.x

*ImplementationDataType* shall not to be referenced directly by a *DataProtoType*, but a *DataProtoType* shall refer to an *ApplicationDataType* which in turn is mapped to *ImplementationDataType*

## 7.1.2.2 ImplementationDataType\_Array

### Example:

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>ImplementationDataType_Array</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>ImplementationDataType_Element</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <ARRAY-SIZE>50</ARRAY-SIZE>
      <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <BASE-TYPE-REF DEST="SW-BASE-TYPE">
              <!-- Refer to a BaseType -->
            </BASE-TYPE-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

### 7.1.2.2.1 Rules

#### Rule Data\_139: Usage of *ArraySize* and *ArraySizeSemantics*

##### Instruction

**Scope:** AR4.x

ImplementationDataTypes shall have *ArraySize* defined in *ImplementationDataTypeElement* which defines the size of Array.

*ArraySizeSemantics* shall be FIXED-SIZE.

### 7.1.2.3 ImplementationDataType\_Structure

#### ImplementationDataType\_Structure

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>ImplementationDataType_Structure</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>ImplementationDataTypeElement</SHORT-NAME>
      <CATEGORY>TYPE_REFERENCE</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
              <!-- Refer to an ImplementationDataType -->
            </IMPLEMENTATION-DATA-TYPE-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

### 7.1.2.3.1 Rules

tbd

## 7.1.3 Data Type Mapping

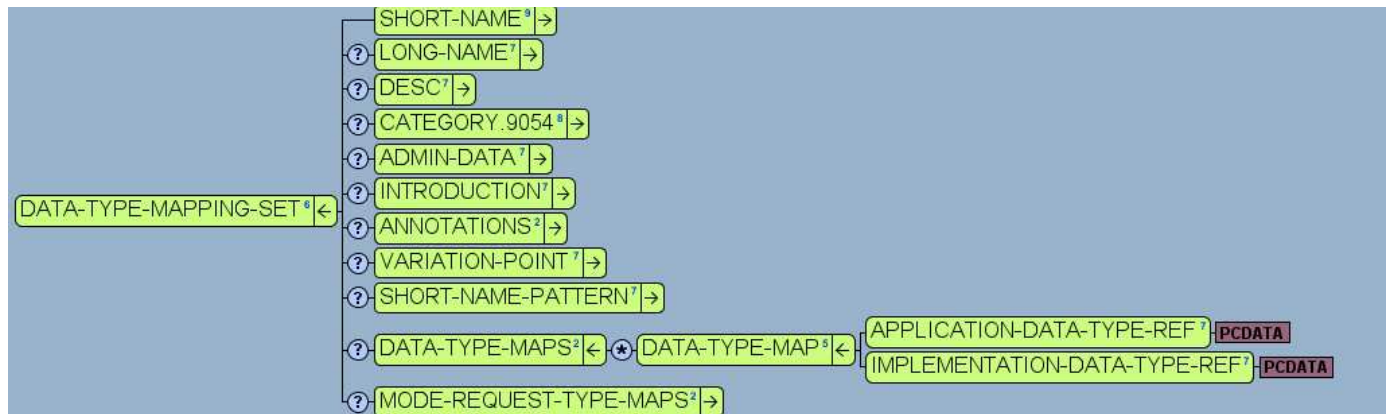
This class represents the relationship between *ApplicationDataType* and its implementing *ImplementationDatatype*.

This is supported by the meta-class *DataTypeMap* by which an *ApplicationDataType* and an *ImplementationDataType* can be mapped to each other in order to describe both aspects of one Data Element.

In order to set up a valid *DataTypeMap* between an *ApplicationDataType* and an *ImplementationDataType* the two types must be compatible. Of course, if *ImplementationDataTypes* are derived from existing *ApplicationDataTypes* it is expected that they will be automatically compatible.

Furthermore, the various mappings are aggregated in a container *DataTypeMappingSet* for easier maintenance in artifacts.

Figure 40 Schematic Representation of Data Type Mapping



## DataTypeMappingSet

### Example:

```

<DATA-TYPE-MAPPING-SET>
  <SHORT-NAME>DataTypeMappingSet</SHORT-NAME>
  <DATA-TYPE-MAPS>
    <DATA-TYPE-MAP>
      <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
        <!-- Refer to an ApplicationPrimitiveDataType -->
      </APPLICATION-DATA-TYPE-REF>
      <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
        <!-- Refer to an ImplementationDataType -->
      </IMPLEMENTATION-DATA-TYPE-REF>
    </DATA-TYPE-MAP>
    <DATA-TYPE-MAP>
      <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
        <!-- Refer to an ApplicationPrimitiveDataType -->
      </APPLICATION-DATA-TYPE-REF>
      <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
        <!-- Refer to an ImplementationDataType -->
      </IMPLEMENTATION-DATA-TYPE-REF>
    </DATA-TYPE-MAP>
  </DATA-TYPE-MAPS>
</DATA-TYPE-MAPPING-SET>

```

## 7.1.3.1 Rules

### Rule Data\_127: Definition of DataTypeMappingSet

#### Instruction

**Scope:** AUTOSAR 4.x

DataTypeMappingSet may be defined centrally or decentral.

- If the referenced ApplicationDataType and the referenced ImplementationDataType are defined centrally, then DataTypeMappingSet is defined in the same AUTOSAR package as the ApplicationDataType
- Mapping between ApplicationDataType and ImplementationDataType shall be defined locally (Component/ Module Specific Mapping) if the ApplicationDataType and ImplementationDataType are also defined locally.

This rule satisfies CEL\_064 and can be referenced [Document "Central Elements" \[A\\_CEL\]](#)



## Rule Data\_129: Mapping Between *ApplicationDataTypes* and Implementation Data Types

### Instruction

**Scope:** AUTOSAR 4.x

For all *ApplicationDataTypes* defined by a component corresponding mapping to *ImplementationDataType* shall also be specified in the same component.

For *ApplicationDataTypes* defined as central elements, their corresponding mapping to an *ImplementationDataType* will also be defined as central elements.

## 7.2 DataPrototypes

A Data prototypes implement a role of data type. This means to say a data prototype represents the implementation of a data type (is-of-type of relation to data type) within the definition of another data type. For example there are "typed" data object declared within a software component or a port interface (Referenced from [\[TPS\\_SWCT\]](#) ).

Data Element represents a reference to a Parameter/Variable within AUTOSAR which can be a local parameter, Autosar parameter, AUTOSAR parameter in implementation data type, local variable, AUTOSAR variable, or an AUTOSAR variable in implementation data type (Referenced from [\[TPS\\_SWCT\]](#) ).

Data Elements can be of the following Types :

1. Variable Data Prototype. Details can be found at [Chapter \[Measurements\]](#)
2. Parameter Data Prototype. Details can be found at [Chapter \[CalParam\]](#)

### 7.2.1 ParameterDataPrototype (Calibration Parameters)

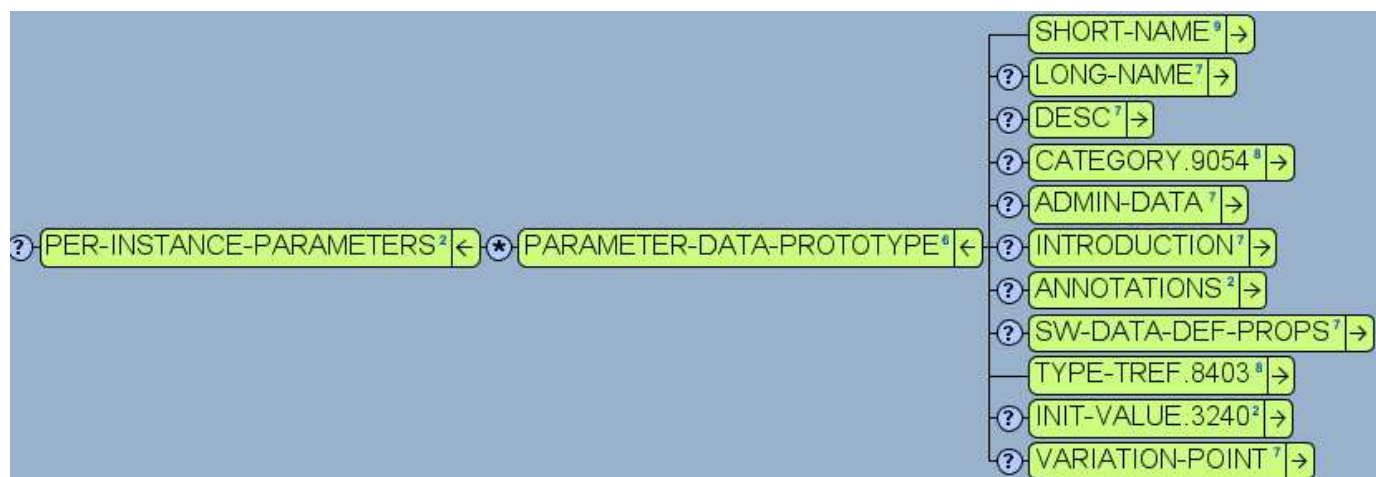
*Calibration* is the adjustment of parameters of sw components realizing the control functionality (namely parameters of AUTOSAR SWC's, *ECU* abstraction or Complex Drivers).

Most of the application software components can be calibrated. However basic software components also provides the possibility to calibrate parameters. Calibration is always done at post-build time. Used techniques to set calibration data include end-of-line programming, garage programming and adaptive calibration (e.g. in the case of anti-pinch protection for power window).

A calibration parameter can be used in the context of *InternalBehavior* as well as in the context of *ParameterInterface*. This section describes the different ways of using the calibration parameter. (Referenced from [Document "Software Component Template" \[TPS\\_SWCT\]](#) )



Figure 41 Schematic Representation of Parameter Data Prototype



### 7.2.1.1 ParameterDataPrototype of Category Value

#### Example:

```

<PARAMETER-DATA-PROTOTYPE>
  <SHORT-NAME>ParameterDataPrototype_Value</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
        <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    <!-- Refer to the Snippet of an ApplicationPrimitiveDataType -->
  </TYPE-TREF>
</PARAMETER-DATA-PROTOTYPE>

```

#### 7.2.1.1.1 Rules

##### Rule Data\_009: Usage of Shared Calibration Parameters and Per Instance Calibration Parameters

#### Instruction

**Scope:** . Applicable for AUTOSAR 4.x

ApplicationSoftwareComponent developer shall decide upon when to use Shared Calibration Parameters and when to use PerInstance Calibration based on the usage. However in Basic Software, PerInstance Parameter shall be used.

In case of Shared Calibration Parameters, calibration value in the ParameterDataPrototype will be shared among all SwComponentPrototypes

In case of PerInstance Parameters, calibration value in the ParameterDataPrototype will be specific for each instance of SwComponentPrototypes.

#### Example:

```

<APPLICATION-SOFTWARE-COMPONENT-TYPE>
  <SHORT-NAME><!--Short Name for Appln Software Component --></SHORT-NAME>
  <PORTS>
    <P-PORT-PROTOTYPE>
      ...
    </P-PORT-PROTOTYPE>

```

```

05  </PORTS>
    <INTERNAL-BEHAVIOR>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME><!--Short Name for InternalBehavior --></SHORT-NAME>
      ...
10  <SHARED-PARAMETERS>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME><!--Short Name for PDP --></SHORT-NAME>
      <CATEGORY>MAP</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
15      <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
20    </SW-DATA-DEF-PROPS>
  </SHARED-PARAMETERS>
</PARAMETER-DATA-PROTOTYPE>
  ...
</SWC-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIOR>
<APPLICATION-SOFTWARE-COMPONENT-TYPE>
Example:
<APPLICATION-SOFTWARE-COMPONENT-TYPE>
  <SHORT-NAME><!--Short Name for Appln Software Component --></SHORT-NAME>
30  <PORTS>
    <P-PORT-PROTOTYPE>
      ...
    </P-PORT-PROTOTYPE>
  </PORTS>
35  <INTERNAL-BEHAVIOR>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME><!--Short Name for InternalBehavior --></SHORT-NAME>
      ...
40  <PER-INSTANCE-PARAMETERS>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME><!--Short Name for PDP --></SHORT-NAME>
      <CATEGORY>CURVE</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
45      <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
50  </PER-INSTANCE-PARAMETERS>
</PARAMETER-DATA-PROTOTYPE>
  ...
</SWC-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIOR>
55  <APPLICATION-SOFTWARE-COMPONENT-TYPE>

```

## Rule Data\_010: Removed

## Rule Data\_011: Sharing Calibration Parameters between instances of different " SwComponent-Type "

### Instruction

**Scope:** Applicable for AUTOSAR 4.x

- ▶ For the `Calibration` parameters to be visible (shared) in other `SwComponentTypes`, a dedicated `ParameterSwComponentType` that inherits from `SwComponentType` has to be used as a `SwComponentPrototype` within a `CompositionSwComponentType`.
- ▶ The `ParameterSwComponentType` has no `InternalBehavior` and employs exclusively `PPortPrototypes` of type `ParameterInterface`.
- ▶ Every `SwComponentType` requiring access to shared Calibration Parameters will have an `RPortPrototype` typed by a `ParameterInterface`.

**Hint** A `ConnectorPrototype` will only be valid if the referenced `RPortPrototype` and `PPortPrototype` are typed by a compatible interface.

#### Example:

```
<PARAMETER-SW-COMPONENT-TYPE>
<SHORT-NAME><!--Short Name for Parameter Software Component --></SHORT-NAME>
<PORTS>
  <P-PORT-PROTOTYPE>
    <SHORT-NAME><!--Short Name for PortPrototype --></SHORT-NAME>
    <PROVIDED-INTERFACE-TREF DEST="PARAMETER-INTERFACE">
      /RB/PT/Calprm_Hugo/IF_Anton_C
    </PROVIDED-INTERFACE-TREF>
  </P-PORT-PROTOTYPE>
</PORTS>
</PARAMETER-SW-COMPONENT-TYPE>

<PARAMETER-INTERFACE>
<SHORT-NAME><!--Short Name for Port Interface --></SHORT-NAME>
<PARAMETERS>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME><!--Short Name for PDP --></SHORT-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
  </PARAMETER-DATA-PROTOTYPE>
</PARAMETERS>
</PARAMETER-INTERFACE>

<APPLICATION-SOFTWARE-COMPONENT-TYPE>
<SHORT-NAME><!--Short Name for Appln Software --></SHORT-NAME>
<PORTS>
  <R-PORT-PROTOTYPE>
    <SHORT-NAME><!--Short Name for Port --></SHORT-NAME>
    <REQUIRED-INTERFACE-TREF DEST="PARAMETER-INTERFACE">
      /RB/PT/Calprm_Hugo/IF_Anton_C
    </REQUIRED-INTERFACE-TREF>
    ...
  </R-PORT-PROTOTYPE>
</PORTS>
</APPLICATION-SOFTWARE-COMPONENT-TYPE>
```

## Rule Data\_031: Initial Value for ParameterDataPrototypes

### Instruction

**Scope:** AUTOSAR 4.x

Every ParameterDataPrototype shall have an initial value specified . This value shall be an implementation based one.

### 7.2.1.2 ParameterDataPrototype of Category Array

#### Example:

```
<PARAMETER-DATA-PROTOTYPE>
  <SHORT-NAME>ParameterDataPrototype_Array</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
        <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
    <!-- Refer to the Snippet of an ApplicationArrayDataType -->
  </TYPE-TREF>
</PARAMETER-DATA-PROTOTYPE>
```

#### 7.2.1.2.1 Rules

tbd

### 7.2.1.3 ParameterDataPrototype of Category Structure

#### Example:

```
<PARAMETER-DATA-PROTOTYPE>
  <SHORT-NAME>ParameterDataPrototype_Struct</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
        <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
    <!-- Refer to an ApplicationRecordDataType -->
  </TYPE-TREF>
</PARAMETER-DATA-PROTOTYPE>
```

#### 7.2.1.3.1 Rules

tbd

### 7.2.1.4 ApplicationSwComponentType Vs ParameterSwComponentType

ApplicationSwComponentType The ApplicationSwComponentType is used to represent the application software.

**ParameterSwComponentType** The **ParameterSwComponentType** defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected **SwComponentPrototypes**.

The following table describes the comparison between **ApplicationSwComponentType** and **ParameterSwComponentType** in the context of calibration and measurement.

Table 29 **ApplicationSwComponentType** Vs **ParameterSwComponentType**

<b>ApplicationSwComponentType</b>	<b>ParameterSwComponentType</b>
<b>ApplicationSwComponentType</b> aggregates a <b>SwcInternalBehavior</b>	<b>ParameterSwComponentType</b> doesnot aggregate a <b>SwcInternalBehavior</b>
<b>ApplicationSwComponentType</b> recieves the parameters and characteristic values accessible via required Ports. The <b>RPortPrototypes</b> shall be typed <b>ParameterInterface</b> .	The <b>ParameterSwComponentType</b> defines parameters and characteristic values accessible via provided Ports. The <b>P-PortPrototypes</b> shall be typed <b>ParameterInterface</b> .
<b>ApplicationSwComponentType</b> may define parameters for own usage (either per instance or shared).	<b>ParameterSwComponentType</b> does not define parameters for own usage.

### 7.2.1.5 Usage of SW-CALIBRATION-ACCESS

The **swCalibrationAccess** determines the access rights to a data object w.r.t. measurement and calibration. **swCalibrationAccess** can take one of the following values

- READ-WRITE
- READ-ONLY
- NOT-ACCESSIBLE

Table 30 **swCalibrationAccess** Values

<b>Literal</b>	<b>Description</b>
<b>notAccessible</b>	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file
<b>readOnly</b>	The element will only appear as read-only in an ASAP file.
<b>readWrite</b>	The element will appear in ASAP file with both read and write access.

#### 7.2.1.5.1 Rules

**Rule Data\_040:** Instantiating a **ParameterDataPrototype** with **SwCalibrationAccess** and **SwImplPolicy**

##### Instruction

**Scope:** AUTOSAR 4.x

A calibration parameter is instantiated with a **ParameterDataPrototype** class that aggregates a **SwDataDefProps** with properties

- ***swCalibrationAccess** = **readWrite** and **swImplPolicy** = **standard** when the **ParameterDataPrototype** shall be visible in the A2L file and calibratable (changeable).*
- ***swCalibrationAccess** = **readOnly** and **swImplPolicy** = **standard** when the **ParameterDataPrototype** shall be visible in the A2L file and not calibratable (changeable).*
- ***swCalibrationAccess** = **notAccessible** and **swImplPolicy** = **standard** when the **ParameterDataPrototype** shall not be written to A2L file at all.*

**Hint** The DataSpec SubTWG have to define, where and with which attribute the SwCalibrationAccess should be defined

## Rule Data\_143: Calibration Access for Measurements

### Instruction

Measurement object shall only have READ-ONLY as Calibration Access

### 7.2.1.6 Usage of SW-IMPL-POLICY

SwImplPolicy defines how a data element needs to be processed at the receiver's end. This attribute can take the following values.

Table 31 SwImplPolicyValues

Literal	Description
const	forced implementation such that running software with in ECU shall not modify it. Applicable for parameters (not for those in NvRam) as well as argument dataprototypes.
fixed	This data element is fixed.
measurementPoint	The data element is created for measurement purposes only. The data element is never read directly within ECU software.
queued	The data elements are stored in a queue and all the data elements are processed in the "first-in-first-out" order. Queuing is intended to be implemented by RTE_gen. This value is not applicable for parameters.
standard	This is applicable for all data elements.

#### 7.2.1.6.1 Rules

### 7.2.2 VariableDataPrototype(Measurement Points)

Variable Data Prototype shall be used in one of the following contexts (Referenced from *Document "Software Component Template" [TPS\_SWCT]*).

#### Local Variable

Which is used as a whole (e.g. InterRunnableVariable, inputValue for curve).

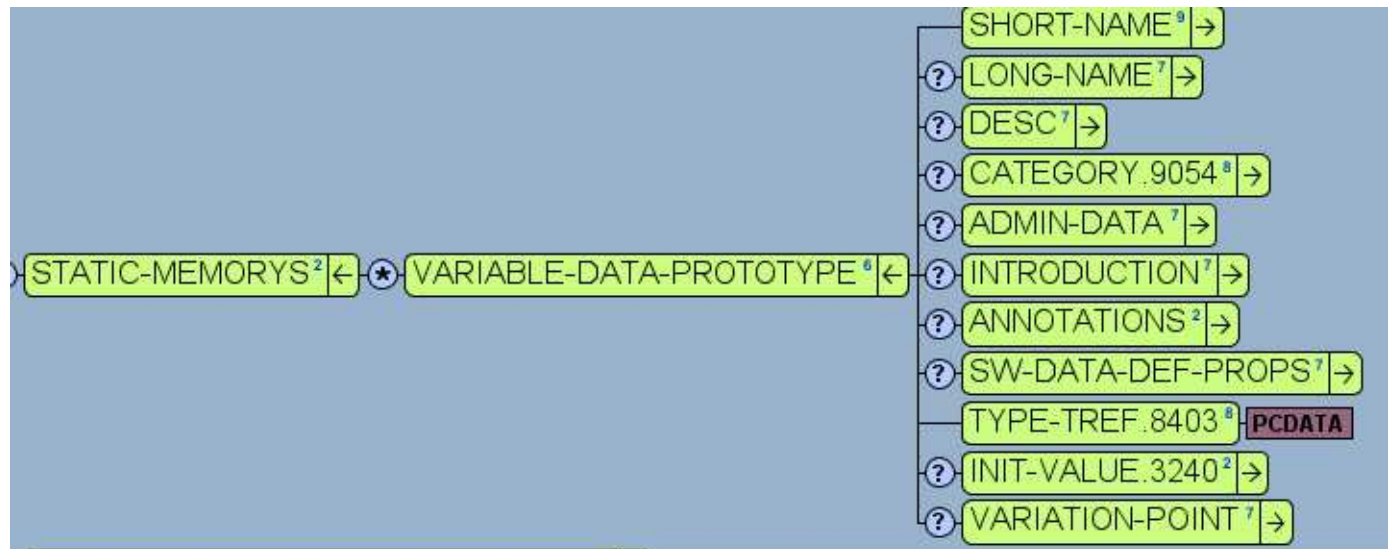
#### Autosar Variable

1. As a Variable provided via Port which is used as whole (e.g. dataAccesspoints)
2. As an element inside of a composite local variable typed by ApplicationDataType (e.g. inputValue for a curve)
3. As an element inside of a composite variable provided via Port and typed by ApplicationDataType (e.g. inputValue for a curve)

#### AUTOSAR variable in implementation data type

1. As an element inside of a composite local variable typed by ImplementationDatatype (e.g. nvramData mapping)
2. As an element inside of a composite variable provided via Port and typed by ImplementationDatatype (e.g. inputValue for a curve)

Figure 42 Schematic Representation of Variable Data Prototype



### 7.2.2.1 VariableDataPrototype of Category Value

#### Example:

```

<VARIABLE-DATA-PROTOTYPE>
  <SHORT-NAME>VariableDataPrototype_Value</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    <!-- Refer to an ApplicationPrimitiveDataType -->
  </TYPE-TREF>
</VARIABLE-DATA-PROTOTYPE>

```

#### 7.2.2.1.1 Rules

tbd

### 7.2.2.2 VariableDataPrototype of Category Boolean

A Variable Data Prototype of Category Boolean can contain one boolean state.

#### Example:

```

<VARIABLE-DATA-PROTOTYPE>
  <SHORT-NAME>VariableDataPrototype_Boolean</SHORT-NAME>
  <CATEGORY>BOOLEAN</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>

```

```

05  </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
        <!-- Refer to an ApplicationDataType of Category Boolean -->
    </TYPE-TREF>
</VARIABLE-DATA-PROTOTYPE>

```

### 7.2.2.2.1 Rules

tbd

### 7.2.2.3 VariableDataPrototype of Category Array

#### Example:

```

20  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME><!--Short Name for VDP --></SHORT-NAME>
    <CATEGORY>ARRAY</CATEGORY>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
                <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
                <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
        <!-- Refer to an ApplicationArrayDataType -->
    </TYPE-TREF>
</VARIABLE-DATA-PROTOTYPE>

```

### 7.2.2.3.1 Rules

tbd

### 7.2.2.4 VariableDataPrototype of Category Structure

#### Example:

```

45  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>VariableDataPrototype_Struct</SHORT-NAME>
    <CATEGORY>STRUCTURE</CATEGORY>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <ANNOTATIONS>
                    <ANNOTATION>
                        <LABEL>
                            <L-4 L="FOR-ALL">This is the annotation for VariableDataPrototype_Struct</L-4>
                        </LABEL>
                    </ANNOTATION>
                </ANNOTATIONS>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
                <DISPLAY-FORMAT>%5.0f</DISPLAY-FORMAT>
                <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
        <!-- Refer to an ApplicationRecordDataType -->
    </TYPE-TREF>
</VARIABLE-DATA-PROTOTYPE>

```



### 7.2.2.4.1 Rules

tbd

### 7.2.2.5 Compatibility of Variable Data Prototypes and Parameter Data Protoptypes

To refer to a `VariableDataPrototype` or `ParameterDataPrototye` it must be compatible. To ensure compatibility certain properties must be fulfilled which is described in [\[constr\\_1068\]](#) [\[TPS\\_SWCT\]](#)

### 7.2.2.6 Rules

TBD.

#### Rule Data\_120: Storage for Logical State of Boolean Category

##### Instruction

**Scope:** AR4.x

Depending on the CPU direct addressing of single bits may not be available. So a byte or a word can be used to store only one logical state. (Referenced from [Document "Software Component Template"](#) [\[TPS\\_SWCT\]](#) .

## 7.3 Usage of Attributes of `SwDataDefProps`

`SwDataDefProps` are an important schema element in AUTOSAR. They are applicable for different levels:

- ▶ `ApplicationDataTypes`
- ▶ `ImplementationDataTypes`
- ▶ `ParameterDataPrototypes` / `VariableDataPrototypes`
- ▶ `ParameterAccessss`

... and some more.

The properties inside `SwDataDefProps` are basically all optional. But AUTOSAR defines a precise rule where which property can or shall be set and where it can be overwritten. This is defined in [constr\\_1015](#) of [\[TPS\\_SWCT\]](#) in a huge table. Since many properties are not used in our environment we provide here a more restrictive but even much more simple table ["Usage of Attributes of SwDataDefProps"](#) . All properties which are not to be used are listed in table ["Not Used Attributes of SwDataDefProps"](#) .

### 7.3.1 Rules

#### Rule Data\_135: Usage of Attributes of `SwDataDefProps`

##### Instruction

Properties of `SwDataDefProps` shall be defined according to table ["Usage of Attributes of SwDataDefProps"](#) .

**Hint** This rule is preliminary since it is not finally reviewed.

Table 32 Usage of Attributes of *SwDataDefProps*

Attributes of <i>SwDataDefProps</i>	ApplicationDataType	ImplementationDataType	DataPrototype	ParameterAccess
annotations	D	A	A	na
baseType	na	D	I	na
compuMethod	D	[ na ]	I	na
dataConstr	D	[ na ]	[ I ]	na
displayFormat	D	na	I/R	na
implementationDataType	na	D	I	na
swAddrMethod	D	[ na ]	[ I ]	na
swCalibrationAccess	D	[ na ]	R	na
swCalprmAxisSet	D	na	I	na
swCalprmAxisSet.swCalprmAxis/ swAxisIndividual.swVariableRef	na	na	na	[ D ]
swCalprmAxisSet.swCalprmAxis/ swAxisIndividual.inputVariableType	D	na	na	na
swCalprmAxisSet/ SwAxisIndividual.unit	D	[ na ]	[ na ]	na
swImplPolicy	D	[ na ]	R	na
swRecordLayout	D	na	I	na
swRefreshTiming	D	[ na ]	R	na
swValueBlockSize	D	[ na ]	[ na ]	na

D Define this property here.

R Re-define this property if required. If not re-defined property's value will be taken from the left column(s).

A In case of multiplicity of >1 you may define an additional property content here.

I Value of the property will be inherited from the left column(s).

na Property shall not be defined here

Some cells define a more restrictive usage compared to AUTOSAR. This is marked as [ orange ].

## Rule Data\_136: Not Used Attributes of *SwDataDefProps*

### Instruction

Properties of *SwDataDefProps* listed in table "*Not Used Attributes of SwDataDefProps*" shall not be used.

**Hint** This rule is preliminary since it is not finally reviewed.

Table 33 Not Used Attributes of *SwDataDefProps*

Attributes of <i>SwDataDefProps</i>	ApplicationDataType	ImplementationDataType	DataPrototype	ParameterAccess
additionalNativeTypeQualifier	[ na ]	[ na ]	[ na ]	[ na ]
invalidValue	[ na ]	[ na ]	[ na ]	[ na ]
mcFunction	[ na ]	[ na ]	[ na ]	[ na ]
swAlignment	[ na ]	[ na ]	[ na ]	[ na ]
swBitRepresentation	[ na ]	[ na ]	[ na ]	[ na ]
swCalprmAxisSet.swCalprmAxis/ swAxisGrouped.swCalprmRef	[ na ]	[ na ]	[ na ]	[ na ]
swCalprmAxisSet.swCalprmAxis/ swAxisGrouped.sharedAxisType	[ na ]	[ na ]	[ na ]	[ na ]
swCalprmAxisSet.swCalprmAxis.baseType	[ na ]	[ na ]	[ na ]	[ na ]
swComparisonVariable	[ na ]	[ na ]	[ na ]	[ na ]
swDataDependency	[ na ]	[ na ]	[ na ]	[ na ]
swHostVariable	[ na ]	[ na ]	[ na ]	[ na ]
swIntendedResolution	[ na ]	[ na ]	[ na ]	[ na ]
swInterPolationMethod	[ na ]	[ na ]	[ na ]	[ na ]
swIsVirtual	[ na ]	[ na ]	[ na ]	[ na ]
swPointerTargetProps	[ na ]	[ na ]	[ na ]	[ na ]
swTextProps	[ na ]	[ na ]	[ na ]	[ na ]
valueAxisDataType	[ na ]	[ na ]	[ na ]	[ na ]
unit	[ na ]	[ na ]	[ na ]	[ na ]

## Rule Data\_137: Usage of *InstantiationDataDefProps*

### Instruction

*InstantiationDataDefProps* shall not be used.

*InstantiationDataDefProps* is a general class allowing to apply additional *SwDataDefProps* to particular instantiations of a *DataPrototype*. Since no understanding and no tool support is available it will not be used.

**Hint** This rule is preliminary since it is not finally reviewed.

## 7.4 Initial Values

### 7.4.1 Description

Autosar provides the feature to specify an initial value for a *VariableDataPrototype*/*ParameterDataPrototype* in case the value has not been assigned in a controlled manner. However, the definition of an initial value in many cases depends on a context in which the value is accessed.

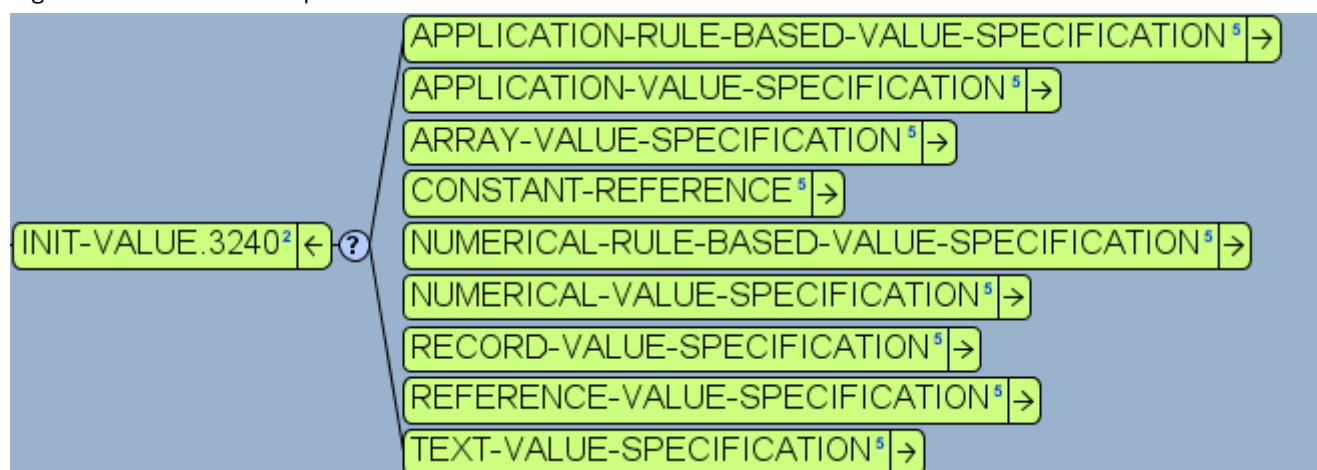
### 7.4.1.1 Initial Values for Calibration Parameter Overview

It is possible to assign instance specific initial values for calibration parameter. This shall override any initial values predefined by a ParameterDataPrototype. These initial values are specified in CalibrationParameterValueSet and CalibrationParameterValue. Following sub chapters describes the existing contexts.

An initial value can be one of the following:

- ▶ Application Value Specification
- ▶ Array Value Specification
- ▶ Constant Reference
- ▶ Numerical Value Specification
- ▶ Record Value Specification
- ▶ Reference Value Specification
- ▶ Text Value Specification

Figure 43 Schematic Representation of Init Value



### 7.4.1.2 Assigning Initial Values for Calibration Parameter

Initial values can be assigned to Calibration Parameter in the following ways.

1. Initial Value as content of the ParameterDataPrototype. All instances of this CalPrm will get the same initial values.
2. All the initial values are assigned in a CalibrationParameterValueSet (with references to FlatMap).

In this case each instance of a ParameterDataPrototype will get its own value. The CalibrationParameterValueSet is a separate ARElement, not aggregated to the SwComponentType and is typically contained in a separate file. It always contains physical values.

**Hint** The Module/SWC developer will only assign the initial value to the individual calibration parameter, since during this time the FlatMaps will not be created, and this will happen only during the integration phase.

#### 7.4.1.2.1 Initial Value assigned to individual Calibration Parameter

Example for a calibration parameter with a defined initial value is described below.

```

<PARAMETER-DATA-PROTOTYPE>
  <SHORT-NAME>ParameterDataPrototype_IF_1</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ApplDataTypes/N1
  </TYPE-TREF>
  <INIT-VALUE>
    <NUMERICAL-VALUE-SPECIFICATION>
      <VALUE>1</VALUE>
    </NUMERICAL-VALUE-SPECIFICATION>
  </INIT-VALUE>
</PARAMETER-DATA-PROTOTYPE>

```

## 7.4.2 Rules

tbd

## 7.5 Elements referred by Data Def Properties

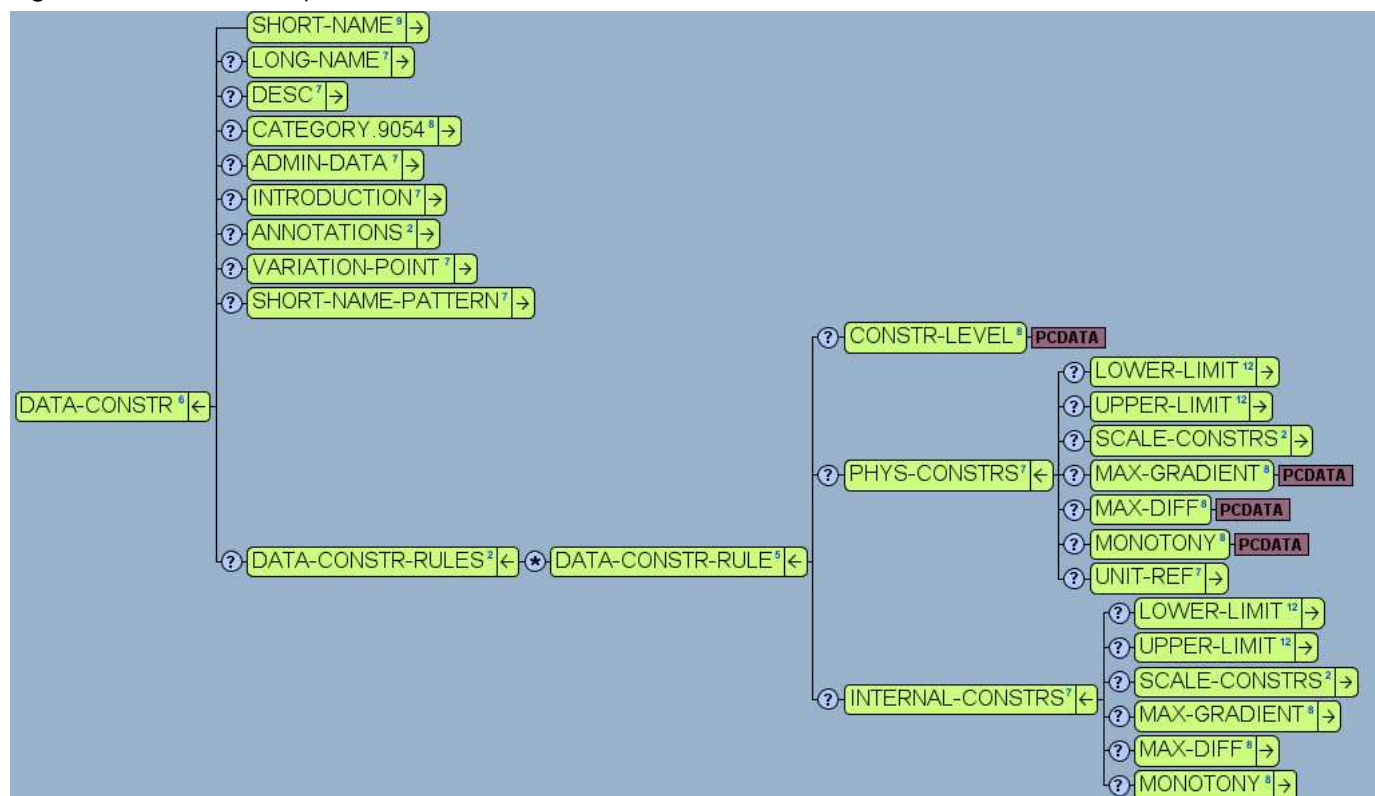
### 7.5.1 Data Constraints

#### What is a Data Constraint

Data Constraints are used to implement limits on element values, these limits are necessary to give the application engineer a senseful range of possible values. In general, the meta-class DataConstr can be referenced (via SwDataDef--Props.dataConstr to define various constraints for the possible values of a data type.

## Types of Data Constraints

Figure 44 Schematic Representation of Data Constraints



**1. Internal Constraint** This class represents the ability to express internal constraints.

### 2. Physical Constraint

This class represents the ability to express physical constraints. Therefore it has (in opposite to InternalConstrs) a reference to a Unit.

DataConstr is an ARElement which can be reused by several data type specifications. Especially an implementation and an *ApplicationDataType* which are mapped to each other, can refer to the same constraints or they can define their own constraints.

To avoid conflicts, in both cases PhysConstrs shall be interpreted by tools only with respect to *ApplicationDataTypes* while InternalConstrs shall be interpreted only with respect to implementation data types. If either a physical or internal constraint is missing an existing CompuMethod can be used to calculate the missing information. (Referenced from *Document "Software Component Template" [TPS\_SWCT]* ).

## Limits on Data Constraints

Limit specifies a boundary of the interval of valid values for a given context (i.e. a data type). Please note that the boundary might or might not be part of the interval itself, i.e. the interval might be open, closed or infinite. (Referenced from *Document "Software Component Template" [TPS\_SWCT]* ).

- ▶ **CLOSED** – value is part of the limit
- ▶ **OPEN** – value is not part of the limit
- ▶ **INF** – value is an infinity. This is achieved by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval.

If the category of Interval-Type is missing then CLOSED type can be taken as default value.

## Structure of Data Constraint

### Example:

```
<AR-PACKAGE>
  <SHORT-NAME>DataConstrs</SHORT-NAME>
  <ELEMENTS>
    <DATA-CONSTR>
      <SHORT-NAME>FlxrXcvr_DeviceSelection_C_CNS</SHORT-NAME>
      <DATA-CONSTR-RULES>
        <DATA-CONSTR-RULE>
          <PHYS-CONSTRS>
            <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
            <UPPER-LIMIT INTERVAL-TYPE="INF"></UPPER-LIMIT>
            <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/n</UNIT-REF>
          </PHYS-CONSTRS>
        </DATA-CONSTR-RULE>
      </DATA-CONSTR-RULES>
    </DATA-CONSTR>
  </ELEMENTS>
</AR-PACKAGE>
```

### 7.5.1.1 Rules

tbd

## 7.5.2 Computation Methods

An important part of semantics is the specification of a so-called computation method which specifies the conversion between the physical and the internal representation of data. This usually makes sense only for primitive data types.

CompuMethods are used for the conversion of internal values into their physical representation and vice versa. The direction of the conversion depends on the origin of the value to be converted:

1. If the value is provided by the ECU then the conversion direction is from internal to physical.
2. If a physical value is provided by the tester it is converted to internal values before being sent to the ECU.

CompuMethods can also be used to assign symbolic names to internal values (like an enumeration in C) or to ranges of internal values or to single bits (like a bitfield in C).

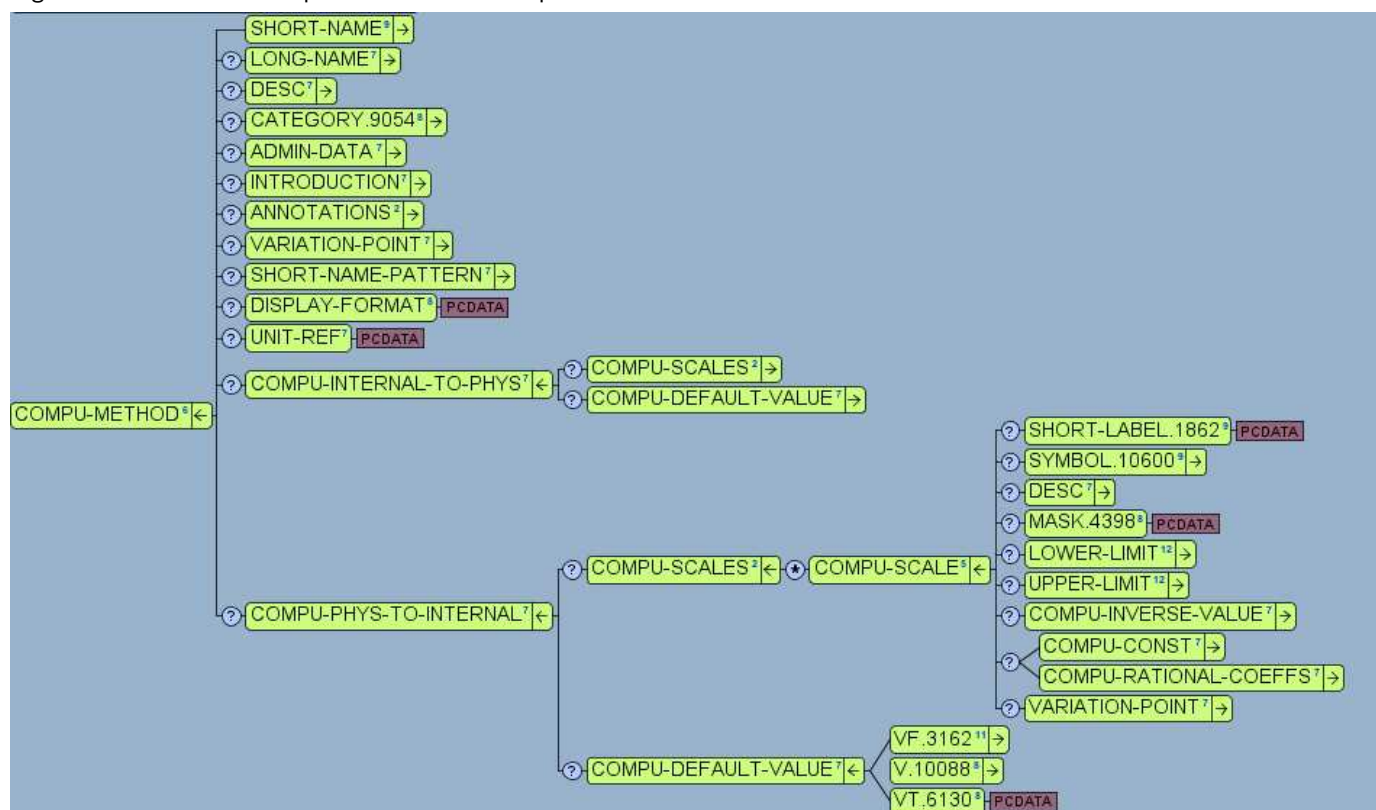
The following categories of CompuMethods are allowed

1. IDENTICAL
2. LINEAR
3. RAT\_FUNC
4. TEXTTABLE
5. TAB\_NOINTP
6. BITFIELD\_TEXTTABLE
7. SCALE\_LINEAR
8. SCALE\_LINEAR\_AND\_TEXTTABLE
9. SCALE\_RAT\_FUNC
10. SCALE\_RATIONAL\_AND\_TEXTTABLE

**Hint** The categories IDENTICAL, LINEAR, RAT\_FUNC, TEXTTABLE are alone supported in Ar\_A2L\_Gen.

SCALE\_LINEAR\_AND\_TEXTTABLE and SCALE\_RATIONAL\_AND\_TEXTTABLE are supported from Autosar 4.1.1 onwards

Figure 45 Schematic Representation of CompuMethod



### 7.5.2.1 IDENTICAL

This is the simplest type of a **CompuMethod**. This **CompuMethod** just hands over the internal value with an optional unit.

#### CompuMethod\_Idential

##### Example:

```

<COMPU-METHOD>
  <SHORT-NAME>Identical</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">The internal value and the physical one are identical. No physical unit available</L-4>
  </LONG-NAME>
  <CATEGORY>IDENTICAL</CATEGORY>
</COMPU-METHOD>
  
```

### 7.5.2.2 LINEAR

A linear conversion can be performed in two steps: The internal value is multiplied with a factor; after that, an offset is added to the result of the multiplication. Only the base elements are allowed and unit, physConstr and internalConstr are optional. A linear CompuMethod supports exactly one CompuScale with 2 Compu Numerator and 1 Compu Denominator.

The formula for Linear conversion is as follows

$$\text{Internal} = (v_{[0]} * x^0 + v_{[1]} * x^1) / (w_{[0]} * x^0)$$



**Hint** The conversion formula can be found at [Document "Software Component Template" \[TPS\\_SWCT\]](#)

## CompuMethod\_Linear

The following examples illustrates how a linear conversion is specified using CompuMethod.

$$F_{[ms]} = 5.0f_{[ms]} + 2.0f_{[ms]} * X$$

### Example:

```
<COMPU-METHOD>
  <SHORT-NAME>CompuName_Linear</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/ms</UNIT-REF>
  <COMPU-PHYS-TO-INTERNAL>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>5.0</V>
            <V>2.0</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1.0</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>
```

## 7.5.2.3 RAT\_FUNC

The rational function type is similar to the linear type without the restrictions for the compuNumerators and compuDenominators. It can have as many v elements as needed for the rational function. The sequence of the values v carries the information for the exponents, that means the first v is the coefficient for x0, the second v is the coefficient for x1, etc.

The formula for Rat\_Fun conversion is as follows

$$\text{Internal} = (v_{[0]} * x^0 + v_{[1]} * x^1 + \dots) / (w_{[0]} * x^0 + w_{[1]} * x^1) = \dots$$

**Hint** The conversion formula can be found at [Document "Software Component Template" \[TPS\\_SWCT\]](#) p. 403

The following examples illustrates how a Rat\_Fun conversion is specified using CompuMethod.

$$I = 1000 / 60 + (2_{[k-1]} * X_{[k]})$$

## CompuMethod\_RatFunc

### Example:

```
<COMPU-METHOD>
  <SHORT-NAME>CompuName_RatFunc</SHORT-NAME>
  <CATEGORY>RAT_FUNC</CATEGORY>
  <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/Kelvin</UNIT-REF>
  <COMPU-PHYS-TO-INTERNAL>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-29</LOWER-LIMIT>
```

```

<UPPER-LIMIT INTERVAL-TYPE="OPEN">INF</UPPER-LIMIT>
<COMPU-RATIONAL-COEFFS>
  <COMPU-NUMERATOR>
    <V>1000</V>
  </COMPU-NUMERATOR>
  <COMPU-DENOMINATOR>
    <V>60</V>
    <V>2</V>
  </COMPU-DENOMINATOR>
</COMPU-RATIONAL-COEFFS>
</COMPU-SCALE>
</COMPU-SCALES>
</COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>

```

### 7.5.2.4 TEXTTABLE

The type TEXTTABLE is used for transformations of the internal value into textual elements.

**Caution** TEXTTABLE CompuMethod do not support physConstr.

#### CompuMethod\_TextTable

**Example:**

```

<COMPU-METHOD>
  <SHORT-NAME>CompuName_TextTable</SHORT-NAME>
  <CATEGORY>TEXTTABLE</CATEGORY>
  <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/Kelvin</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <DESC><L-2 L="EN">0 = Opend (Opened)</L-2></DESC>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0.0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0.0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>CANIF_CS_OPENED</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <DESC><L-2 L="EN">1 = Inter (Intermediate)</L-2></DESC>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1.0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1.0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>CANIF_CS_INTER</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <DESC><L-2 L="EN">2 = NearClsd (Near Closed)</L-2></DESC>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">2.0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2.0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>CANIF_CS_NRCLSD</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <DESC><L-2 L="EN">3 = HardClsd (Hard Closed)</L-2></DESC>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">3.0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">3.0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>CANIF_CS_HDCLSD</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>

```

```
</COMPU-SCALES>  
</COMPU-INTERNAL-TO-PHYS>  
</COMPU-METHOD>
```

### 7.5.2.5 Rules

#### Rule Data\_090: Usage of IDENTICAL, LINEAR and RAT\_FUNC in CompuMethod

##### Instruction

**Scope:** AUTOSAR 4.x

Developer shall use IDENTICAL CompuMethod for identical mapping and LINEAR CompuMethod shall be used where ever IDENTICAL CompuMethod is not possible to use. RAT\_FUNC CompuMethod shall be used only for Rational Functions.

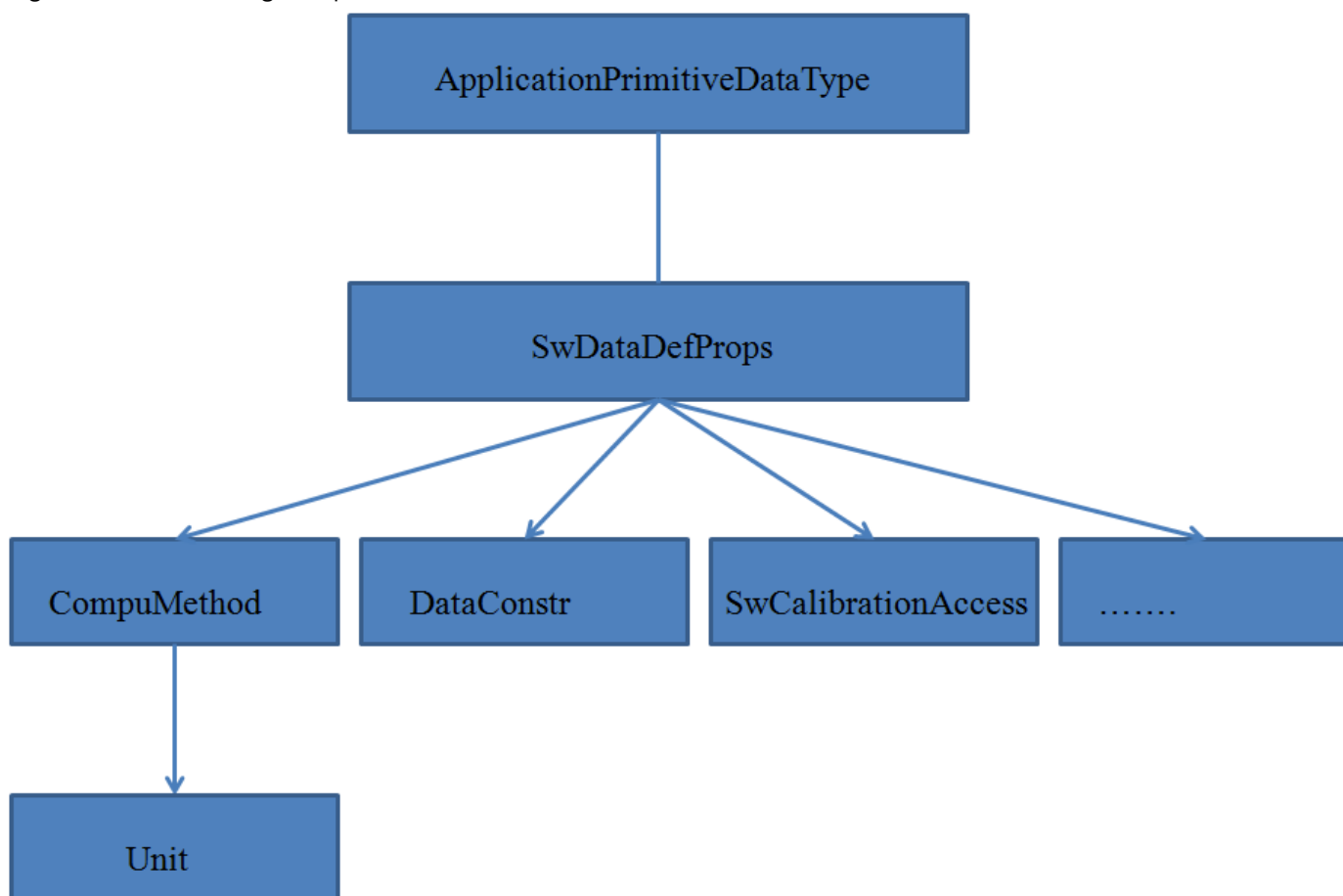
#### Rule Data\_091: Referencing a CompuMethod from an ApplicationDataType

##### Instruction

**Scope:** AUTOSAR 4.x

CompuMethod can be referenced by an ApplicationDataType through SwDataDefProps.

Figure 46 Referencing CompuMethod



#### Rule Data\_092: Floating numbers to be used for Computation Formulas

##### Instruction

**Scope:** AUTOSAR 4.x

The parameters of computation formulas have to be entered as float with a suffix "f".  
e.g. "5.0f" instead of "5"

**Hint** Floating point numbers shall be suffixed with "f" type character as the value is to be considered as double by default by compiler.

**Hint** Developers shall take care while using computation formulas such that divide by zero is avoided.

## Rule Data\_110: Concerns to Computation Method

### Instruction

**Scope:** AUTOSAR 4.x

1. Every CompuMethod shall have a reference to UNIT.
2. If the *ApplicationDataType* has no physical unit, e.g. a Counter, then it shall reference to a unit by name "NoUnit".

## Rule Data\_113: Removed

## Rule Data\_115: Conversion Direction for Category Linear

### Instruction

**Scope:** AUTOSAR 4.X

"COMPU-PHYS-TO-INTERNAL" is the allowed direction of conversion for CompuMethod of category LINEAR.

Inverse function can be derived quite easily from the defined function. The limits for the reverse direction can be gained by applying the forward function to the forward limits.

**Hint** This conversion rule is due to our tooling constraints. It might be removed in future versions.

## Rule Data\_116: Removed

## Rule Data\_117: Conversion Direction for Category Rat\_Func

### Instruction

**Scope:** AUTOSAR 4.X

"COMPU-PHYS-TO-INTERNAL" is the allowed direction of conversion for CompuMethod of category RAT\_FUNC.

It is usually not possible to compute the inverse function automatically. The inversion yields ambiguous results if the function is not monotonic. To deal with such possible ambiguities in a direct way an inverse value can be provided explicitly for the function or for each of its parts respectively.

**Hint** This conversion rule is due to our tooling constraints. It might be removed in future versions.

## Rule Data\_118: Conversion Direction for Category TextTable

### Instruction

**Scope:**AUTOSAR 4.X

"COMPU-INTERNAL-TO-PHYS" is the allowed direction of conversion for CompuMethod of category TEXTTABLE.

## Rule Data\_111: Removed

## Rule Data\_124: Definition of CompuMethod

### Instruction

**Scope:**AUTOSAR 4.X

Arithmetic CompuMethods are mainly defined centrally. These includes following Categories

- ▶ LINEAR
- ▶ RAT\_FUNC
- ▶ IDENTICAL

Verbal CompuMethods shall be defined centrally or decentral. These includes following Categories

- ▶ TEXTTABLE

This rule satisfies CEL\_051 and CEL\_052 which can be referenced in [Document "Central Elements" \[A\\_CEL\]](#)

## Rule Data\_132: Allowed Categories for CompuMethod

### Instruction

**Scope:**AUTOSAR 4.X

All the categories defined by Autosar are not supported in BOSCH yet. Developer shall use the categories IDENTICAL, LINEAR, RAT\_FUNC for their developmental activities.

## Rule Data\_133: Removed

## Rule Data\_145: Values for Category TextTable

### Instruction

**Scope:**BOSCH

The text values used in CompuMethod of category TextTable shall be unique.

### Example:

```
<COMPU-CONST>
  <VT><!-- Provide a Unique Value --></VT>
</COMPU-CONST>
```

## 7.5.3 Units and UnitGroup

### 7.5.3.1 Units

#### Physical Units

An important part of the semantics associated with a data type is its physical units. Units are used to specify physical units (e.g., ms) as an additional attribute of physical Application Data Type. That is necessary for a correct interpretation of the physical value for input and output processes.

The conversion of values into other units like km/h into miles/h is also possible. Therefore the unit involves information about its physical dimensions. The substructure of physical dimensions defines all used quantities in the SI-Base-System (e.g. velocity as length/time corresponds to m/s).

#### Example:

```
<AR-PACKAGE>
<SHORT-NAME>RB</SHORT-NAME>
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>Common</SHORT-NAME>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>CentralElements</SHORT-NAME>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>Units</SHORT-NAME>
    <ELEMENTS>
      <UNIT>
        <SHORT-NAME>Kelvin</SHORT-NAME>
        <DESC>
          <L-2 L="FOR-ALL">Description for Kelvin</L-2>
        </DESC>
        <DISPLAY-NAME>Kelvin</DISPLAY-NAME>
        <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
        <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
      </UNIT>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
```

**Hint** If the physical dimension of two units is identical, then a conversion between them is possible. The conversion between units is related to the definition of the physical dimension.

#### Example:

```
<PHYSICAL-DIMENSION>
  <SHORT-NAME>Speed</SHORT-NAME>
  <LENGTH-EXP>1</LENGTH-EXP>
  <TIME-EXP>1</TIME-EXP>
</PHYSICAL-DIMENSION>
```

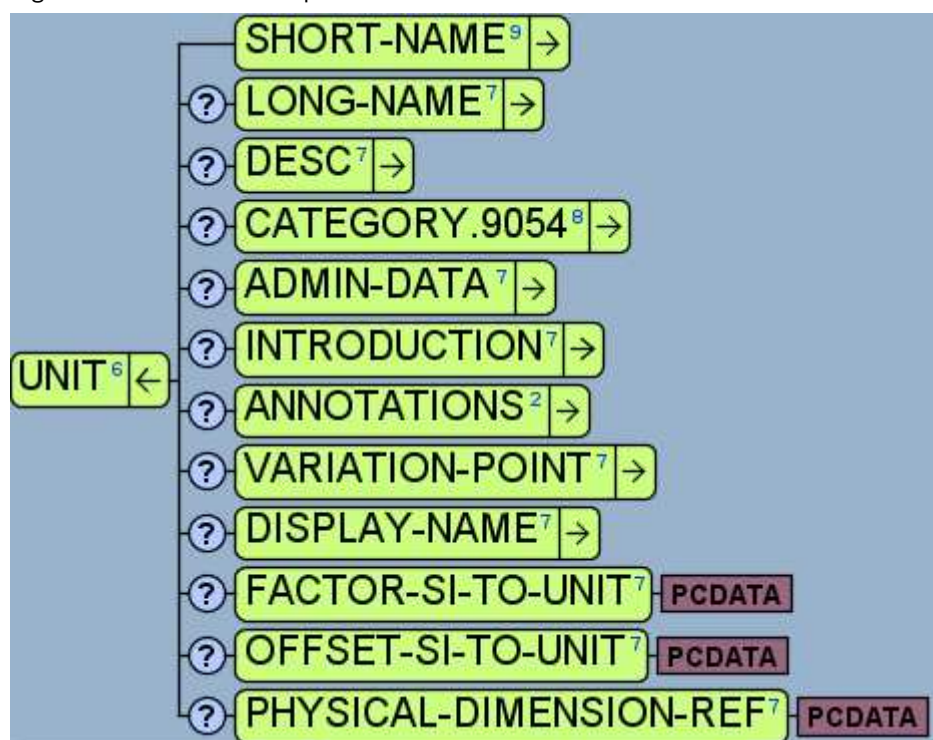
### 7.5.3.2 UnitGroup

This metaclass represents the ability to specify a logical grouping of units. The category denotes the unit system that the referenced units are associated to. In this way, e.g. country-specific unit systems (CATEGORY="COUNTRY") can be defined as well as specific unit systems for certain application domains.

For example, a group of equivalent units can be defined, which are used in different countries, by setting CATEGORY="EQUIV-UNITS". KmPerHour and MilesPerHour could such be combined to one group named "vehicle\_speed". The unit MeterPerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named "speed".

**Hint** The UnitGroup does not ensure the physical compliance of the units. This is maintained by the physical dimension.

Figure 47 Schematic Representation of Unit



Example:

```
<AR-PACKAGE>
  <SHORT-NAME>RB</SHORT-NAME>
</AR-PACKAGE>
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>Common</SHORT-NAME>
  </AR-PACKAGE>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>CentralElements</SHORT-NAME>
    </AR-PACKAGE>
    <AR-PACKAGES>
      <AR-PACKAGE>
        <SHORT-NAME>Units</SHORT-NAME>
        <ELEMENTS>
          <UNIT>
            <SHORT-NAME>Kelvin</SHORT-NAME>
            <DESC>
              <L-2 L="FOR-ALL">Description for Kelvin</L-2>
            </DESC>
            <DISPLAY-NAME>Kelvin</DISPLAY-NAME>
            <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
            <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
          </UNIT>
        </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
```

```

05      <UNIT>
        <SHORT-NAME>ms</SHORT-NAME>
        <DESC>
          <L-2 L="FOR-ALL">Description for ms</L-2>
        </DESC>
10      <DISPLAY-NAME>ms</DISPLAY-NAME>
        <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
        <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
      </UNIT>
      <UNIT>
        <SHORT-NAME>rpm</SHORT-NAME>
        <DESC>
          <L-2 L="FOR-ALL">Description for rpm</L-2>
        </DESC>
        <DISPLAY-NAME>rpm</DISPLAY-NAME>
        <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
        <OFFSET-SI-TO-UNIT>0.0</OFFSET-SI-TO-UNIT>
      </UNIT>
    </ELEMENTS>
  </AR-PACKAGE>
  <AR-PACKAGE>
    <SHORT-NAME>UnitGroup</SHORT-NAME>
    <ELEMENTS>
      <UNIT-GROUP>
        <SHORT-NAME>UnitGroup1</SHORT-NAME>
        <UNIT-REFS>
          <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/Kelvin</UNIT-REF>
          <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/ms</UNIT-REF>
          <UNIT-REF DEST="UNIT">/RB/Common/CentralElements/Units/rpm</UNIT-REF>
        </UNIT-REFS>
      </UNIT-GROUP>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGES>
</AR-PACKAGES>
</AR-PACKAGES>
</AR-PACKAGES>
</AR-PACKAGES>
</AR-PACKAGES>

```

### 7.5.3.3 Rules

#### Rule Data\_112: Every Unit shall contain FactorSiToUnit and OffsetSiToUnit

##### Instruction

**Scope:** AUTOSAR 4.x

**1.** UNIT shall contain FactorSiToUnit.

**2.** UNIT shall contain OffsetSiToUnit.

This rule satisfies CEL\_028 *Document "Central Elements" [A\_CEL]*

##### Hint

► factorSiToUnit – This is the factor for the conversion from and to siUnits

► offsetSiToUnit – This is the offset for the conversion from and to siUnits

All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined.



For the calculation from SI-unit to the defined unit the factor (factorSiToUnit ) and the offset (offsetSiToUnit ) are applied.

$$\text{siUnit} = (\text{unit} - \text{offsetSiToUnit}) / \text{factorSiToUnit}$$

For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit ) and the negation of the offset (offsetSiToUnit ) are applied:

$$\text{unit} = \text{siUnit} * \text{factorSiToUnit} + \text{offsetSiToUnit}$$

## Rule Data\_121: Removed

## Rule Data\_125: Definition of Units and Unit Groups

### Instruction

**Scope:**AUTOSAR 4.X

Units and UnitGroups shall be defined centrally. Developers shall not make a reference to unit directly from their data element as they are referenced by CompuMethod.

This rule satisfies CEL\_001 and CEL\_031 which can be referenced in [Document "Central Elements" \[A\\_CEL\]](#)

**Hint** UnitGroups are currently not in use.

## Rule Data\_126: Removed

## 8 Rule Set: Basis Software Module Description (BSWMD)

The BSWMD is a formal notation for information belonging to a certain BSW module in addition to the code implementation of that module. A BSWMD file is as mandatory as C/H files of a BSW module because it contains some technical relevant features and use cases which can only be described with an ARXML based artifact.

The BSWMD is based on the Basis Software Module Description Template ( [\[TPS\\_BSWMDT\]](#) ) which is the standardized format represented in UML as part of the overall AUTOSAR meta-model and XML schema.

The explanation of BSWMD can be made based on different approaches. One approach is to describe the use cases of BSWMD, another approach is to describe the internal structure of a BSWMD file from a model point of view. Both approaches are not strictly separable because some use cases require different layers of a BSWMD model. Anyway it is substantial to know the internal structure of a BSWMD file containing the knowledge about the model view and the file partitioning. This will help to understand the description of the use cases and the location of the specific elements. Therefore both approaches will be covered by this rule set.

First a small overview of common aspects of BSWMD will be given in chapter [8.1 "Common Aspects of BSWMD" p. 218](#) . Here some rules regarding the design and style of a BSWMD ARXML file are given. In the next chapter [8.2 "BSWMD Model View" p. 222](#) there will be a look to the model view containing general rules for the single model layers of a BSWMD file. The last chapter [8.3 "BSWMD Use Cases" p. 237](#) explains the different use cases provided from a BSWMD file.

### 8.1 Common Aspects of BSWMD

This chapter gives an overview of common aspects of BSWMD files and specifies common rules for the handling of BSWMD ARXML files.

#### Rule BSWMD\_Common\_002: Headline of a BSWMD ARXML File

**Instruction** Each BSWMD ARXML file shall start with a headline containing the definition of the used AUTOSAR schema.

In detail there are two headlines which are mandatory for every BSWMD ARXML file. In the following the headlines are specified:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
```

Currently an AUTOSAR schema which is conform to AR4.0.3 is used. This schema is chosen because it contains the support of calibration and measurement data within BSWMD which was not supported by previous schemas. The schema information is given within the <AUTOSAR> tag. It is obvious that at the end of each ARXML file the end tag </AUTOSAR> shall be placed.

**Hint** The three lines from the definition above containing the schema information should be written as single line in a productive BSWMD file. The split to three lines is here only done for clearness.

#### Rule BSWMD\_Common\_001: BSWMD Standard File Header.

**Instruction** Every non-generated BSWMD ARXML file shall be headed by a standard file header. This file header shall be located directly behind the AUTOSAR schema definition.

Definition of the standard file header of a BSWMD ARXML file (the *italic* part):

```

<?xml version...>
<AUTOSAR xmlns:sxi=...>

  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <SDGS>
      <SDG GID="RBHead-BASD-eASEE-Keywords">
        <SD GID="Domain"></SD>
        <SD GID="Namespace"></SD>
        <SD GID="Class"></SD>
        <SD GID="Name"></SD>
        <SD GID="Variant"></SD>
        <SD GID="Revision"></SD>
        <SD GID="History"></SD>
      </SDG>
    </SDGS>
  </ADMIN-DATA>

  <AR-PACKAGES>
    <AR-PACKAGE>
      ...
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>

```

The file header is needed such that the SCM system can place versioning information into a checked-out ARXML file. With that information the traceability of files relating to a specific version is ensured.

### Rule BSWMD\_Common\_003: ArPackage Hierarchy Within BSWMD Files

**Instruction** BSWMD files shall be conform to the *ArPackage* structure. That means that

- ▶ BSW modules specified by AUTOSAR shall use /AUTOSAR\_<ModulePrefix>/ as top level hierarchy.
  - ▶ BSW modules not specified by AUTOSAR shall use /RB/RBA/<ModulePrefix>/ as top level hierarchy.
- The <ModulePrefix> represents the module prefix of the BSW module (but without a VendorId and a VendorApilnfix). In the BSWMD file the ArPackage structure shall be derived from the top level hierarchy.

Each BSWMD file has to be embedded in an *ArPackage* structure. The general topics to the *ArPackage* structure is specified in [Chapter 9 "Rule Set: AUTOSAR Package Structure" p. 290](#) . The relevant rules for BSWMD files are the rules [\[ARPac\\_41\] p. 300](#) and [\[ARPac\\_43\] p. 301](#) which are the base for this rule. Valid module prefixes of AUTOSAR based BSW modules are listed in [Chapter C "List of Basic Software Modules" p. 395](#) . BSW modules which are not specified by AUTOSAR use a module prefix starting with "rba\_".

Example for the *ArPackage* structure for the AUTOSAR based module Adc:

```

<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>AUTOSAR_Adc</SHORT-NAME>
  </AR-PACKAGE>
</AR-PACKAGES>

```

Example for the *ArPackage* structure for the non AUTOSAR based module rba\_Hugo:

```

05 <AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>RB</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>RBA</SHORT-NAME>
10        <AR-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>rba_Hugo</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
15                    <!-- Specifiy here the BSWMD specific ArPackages -->
                    <!-- like BswModuleDescription, BswImplementation -->
                    <!-- ApplicationDataTypes, BswModuleEntries, ... -->
                    ...
                </AR-PACKAGE>
            </AR-PACKAGES>
20        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGES>
25 </AR-PACKAGES>
</AR-PACKAGES>

```

## Rule BSWMD\_Common\_004: Comments Within BSWMD Files

**Instruction** Within a BSWMD file comments may be set using the comment marker <!-- ... -->.

Comments within BSWMD files are really only comments and have no character of documentation. Everything between the comment markers will be ignored from the tools processing ARXML files. Using comments is optional. Example:

```

35 <AR-PACKAGES>
    <AR-PACKAGE>
        <!-- This is the AR-Package for the BSW module Adc -->
        <SHORT-NAME>AUTOSAR_Adc</SHORT-NAME>
40        ...

```

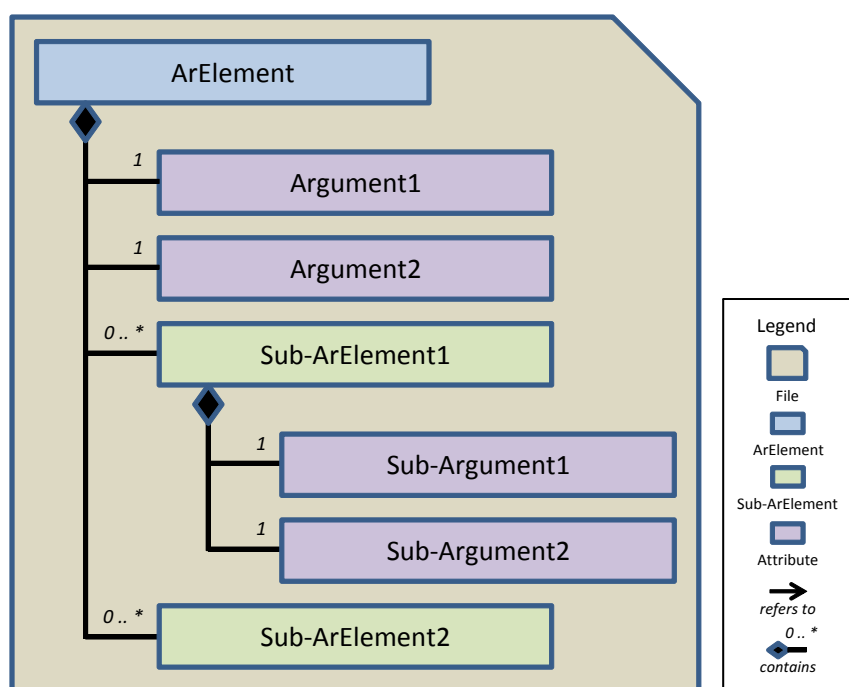
## Rule BSWMD\_Common\_005: Order of Tag Sequences Within BSWMD Files

**Instruction** The tag structure and tag sequence of BSWMD syntax are shown in figures and examples. Unless otherwise specified the illustrated structure and tag sequences shall be followed.

The tag sequence within BSWMD ARXML files usually follows a defined order specified by the ARXML schema. The tool chain expects only BSWMD files which are conform to the schema otherwise the tools could abort with errors. It would be too complex to define specific rules which regulate the order of tag sequences. Therefore only this common rule is specified to follow the tag structure shown in figures and examples. Possible exceptions will be mentioned explicitly if needed.

In [Figure 48](#) a common example is given and after that the figure is translated to ARXML syntax.

Figure 48 Example for a Tag Sequence within a BSWMD File



Derived tag structure in ARXML:

```

<ARELEMENT>
  <ARGUMENT1>...</ARGUMENT1>
  <ARGUMENT2>...</ARGUMENT2>
  <SUB-ARELEMENT1>
    <SUB-ARGUMENT1>...</SUB-ARGUMENT1>
    <SUB-ARGUMENT2>...</SUB-ARGUMENT2>
  </SUB-ARELEMENT1>
  <SUB-ARELEMENT2>
    ...
  </SUB-ARELEMENT2>
</ARELEMENT>

```

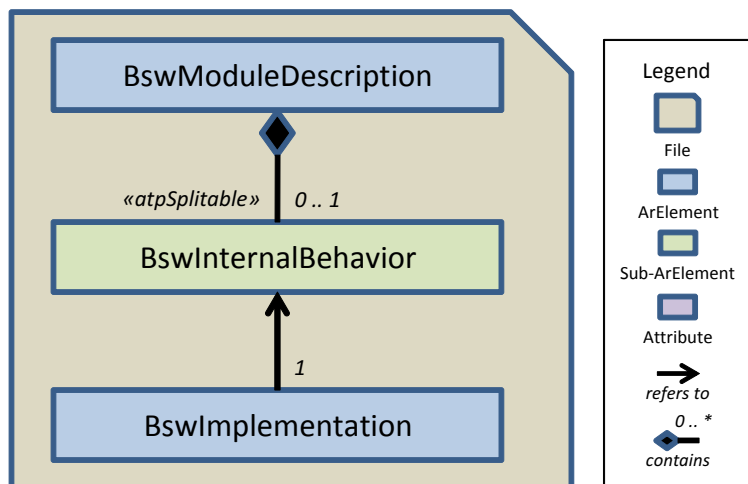
It is not possible to swap Argument1 and Argument2, also it is not possible to swap Sub-ArElement1 and Sub-Ar-Element2.

## 8.2 BSWMD Model View

### 8.2.1 Top Layers of BSWMD

The meta-model of the BSWMDT (Basis Software Module Description Template) consists of three abstraction layers similar to the SWCT (Software Component Template) and is shown in [Figure 49](#).

Figure 49 Three Layers of the BSW Module Description



The upper layer, the *BswModuleDescription*, contains the specification of all the provided and required interfaces including the dependencies to other BSW modules. This layer represents the properties and features of the BSW module to the outside of the module.

The middle layer, the *BswInternalBehavior*, contains a model of some basic activity inside the BSW module. This model layer defines the requirements of the BSW module for the configuration of the OS and the BSW Scheduler. It has a look at the internal needs of the BSW module and ensures the correct functionality of the BSW module such that the BSW module can operate.

The bottom layer, the *BswImplementation* contains information about the individual code of the BSW module. Here the kind of implementation (e.g. source code written in C language) and the characteristic of the BSW module (properties like software version and AUTOSAR version) are described and documented.

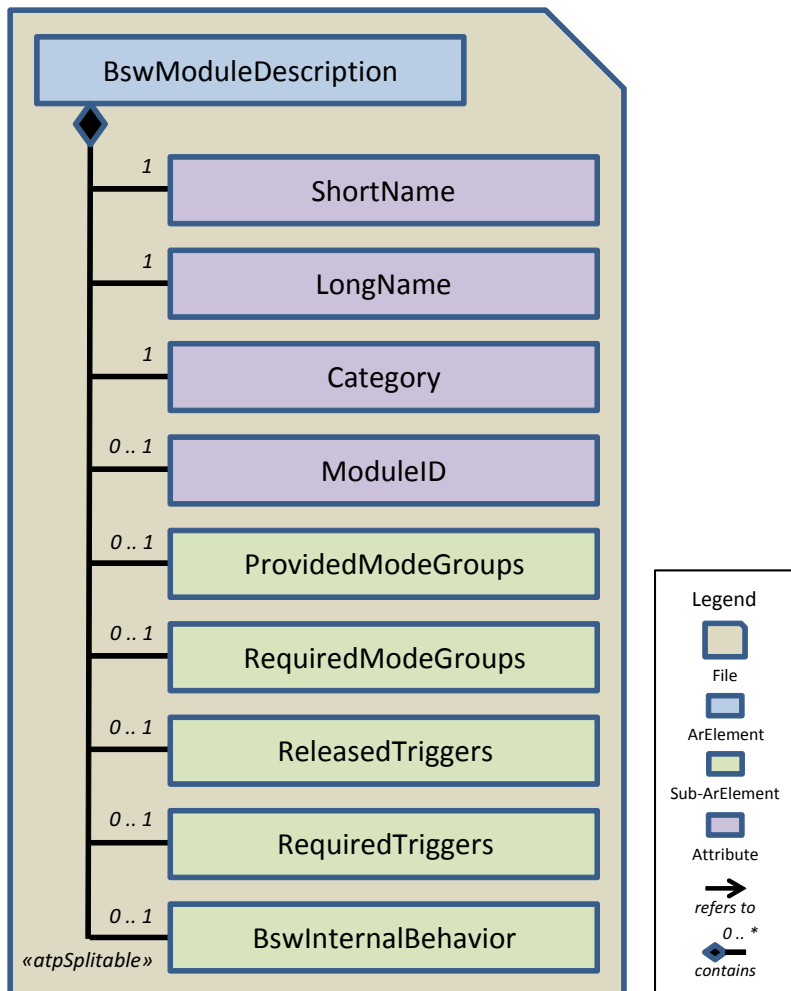
The multiplicity between the *BswModuleDescription* and the *BswInternalBehavior* is given by 0 .. 1 from point of the model view. This implies that the *BswInternalBehavior* is optional. But if parts of the *BswInternalBehavior* are needed (e.g. for the definition of measurement and calibration data, *BswEvents*, BSW module entities or BSW modes, see chapter [8.3 "BSWMD Use Cases"](#) for more details) the *BswInternalBehavior* is mandatory and has to be specified for a BSW module. AUTOSAR also allows to have several different variants of same instances of a *BswInternalBehavior* based on the same *BswModuleDescription*. This kind of instantiation will not be followed up, to have only one instance of a *BswInternalBehavior* will currently be the focus of the description of the BSWMD layers. Finally note that the term "behavior" has been chosen in analogy to a similar term in SWCT. It is restricted only to the scheduling behavior and does not describe the algorithmic behavior of the BSW module or cluster.

Between the *BswInternalBehavior* and the *BswImplementation* there is a 1:1 connection which is made by a reference. This implies that if a *BswInternalBehavior* exists also a *BswImplementation* has to be provided.

### 8.2.1.1 BswModuleDescription

The BswModuleDescription describes the external view of a BSW module. In [Figure 50](#) an overview of a *BswModuleDescription* is given. Only those elements are shown which are currently on focus and are described within the BSW Coding Guidelines. Updates will be done gradually.

Figure 50 Details of BswModuleDescription



The following rules define the handling of common attributes of the *BswModuleDescription*. Sub-ArElements are not described in detail. This is done in the description of the single use cases in chapter [8.3 "BSWMD Use Cases"](#). The following ARXML snippet shows an example for the AUTOSAR BSW module Adc.

```

<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleDescriptions</SHORT-NAME> <!-- Rule [BSWMD_ModuleDesc_001] -->
    <ELEMENTS>
      <BSW-MODULE-DESCRIPTION>
        <SHORT-NAME>Adc</SHORT-NAME> <!-- Rule [BSWMD_ModuleDesc_002] -->
        <LONG-NAME> <!-- Rule [BSWMD_ModuleDesc_003] -->
          <L-4 L="EN">ADC Driver</L-4>
        </LONG-NAME>
        <CATEGORY>BSW_MODULE</CATEGORY> <!-- Rule [BSWMD_ModuleDesc_004] -->
        <MODULE-ID>123</MODULE-ID> <!-- Rule [BSWMD_ModuleDesc_005] -->

        <!-- Use Case specific contents: -->
        <PROVIDED-MODE-GROUPS> <!-- Provided Modes -->
          ... <!-- Refer to Chapter 8.3.2.2.1 p. 254 -->
        </PROVIDED-MODE-GROUPS>
      </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>

```

```

05      <REQUIRED-MODE-GROUPS>                                <!-- Required Modes          -->
        ...                                                    <!-- Refer to Chapter 8.3.2.2.2 p. 256 -->
      </REQUIRED-MODE-GROUPS>
      <RELEASED-TRIGGERS>                                     <!-- Released Triggers       -->
        ...                                                    <!-- Refer to Chapter 8.3.2.3.1 p. 263 -->
      </RELEASED-TRIGGERS>
      <REQUIRED_TRIGGERS>                                     <!-- Required Triggers       -->
        ...                                                    <!-- Refer to Chapter 8.3.2.3.2 p. 265 -->
      </REQUIRED_TRIGGERS>

      <!-- Internal behavior -->
      <INTERNAL-BEHAVIOR>                                     <!-- BSW Internal Behavior    -->
        <BSW-INTERNAL-BEHAVIOR>                               <!-- Refer to Chapter 8.2.1.2 p. 226 -->
          ...
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIOR>
    </BSW-MODULE-DESCRIPTION>
    ...
  </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>

```

**Hint** Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined in rule [\[BSWMD\\_Common\\_005\] p. 220](#)). Otherwise the tool cannot interpret the BSWMD file correctly.

## Rule BSWMD\_ModuleDesc\_001: ShortName of ArPackage Child Element BswModuleDescription

**Instruction** The ShortName of the *ArPackage* for the *BswModuleDescription* shall be set to "BswModuleDescriptions".

This rule is derived from the rule [\[ARPac\\_44\] p. 301](#) .

```

...
<AR-PACKAGE>
  <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-DESCRIPTION>
    ...

```

## Rule BSWMD\_ModuleDesc\_002: ShortName of the BswModuleDescription

**Instruction** The ShortName of the *BswModuleDescription* shall be identical to the module prefix of the BSW module (but without a VendorId and a VendorApiInfix).

The module prefix shall be built conform to rule [\[CDGNaming\\_001\] p. 80](#) but without the case that the module prefix contains a VendorId or BOSCH specific name (VendorApiInfix). Possible existing VendorIds and VendorApiInfixes will be specified with special tags of the BswImplementation (see rule [\[BSWMD\\_Impl\\_003\] p. 229](#) ) and are not set in the short name representing the module prefix.

This rule shall also be applied in case of splitting the BswInternalBehavior from the BswModuleDescription which is described in rule [\[BSWMD\\_Split\\_002\] p. 233](#) . Here the BswModuleDescription is used as parent element to embed the BswInternalBehavior. In this case the short name of the BswModuleDescription has to be repeated and set to the same name which is used in the main BswModuleDescription.

## Rule BSWMD\_ModuleDesc\_003: LongName of the BswModuleDescription

**Instruction** The LongName of the *BswModuleDescription* shall contain a meaningful description.



For AUTOSAR BSW modules the LongName shall be set identical to the name given in [Chapter C "List of Basic Software Modules" p. 395](#) . Here the name specified in column "Modules Short Name" has to be used. For Non-AUTOSAR BSW modules also a meaningful name has to be specified and used consistently.

For the definition of a long name multiple languages are possible. By default a long name has to be specified in English. Other languages could be requested but currently only a long name in English has to be provided.

Example:

```
<LONG-NAME>
  <L-4 L="EN">ADC Driver</L-4>
</LONG-NAME>
```

The LongName is relevant for the generation of the A2L file and is also relevant for documentation in general.

## Rule BSWMD\_ModuleDesc\_004: Category of BswModuleDescription

**Instruction** Within the *BswModuleDescription* a Category has to be set using the tag <CATEGORY>.

The following three settings are possible: *BSW\_MODULE*, *BSW\_CLUSTER* or *LIBRARY*.

The generic category attribute shall be used for a general classification of a *BswModuleDescription* as shown in the following table.

Table 34 Selectable Values for Category of BswModuleDescription

Category Value	Explanation
<b>BSW_MODULE</b>	Specifies a single BSW module (default)
<b>BSW_CLUSTER</b>	Specifies a BSW module cluster
<b>LIBRARY</b>	Specifies a Library (Restricted to BSW modules which are classified as library)

The main difference between a library and a "normal" BSW module is, that library services can directly be called from application SWCs without going via the RTE. As a consequence, there will be certain restrictions on the model elements which can be used for libraries, e.g. a library should not have scheduled functions. The category LIBRARY shall only be used from that BSW modules which are classified as libraries in appendix [Chapter C "List of Basic Software Modules" p. 395](#) .

## Rule BSWMD\_ModuleDesc\_005: BSW Module Identifier

**Instruction**

**Scope:** Software based on an AUTOSAR SWS

Within the *BswModuleDescription* a module identifier shall be set within the tag <MODULE-ID>.

This tag shall refer to the identifier of the standardized AUTOSAR modules. Valid module identifiers of AUTOSAR based BSW modules are listed in [Chapter C "List of Basic Software Modules" p. 395](#) .

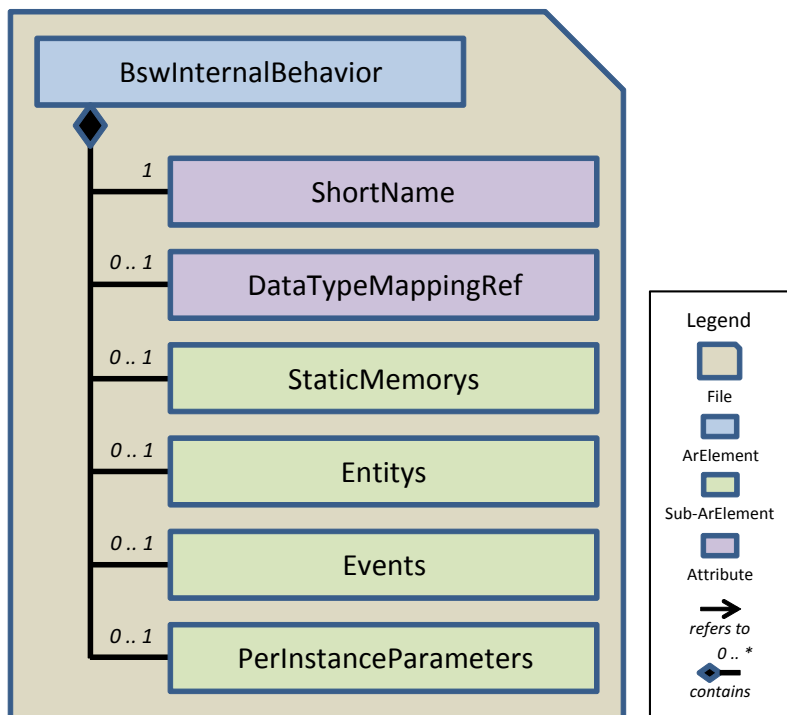
This rule is similar to rule [\[BSW\\_VersionInfo\\_003\] p. 114](#). Here the same requirement is given for the module header. It is obvious that the module id in the module header file and in the BSWMD file shall be identical.

### 8.2.1.2 BswInternalBehavior

The *BswInternalBehavior* describes internal aspects of a BSW module. For example the properties of BSW module interfaces are described within the *BswInternalBehavior* (this is done with the *BswModuleEntity*) and also the events that can start a scheduled entity. More details about those issues are given in the chapter "*BSWMD Use Cases*" p. 237 .

In *Figure 51* an overview of a *BswInternalBehavior* is given. Only those elements which are currently on focus and are described within the BSW Coding Guidelines. Updates will be done gradually.

Figure 51 Details of BswInternalBehavior



The following rules define the handling of common attributes of the *BswInternalBehavior*. Sub-ArElements are not described in detail. This is done in the description of the single use cases in chapter 8.3 "*BSWMD Use Cases*". The following ARXML snippet shows an example for the AUTOSAR BSW module Adc.

```
<INTERNAL-BEHAVIORS>
  <BSW-INTERNAL-BEHAVIOR>
    <SHORT-NAME>BswInternalBehavior</SHORT-NAME>    <!-- Rule [BSWMD_IntBehav_001] -->
    <DATA-TYPE-MAPPING-REFS>                          <!-- Rule [BSWMD_IntBehav_002] -->
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
        /AUTOSAR_Adc/DataTypeMappingSets/DataTypeMappingSet
      </DATA-TYPE-MAPPING-REF>
    </DATA-TYPE-MAPPING-REFS>

    <!-- Use Case specific contents: -->
    <STATIC_MEMORYS>                                  <!-- Measurement Data -->
      ...                                              <!-- Refer to Chapter 8.3.1.1 p. 239 -->
    </STATIC_MEMORYS>
    <ENTITYS>                                          <!-- Properties of Code Fragments -->
      ...                                              <!-- Refer to Chapter 8.3.2.1 p. 244 -->
    </ENTITYS>
    <EVENTS>                                          <!-- BSW Module Events -->
      ...                                              <!-- Refer to Chapter 8.3.2.1 p. 244 -->
    </EVENTS>
    <PER-INSTANCE-PARAMETERS>                        <!-- Calibration Data -->
      ...                                              <!-- Refer to Chapter 8.3.1.2 p. 240 -->
    </PER-INSTANCE-PARAMETERS>
```

```
</BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
```

**Hint** Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined in rule [\[BSWMD\\_Common\\_005\] p. 220](#)). Otherwise the tool cannot interpret the BSWMD file correctly.

**Hint** The `DataTypeMappingRef` has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the `DatatypeMappingRef` is written in three lines only for clearness.

## Rule BSWMD\_IntBehav\_001: ShortName of the BswInternalBehavior

**Instruction** The `ShortName` of the *BswInternalBehavior* shall be set to "BswInternalBehavior".

## Rule BSWMD\_IntBehav\_002: Reference to DataTypeMapping

**Instruction** If a BSW module specifies *ApplicationDataTypes* then a reference(s) to the `DataTypeMappingSet`(s) shall be set within the *BswInternalBehavior*. The following settings shall be made:

- ▶ The `DataTypeMapping` shall be set using the tag `<DATA-TYPE-MAPPING-REF>` containing the destination type "DATA-TYPE-MAPPING-SET"
- ▶ The reference shall contain a complete `ARPackage` path to the `DataTypeMapping`
- ▶ The `DataTypeMapping` shall be enclosed from the tags `<DATA-TYPE-MAPPING-REFS>`

*ApplicationDataTypes* and the `DataTypeMappingSet` are specified outside of the *BswInternalBehavior* in own *ArPackage*(s). More than one `DataTypeMappingSet` may be defined. In this case multiple references has to be set. For more details about that topic refer to chapter [7 "Rule Set: Data Description"](#).

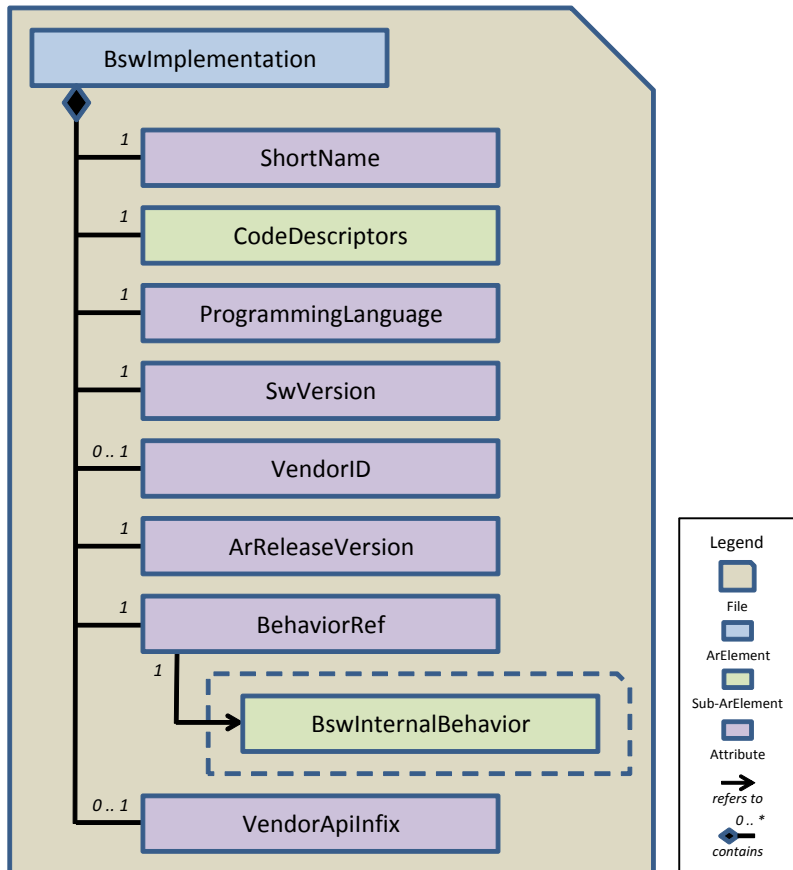
The `DataTypeMappingRef` has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file.

```
<DATA-TYPE-MAPPING-REFS>
  <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">...ARPackagePath...</DATA-TYPE-MAPPING-REF>
</DATA-TYPE-MAPPING-REFS>
```

### 8.2.1.3 BswImplementation

The *BswImplementation* describes the kind of implementation of the BSW module. It contains common properties of a BSW module which also can be used by an AUTOSAR conformance test. In [Figure 52](#) an overview of a BswImplementation is given. Only those elements are shown which are currently on focus and are described within the BSW Coding Guidelines. Updates will be done gradually.

Figure 52 Details of BswImplementation



The following rules define the handling of common attributes of the *BswImplementation*. Currently no further sub elements are on focus. All relevant attributes are described in the current chapter. The following ARXML snippet shows an example for the AUTOSAR BSW module Adc.

```

<AR-PACKAGE>
  <SHORT-NAME>Adc</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>BswImplementations</SHORT-NAME>
      <ELEMENTS>
        <BSW-IMPLEMENTATION>
          <SHORT-NAME>Adc</SHORT-NAME>
          <CODE-DESCRIPTORS>
            <CODE>
              <SHORT-NAME>CodeDescriptor</SHORT-NAME>
              <ARTIFACT-DESCRIPTORS>
                <AUTOSAR-ENGINEERING-OBJECT>
                  <SHORT-LABEL>ArEngObj</SHORT-LABEL>
                  <CATEGORY>SWSRC</CATEGORY>
                </AUTOSAR-ENGINEERING-OBJECT>
              </ARTIFACT-DESCRIPTORS>
            </CODE>
          </CODE-DESCRIPTORS>
        </BSW-IMPLEMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
  
```

<!-- Rule [BSWMD\_Impl\_001] -->  
 <!-- Rule [BSWMD\_Impl\_002] -->  
 <!-- Rule [BSWMD\_Impl\_004] -->

```

05      <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>      <!-- Rule [BSWMD_Impl_005] -->
      <SW-VERSION>1.0.0</SW-VERSION>      <!-- Rule [BSWMD_Impl_006] -->
      <VENDOR-ID>6</VENDOR-ID>      <!-- Rule [BSWMD_Impl_003] -->
      <AR-RELEASE-VERSION>4.0.3</AR-RELEASE-VERSION>      <!-- Rule [BSWMD_Impl_007] -->
      <BEHAVIOR-REF DEST="BSW-INTERNAL-BEHAVIOR">      <!-- Rule [BSWMD_Impl_008] -->
10          /AUTOSAR_Adc/Adc/BswInternalBehavior
      </BEHAVIOR-REF>
      <VENDOR-API-INFIX>SpecificName</VENDOR-API-INFIX>      <!-- Rule [BSWMD_Impl_003] -->
      </BSW-IMPLEMENTATION>
      </ELEMENTS>
15      </AR-PACKAGE>
      </AR-PACKAGES>
      </AR-PACKAGE>

```

**Hint** Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined in rule [\[BSWMD\\_Common\\_005\] p. 220](#)). Otherwise the tool cannot interpret the BSWMD file correctly.

**Hint** The BehaviorRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the BehaviorRef is written in three lines only for clearness.

## Rule BSWMD\_Impl\_001: ShortName of the ArPackage Child Element BswImplementation

**Instruction** The ShortName of the *ArPackage* for the *BswImplementation* shall be set to "BswImplementations".

This rule is derived from the rule [\[ARPac\\_44\] p. 301](#) .

```

...
<AR-PACKAGE>
  <SHORT-NAME>BswImplementations</SHORT-NAME>
  <ELEMENTS>
    <BSW-IMPLEMENTATION>
    ...

```

## Rule BSWMD\_Impl\_002: ShortName of the BswImplementation

**Instruction** The ShortName of the *BswImplementation* shall be identical to the module prefix of the BSW module.

The module prefix shall be built conform to rule [\[CDGNaming\\_001\] p. 80](#) but without the case that the module prefix contains a VendorId or BOSCH specific name (VendorApiInfix). Possible existing VendorIds and VendorApiInfixes will be specified with special tags of the *BswImplementation* (see rule [\[BSWMD\\_Impl\\_003\] p. 229](#) ) and are not set in the short name representing the module prefix.

## Rule BSWMD\_Impl\_003: Specification of a VendorId and a VendorApiInfix

### Instruction

**Scope:** If there are multiple instances of BSW driver modules available (>1) based on AUTOSAR SWS

A vendor identification shall be specified with tag <VENDOR-ID>.

Additionally a specific name shall be specified as VendorApiInfix using the tag <VENDOR-API-INFIX>.

This rule is only applicable for instances of BSW driver modules which are based on AUTOSAR SWS specifications. Here it is necessary to avoid naming conflicts because it is possible that different driver implementations provided by different vendors are linked together on the same ECU. Therefore the VendorId and VendorApiInfix have to be specified. For the sake of completeness it is also possible that one vendor provides different driver implementations. In this case the VendorId is identical but the VendorApiInfix shall be various.

But in contrast to the definition of the module prefix on C language level (the third use case of rule [\[CDGNaming\\_001\] p. 80](#)) the ARXML-based description provides additional tags to specify the VendorId and the VendorApiInfix. Both information are not integral parts of the short name representing the module prefix. In all relevant cases (e.g. file names, published parameters and memory allocation keywords) the RTE generator will generate complete module prefixes based on the given ShortName, VendorId and VendorApiInfix.

For BOSCH the Vendor-Id has to be set to the value "6" as it is specified by HIS. The VendorApiInfix is an additional name which has to be specified conform to the commitment specified in rule [\[CDGNaming\\_001\] p. 80](#). It is not allowed to specify a VendorId and VendorApiInfix for specific modules within the name space of rba\_. The tags <VENDOR-ID> and <VENDOR-API-INFIX> shall only be set if there are really instances of AUTOSAR based BSW driver modules are implemented. In all other cases both tags shall not be set.

## Rule BSWMD\_Impl\_004: Provision of a Code Descriptor

**Instruction** For each BswImplementation a code descriptor shall be provided. The following ARXML snippet shall be used:

```
<CODE-DESCRIPTORS>
  <CODE>
    <SHORT-NAME>CodeDescriptor</SHORT-NAME>
    <ARTIFACT-DESCRIPTORS>
      <AUTOSAR-ENGINEERING-OBJECT>
        <SHORT-LABEL>ArEngObj</SHORT-LABEL>
        <CATEGORY>SWSRC</CATEGORY>
      </AUTOSAR-ENGINEERING-OBJECT>
    </ARTIFACT-DESCRIPTORS>
  </CODE>
</CODE-DESCRIPTORS>
```

The code descriptor is relevant for the RTE generator. The code descriptor declares that the code of the BSW module is provided as source code (category value SWSRC) or as object code (category value SWOBJ). Depending on the kind of code the RTE generator generates function definitions in the file Rte.c for each API which is handled by the RTE. This is done if the code is provided as object code which is also the default setting for the RTE generator if no code descriptor is set. But usually BSW modules are delivered as source code and not as object code. Therefore it is necessary to provide the code descriptor with the category SWSRC. This will reduce the resource consumption because BSW module APIs are defined from the BSW module itself and there is no need that the RTE generator creates additional function definitions.

## Rule BSWMD\_Impl\_005: Documentation of Programming Language

**Instruction** Within the BswImplementation the programming language shall be specified. The related tag <PROGRAMMING-LANGUAGE> shall be set to "C".

The schema allows in general following values for the programming language tag: "C", "CPP" or "JAVA". Because it is currently only allowed to program a BSW module conform to C (see rule [\[CCode\\_001\] p. 17](#)) the tag <PROGRAMMING-LANGUAGE> has to be set allways to "C".

## Rule BSWMD\_Impl\_006: Provision of Software Version Number

**Instruction** A software version number shall be provided within the BswImplementation using the tag <SW-VERSION>. The value shall be set conform to the schema "<Major>.<Minor>.<Patch>" where Major, Minor and Patch represent an integer number.

The software version number shall be updated even before a new version of the software will be released.

The version numbers of AUTOSAR Basic Software Modules shall be enumerated according to the following rules:

- The major version is incremented if the module is not compatible any more (e.g. existing API changed)

- ▶ The minor version is incremented if the module is still downwards compatible (e.g. new functionality added)
  - ▶ The patch version is incremented if the module is still upwards and downwards compatible (e.g. bug fixed)
- Increasing a more significant digit of a version number resets all less significant digits.

Example:

```
<SW-VERSION>1.0.0</SW-VERSION>
```

The software version number shall also be conform to the vendor specific versioning which is used in the SCM.

**Hint** There is a similar rule to provide version information also in the module header file: Rule [\[BSW\\_VersionInfo\\_004\]](#) p. 114. Please consider that the version information given in the module header is consistent to the version information given in the BSWMD file.

## Rule BSWMD\_Impl\_007: Provision of AUTOSAR Release Version Number

### Instruction

**Scope:** Software based on an AUTOSAR SWS

The inclusion of the AUTOSAR version information within the *BswImplementation* shall be given using the tag <AR--RELEASE-VERSION>. The value shall be set conform to the schema "<Major>.<Minor>.<Patch>" where Major, Minor and Patch represent an integer number.

The AUTOSAR version information shall be updated even if the module supports a new version of an AUTOSAR specification.

The AUTOSAR version information shows the version of the AUTOSAR specification which an appropriate implementation of a BSW module is based on.

Example:

```
<AR-RELEASE-VERSION>4.0.3</AR-RELEASE-VERSION>
```

**Hint** There is a similar rule to provide version information also in the module header file: Rule [\[BSW\\_VersionInfo\\_004\]](#) p. 114. Please consider that the version information given in the module header is consistent to the version information given in the BSWMD file.

## Rule BSWMD\_Impl\_008: Reference to BswInternalBehavior

**Instruction** A reference to the *BswInternalBehavior* (with a complete ARPackage path) shall be specified within the *BswImplementation* using the tag <BEHAVIOR-REF> containing the destination type "BSW-INTERNAL-BEHAVIOR".

The reference is the link between the *BswInternalBehavior* and *BswImplementation* which is also illustrated in [Figure 49](#).

The BehaviorRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file.

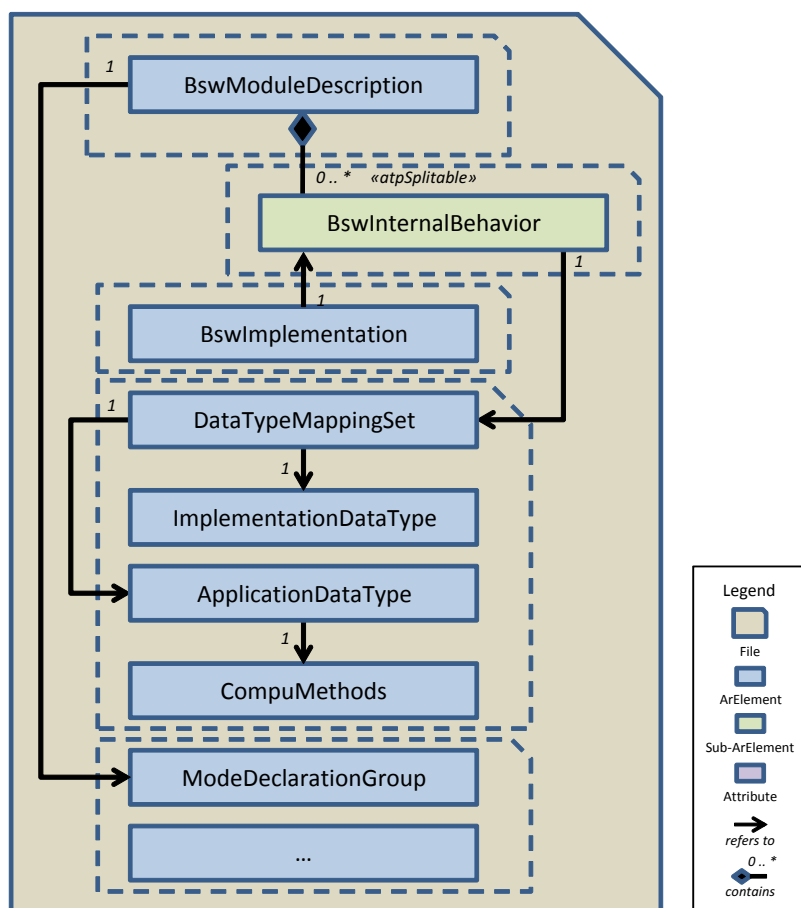
```
<BEHAVIOR-REF DEST="BSW-INTERNAL-BEHAVIOR">...ARPackagePath...</BEHAVIOR-REF>
```

## 8.2.2 Splitting of BSWMD Files

In addition to the top layers of a BSWMD (*BswModuleDescription*, *BswInternalBehavior* and *BswImplementation*) which are described in chapter 8.2.1 "Top Layers of BSWMD" a lot of other ArElements can be located in a BSWMD file and exist in parallel to the *BswModuleDescription* and *BswImplementation* (e.g. *ApplicationDataTypes*, *ImplementationDataTypes*, *DataTypeMappingSet*, ... and many more). These elements are also top level ArElements of ArPackages and could generally be split into different files. But it is not the goal to have many BSWMD files for one BSW module, a split shall only be done if there is a need to do it e.g. if a use case demands it. Rule [\[BSWMD\\_Split\\_001\] p. 233](#) will handle that point. Additionally the split of BSWMD files is limited to some restrictions which are explained in this chapter.

The relations between the different ArElements provide information about the possibility to split their contents. Two different forms of relations are available: aggregations and references. In normal case aggregations cannot be split because they connect internal sub-parts of existing ArElements. The other form of relations are references. They are using *ARPackage* paths to establish the connection between two elements and if the path is correctly and completely specified the referenced element could be located in another file. [Figure 53](#) shows an rough overview of the file splitting context containing the different relations (But not all ArElements and relations are shown). The figure will not propose that the file splitting (indicated with the dotted file symbols) shall be made in the illustrated form. It shows only the possibility of file splitting.

Figure 53 Overview Splitting of BSWMD Files



The *BswInternalBehavior* is aggregated to the *BswModuleDescription* with an aggregation relation. Normally this aggregation cannot be split but AUTOSAR specifies here a special case. This case is marked as *«atpSplitable»* and means that it is possible to distribute the aggregated elements over several physical files. The *BswInternalBehavior* can now be split from the *BswModuleDescription*. Details and restrictions of this file split are described in the rules [\[BSWMD\\_Split\\_002\] p. 233](#) and [\[BSWMD\\_Split\\_003\] p. 234](#).



The BswImplementation is connected by a reference relation to the BswInternalBehavior. Thus, in general it is possible to keep the BswImplementation and the BswInternalBehavior in separate files. This could be done if there are technical reasons otherwise the BswImplementation should be located together with the BswModuleDescription or BswInternalBehavior in only one BSWMD file. The rules [\[BSWMD\\_Split\\_004\] p. 235](#) and [\[BSWMD\\_Split\\_005\] p. 235](#) will give more details for this topic.

The last option to split files are ArPackages containing ArElements and is described in rule [\[BSWMD\\_Split\\_006\]](#).

## Rule BSWMD\_Split\_001: Main Rule of Splitting of BSWMD Files

**Instruction** It is possible to create multiple BSWMD files. But there shall exist as few BSWMD files as possible for a single BSW module. Splitting of BSWMD files is optional and shall only be done if it cannot be prevented (e.g. if there is a technical reason or use case to do that).

At best, only a single BSWMD file is provided for a single BSW module. But sometimes this is not possible because e.g. some BSWMD contents could be dependent on ECU configuration and other contents are static. This is a use case which requires that the BSWMD contents are split up to different files (static and generated ones). This increases the flexibility for a software developer. However, the following rules which specify the relevant restrictions have to be kept.

## Rule BSWMD\_Split\_002: Possibility to Split BswInternalBehavior from BswModuleDescription

**Instruction** It is possible to split the *BswInternalBehavior* from the *BswModuleDescription* to a separate BSWMD file. Splitting of BSWMD files is optional and shall only be done if there is a technical reason or use case to do that.

If a split is done it shall be ensured that the structure of the corresponding BSWMD files is correct:

- ▶ In the BSWMD file containing the BswModuleDescription only the relevant attributes and elements of the BswModuleDescription shall be specified, but without parts of the BswInternalBehavior.
- ▶ In the BSWMD file containing the relevant attributes and elements of the BswInternalBehavior the BswInternalBehavior shall be embedded inside the BswModuleDescription and only the ShortName of the BswModuleDescription shall be repeated.

The BswInternalBehavior is aggregated to the BswModuleDescription. That means that the BswInternalBehavior represents a sub-element of the BswModuleDescription. The BswModuleDescription itself is a top level ArElement of an ArPackage. The following ARXML example is simplified and focuses on the structure of the relation of BswInternalBehavior and BswModuleDescription:

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
    <ELEMENTS>
      <BSW-MODULE-DESCRIPTION>
        <SHORT-NAME>ModuleName</SHORT-NAME>
        ...
        <INTERNAL-BEHAVIORS>
          <BSW-INTERNAL-BEHAVIOR>
            ...
            <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
            ...
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
        ...
      </BSW-MODULE-DESCRIPTION>
      ...
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
```

```
<!-- BswModuleDescription is a -->
<!-- top level ArElement of an -->
<!-- ArPackage. -->

<!-- BswInternalBehavior is a -->
<!-- part of InternalBehavior -->
<!-- which is a sub element of -->
<!-- the BswModuleDescription -->
```

Usually such an ARXML structures cannot be split but here AUTOSAR admits that the *BswInternalBehavior* can be split from the *BswModuleDescription*. This special case is denominated as «*atpSplitable*» and allows that the *BswModuleDescription* and the *BswInternalBehavior* can be split into two different BSWMD files. From the model point of view the *BswInternalBehavior* is still a sub-element of the *BswModuleDescription*. This association is still reflected in the file containing the *BswInternalBehavior* due to the fact that the *BswInternalBehavior* is embedded inside the *BswModuleDescription* construct. The following example will show a rough structure of both files with respect to the file split:

BSWMD file containing the *BswModuleDescription*:

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
    <ELEMENTS>
      <BSW-MODULE-DESCRIPTION>
        <SHORT-NAME>ModuleName</SHORT-NAME>
        ...
        <!-- Specify all relevant attributes and elements -->
        <!-- of the BswModuleDescription here -->
        <!-- Details are specified in Chapter 8.2.1.1 p. 223 -->
        ...
      </BSW-MODULE-DESCRIPTION>
      ...
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
```

BSWMD file containing the *BswInternalBehavior*:

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
    <ELEMENTS>
      <BSW-MODULE-DESCRIPTION>
        <SHORT-NAME>ModuleName</SHORT-NAME>
        <!-- Only the ShortName of the BswModuleDescription -->
        <!-- shall be given but nothing more -->
        <INTERNAL-BEHAVIORS>
          <BSW-INTERNAL-BEHAVIOR>
            <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
            ...
            <!-- Specify all relevant attributes and elements -->
            <!-- of the BswInternalBehavior here -->
            <!-- Details are specified in Chapter 8.2.1.2 p. 226 -->
            ...
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </BSW-MODULE-DESCRIPTION>
      ...
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
```

### Rule BSWMD\_Split\_003: *BswInternalBehavior* is not Splitable

**Instruction** It is not allowed to split the *BswInternalBehavior* into different files. All attributes and subelements of the *BswInternalBehavior* shall be only specified in a single BSWMD file.

The *BswInternalBehavior* is not a top level *ArElement* of an *ArPackage* and contains no *ArElements* in it. No sub-element is marked as «*atpSplitable*» and therefore all elements of the *BswInternalBehavior* are only located in a single BSWMD file.

This could be a hard constraint even if it is intended to generate parts of the *BswInternalBehavior*. For example within the *BswInternalBehavior* calibration parameters are specified. It could be that some of these calibration parameter are dependent on a specific ECU configuration. Therefore with an ECU configuration generator more or less definitions of calibration parameters could be generated dependent on that specific ECU configuration values. Other parts of the *BswInternalBehavior* could be not dependent to an ECU configuration and exist permanently. But because of that the complete *BswInternalBehavior* is not splittable, all parts have to be located in a single BSWMD file. In summary this means that if only one part of a *BswInternalBehavior* can be generated by an ECU configuration generator all the other (static) parts have to be generated, too. It is not possible to have a BSWMD file containing static parts of the *BswInternalBehavior* and another BSWMD file containing the dynamic parts which depend on the ECU configuration values.

Details about the contents of the *BswImplementation* are given in chapter [8.2.1.2 "BswInternalBehavior" p. 226](#).

## Rule BSWMD\_Split\_004: Splitting of BswImplementation to a Single BSWMD File

**Instruction** The *BswImplementation* may be split to a separate BSWMD file and can exist in parallel to a BSWMD file containing the *BswInternalBehavior* (and/or the *BswModuleDescription*). The split is optional and shall only be done if there is a technical reason to do that.

The split of the *BswImplementation* to a discrete BSWMD file is possible because the *BswImplementation* is a top level *ArElement* within an *ArPackage*. Also in contrast to the aggregation relation between the *BswModuleDescription* and the *BswInternalBehavior* there is only a reference relation between the *BswInternalBehavior* and the *BswImplementation* (shown in [Figure 49 p. 222](#)). Therefore the *BswImplementation* is self-contained and could be split to a separate file. A usecase could be if there are multiply instantiated BSW driver modules are available. In this case multiple *BswImplementations* exist because the *VendorApiInfix* needs to be unambiguously specified (see rule [\[BSWMD\\_Impl\\_003\] p. 229](#)). But in all other cases primarily, the *BswImplementation* shall be located together with the *BswModuleDescription* and/or the *BswInternalBehavior* in a single BSWMD file. Details about the contents of the *BswImplementation* are given in chapter [8.2.1.3 "BswImplementation" p. 228](#).

## Rule BSWMD\_Split\_005: Contents of BswImplementation are not Splittable

**Instruction** Contents of the *BswImplementation* shall not be split into different files. All attributes and elements of the *BswImplementation* shall be specified in the same BSWMD file.

AUTOSAR does not allow to split the *BswImplementation*. Therefore all elements of the *BswImplementation* have to be located in the same BSWMD file. Details about the contents of the *BswImplementation* are given in chapter [8.2.1.3 "BswImplementation" p. 228](#).

## Rule BSWMD\_Split\_006: Splitting of Other BSWMD Specific ArPackages do Single BSWMD Files

**Instruction** *ArPackages* containing *ArElements* may be split to separate BSWMD files. The split is optional and shall only be done if there is a technical reason to do that.

An example for such an *ArPackage* is *ImplementationDataTypes*. It contains an *<ELEMENTS>* tag structure and specifies individual *ImplementationDataTypes*, as the following example illustrates:

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
    <ELEMENTS>                                <!-- Elements structure within an ARPackage -->
      <IMPLEMENTATION-DATA-TYPE>              <!-- Individual and complete definition of -->
        <SHORT-NAME>...</SHORTNAME>          <!-- an ImplementationDataType as ArElement -->
        ...
      </IMPLEMENTATION-DATA-TYPE>
      <IMPLEMENTATION-DATA-TYPE>              <!-- Another definition of an ImplDataType -->
        <SHORT-NAME>...</SHORTNAME>
        ...
```

```
</IMPLEMENTATION-DATA-TYPE>
...
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
```

This kind of ArPackages could be split into separate files if there is a technical reason to do that, e.g. if some ArElements depend on ECU configuration and others are static. The static and the generated file must both contain completely defined ArElements. Relating to the example above, each *ImplementationDataTypes* has to be specified completely with all relevant attributes and elements. The definition of a single *ImplementationDataType* cannot be split to different BSWMD files.

More details about the content of such ArPackages could be found in chapters 7 "[Rule Set: Data Description](#)" p. 179 and 8.3 "[BSWMD Use Cases](#)" p. 237 .

## 8.2.3 Differences between BSWMD and SWCD

In general a BSW module can be described with a BSWMD file with less methods on ARXML modeling level compared to an ASW component. The reason for that is e.g. that the connection to the *RTE* is more simple and a lot of the communication between the modules is done by the exchange of header files. It is not a secret that with a BSWMD file the same things can be described as with a SWCD file. But there are still two file formats available to focus the user to the specific behavior of a BSW module or ASW component. In most cases a BSWMD file is enough to describe a BSW module. But in some cases a SWCD is needed to support following use cases:

- ▶ A BSW module provides AUTOSAR interfaces or Standardized AUTOSAR Interfaces which are handled by the *RTE*
- ▶ A communication by sender receiver or by client server is needed
- ▶ If ports are needed

All other use cases can be handled by a BSWMD ARXML file and therefore the BSWMD is the standard file for data description of a BSW module. For more details about the SWCD ARXML file and the handling of its use cases take a look to the ArCaDe guideline (AUTOSAR Coding and Data Description Guideline, [\[G\\_ArCaDe\]](#) ) which describes all use cases of the application software ASW.

## 8.3 BSWMD Use Cases

The present rule set describes which parts of the BSWMD shall be used and how they shall be used. In general the following use cases can be described within the BSWMD:

- ▶ BSW Measurement and Calibration Support (\*)
- ▶ Scheduling of BSW via *RTE* (\*)
- ▶ BSW Documentation
- ▶ BSW Module API Description (\*)
- ▶ BSW Service Needs
- ▶ BSW Exclusive Areas

Currently the focus is set on the elements marked with a (\*). Other use cases are handled with later versions of the BSW Coding Guidelines.

### 8.3.1 BSW Measurement and Calibration Support

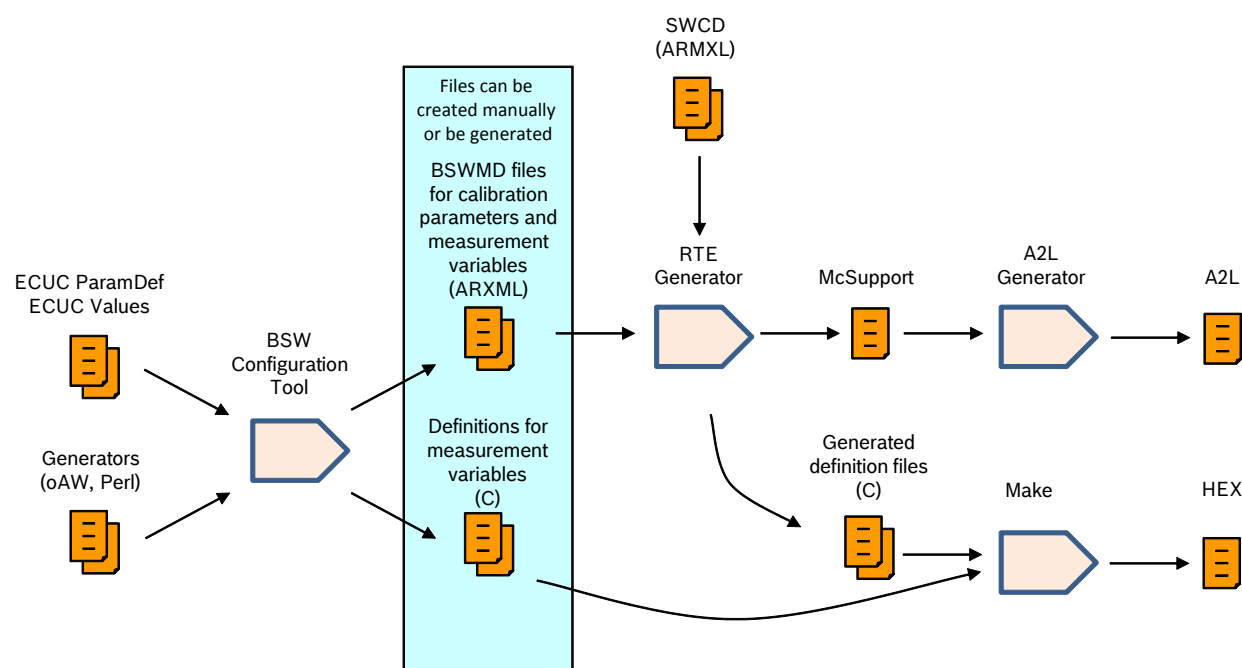
Starting with AUTOSAR release 4.0.3 there is a possibility to describe calibration parameters in BSWMD files very similar to application software in SWCD files. Previous AUTOSAR releases did not have a sufficient support for calibration parameters of BSW modules. The relevant point is that now calibration parameters can be defined as PerInstance-Parameters as part of the *BswInternalBehavior*. Also the *RTE* generator is able to generate the C code for the representation of the calibration parameters in the same way as it is done for application software. Therefore the BSW module developer has only to fill the relevant parts in a BSWMD file. On this way the *RTE* generator also supports the generation of pointer structures for the DSERAP calibration method (which is not further handled in this guideline).

Currently, a similar method for measurement variables does not exist. But with AUTOSAR release 4.0.3 measurement variables can be defined in a BSWMD file using *StaticMemory*s as part of the *BswInternalBehavior*. In parallel the BSW module developer has to provide a module c file containing the definitions for the measurement variables. In this case the *RTE* generator does not generate the representation of measurements into a generated C file.

Instead the *RTE* generator generates a *McSupport* file based on the ARXML files providing the calibration parameters and measurement variables. The *McSupport* file is then used as input for the generation of the A2L file which is required from calibration tools. The C files (generated from the *RTE* generator or for measurement variables provided by the BSW module) are used by a make run to create the *HEX* file which contains the actual code for the ECU.

In [Figure 54](#) the context is shown in a visual form. It shows the main data flow for A2L file and *HEX* file generation for a BSW module using AUTOSAR 4.0.3. The option that the BSWMD files and the C files for the measurement variables can be created manually or be generated by scripts of the ECU configuration process is also illustrated.

Figure 54 Dataflow for BSW Measurement and Calibration Support



### 8.3.1.1 Definition of Measurement Variables

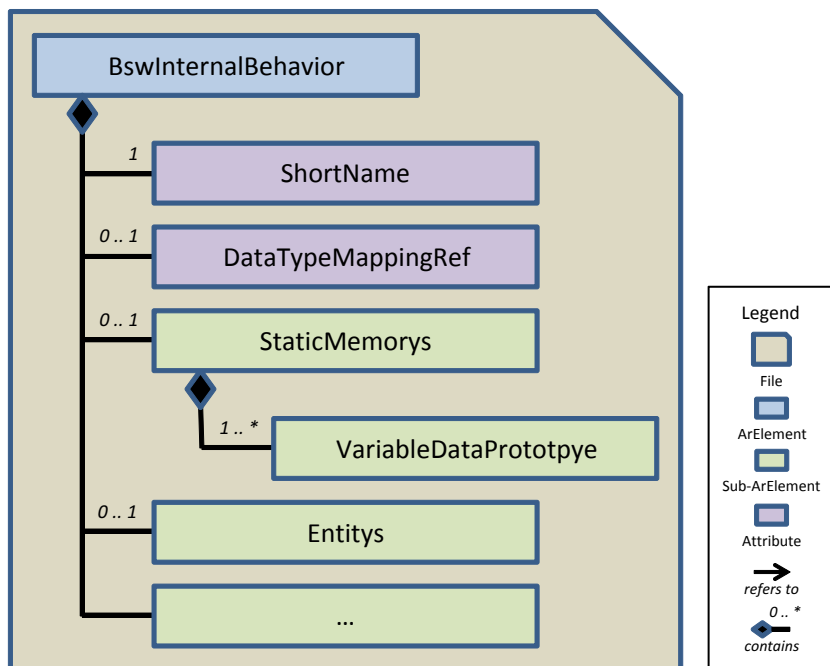
This chapter defines how measurement variables have to be defined.

#### Rule BSWMD\_MCSupport\_001: Definition of Measurement Variables in BSWMD

**Instruction** Measurement variables which shall exist in the A2L file shall be defined using *StaticMemorys* as part of the *BswInternalBehavior* in a BSWMD file.

Within the *StaticMemorys* measurement variables are defined using *VariableDataPrototypes*. Measurement variables can be specified for different kinds of elements like values, arrays, value blocks and structures. How a *VariableDataPrototype* has to be specified is described in [Chapter 7.2.2 "VariableDataPrototype\(Measurement Points\)" p. 198](#). [Figure 55](#) shows at what position the *StaticMemorys* element has to be placed within the *BswInternalBehavior*. The *StaticMemorys* element has to be set after the attribute 'DataTypeMappingRef' and before the Sub-ArElement 'Entitys'. A complete overview of the *BswInternalBehavior* is given in [Chapter 8.2.1.2 p. 226](#). The figure also shows that the *StaticMemory* has to be set only once if measurement variables shall be defined and then inside of the *StaticMemorys* multiple definitions of *VariableDataPrototypes* can be set.

Figure 55 Overview for Specification of StaticMemorys



Example for the definition of measurement variables as part of the *BswInternalBehavior*:

```

<INTERNAL-BEHAVIORS>
<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  <DATA-TYPE-MAPPING-REFS>...</DATA-TYPE-MAPPING-REFS>
  <STATIC-MEMORYS>                                <!-- StaticMemorys used for measurement variables -->
    <VARIABLE-DATA-PROTOTYPE>                      <!-- First measurement variable -->
      ...                                           <!-- Details are in Chapter 7.2.2 p. 198 -->
    </VARIABLE-DATA-PROTOTYPE>
    <VARIABLE-DATA-PROTOTYPE>                      <!-- Second measurement variable -->
      ...                                           <!-- Details are in Chapter 7.2.2 p. 198 -->
    </VARIABLE-DATA-PROTOTYPE>
    ...                                           <!-- And so on -->
  </STATIC-MEMORY>
  <ENTITYS>
    ...
  
```

```

05     </ENTITYS>
        ...
    </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>

```

## Rule BSWMD\_MCSupport\_002: Definition of Measurement Variables in C Files

**Instruction** The representation of measurement variables shall be done in a module C file using the memory mapping concept method. It shall be ensured that the definitions in the module C file match to the definitions in the module BSWMD file (regarding data type and name of the measurement variable).

For measurement variables the part in the BSWMD file and the other part in the C file shall match together. The SW developer of the BSW module is responsible to ensure that.

To define the measurement variables in a BSW module file the memory mapping concept has to be used. For more details to that concept refer to [Chapter "Abstraction of Memory Mapping" p. 50](#). For measurement variables special MemMap keywords are provided.

Example for representation of a measurement variable in a module's C file:

```

25 /* Memory location for 32 bit measurement variable (which is cleared after each reset): */
#define MODULE_START_SEC_INTERNAL_VAR_NO_INIT_32
#include "Module_MemMap.h"
uint32 Module_MeasurementPoint_MP;
#define MODULE_STOP_SEC_INTERNAL_VAR_NO_INIT_32
30 #include "Module_MemMap.h"

```

## Rule BSWMD\_MCSupport\_004: Naming Convention for Measurement Variables

**Instruction** The ShortName of a measurement variable shall be conform to following convention: <Component-Prefix>\_<pp><DescriptiveText>\_MP

With

- ▶ <ComponentPrefix> which represents the module prefix which has to be conform to rule [\[CDGNaming\\_001\] p. 80](#)
- ▶ <pp> which represents a physical and logical type which can be picked up from appendix ["Physical and Logical Types" p. 398](#)
- ▶ <DescriptiveText> is the description of the measurement variable (Camel case without underlines)
- ▶ \_MP is the characteristic suffix for a measurement variable

### 8.3.1.2 Definition of Calibration Parameters

This chapter defines how calibration parameters have to be defined.

## Rule BSWMD\_MCSupport\_007: Consideration of a Neutral Behavior of Calibration Related BSW Modules in Case of Development

### Instruction

**Scope:** BSW modules which are relevant for calibration

Regarding the *calibration* a BSW module shall have a neutral behavior. This means that an update of a BSW module shall have no effect during a calibration. To ensure that the following points shall be considered:

- ▶ New functionalities should be encapsulated with a new calibration parameter to be able to switch of the new functionality.



- ▶ New or extended calibration parameters shall be provided in such a way that their initial effect for the calibration is neutral. This shall be done by setting neutral initialization values whenever it is possible.
- ▶ An explanation shall be given to the delivery note to document the necessary calibration changes.

New or extended features in connection with new or extended calibration parameters have to be implemented but they shall not be active if possible. A BSW module with new or extended calibration parameters has to be provided in such a way that it has the same behavior as the predecessor regarding the calibration context. This is a feasible pre-condition that a *calibration* have not to be made after a new version of a BSW module is integrated to a program version. Init values for the calibration parameters have to be set to neutral values (values that deactivate a specific feature). How to set initialization values is described in "[Rule Set: Data Description](#)" p. 179. Finally meaningful descriptions has to be inserted to the delivery note to explain the calibration parameters and how to set them to a neutral behavior.

It is clear that this rule does not affect the first version of a BSW module (because there is no predecessor) or if completely changed or new calibration concepts are introduced.

### Rule BSWMD\_MCSupport\_003: Definition of Calibration Parameters in BSWMD

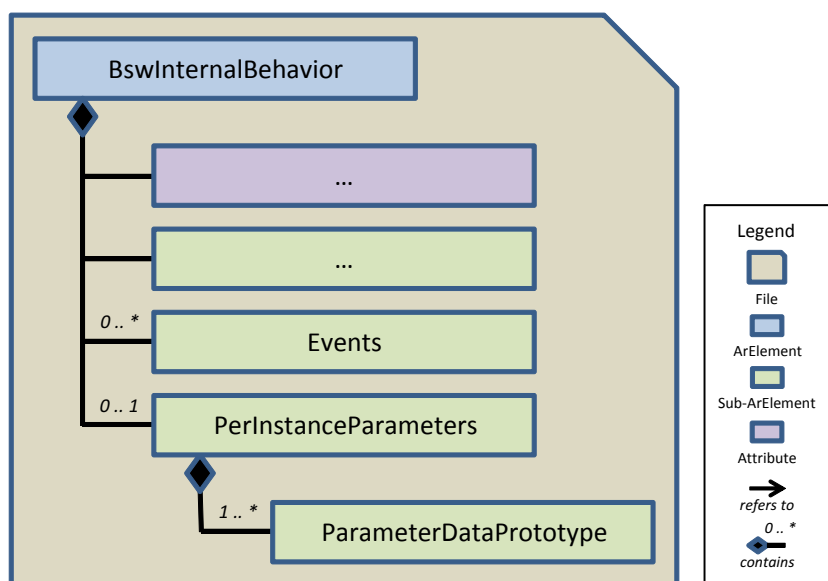
**Instruction** Calibration parameters which shall exist in the A2L file shall be defined using *PerInstanceParameters* as part of the *BswInternalBehavior* in a BSWMD file. The definition using *ConstantMemory*s, which is still available in AUTOSAR 4.0.3, shall NOT be used.

The representation of calibration data in a C file shall not be created by the SW developer because it is provided by the *RTE* generator.

Within the *PerInstanceParameters* calibration parameters are defined using *ParameterDataPrototypes*. This is done in analogy to a SWCD file. Calibration parameters can be specified for different kinds of elements like values, arrays, value blocks and structures. In general curves and maps are also possible but normally not used with BSW software. How a *ParameterDataPrototype* has to be specified is described in [Chapter 7.2.1 "ParameterDataPrototype \(Calibration Parameters\)"](#) p. 192. The *PerInstanceParameters* element has to be set at the end of the *BswInternalBehavior* after the definition of Events (if they exist). A complete overview of the *BswInternalBehavior* is given in [Chapter 8.2.1.2](#) p. 226. [Figure 56](#) shows all of the details.

**Hint** The reason to use *PerInstanceParameters* is that the *RTE* generates the entries in the *McSupport* file and generates also the relevant entries in the *Rte.c* file. Both aspects has not to be handled by the software developer. With *ConstantMemory*s only entries in the *McSupport* file are created but no typedefs in the C code. This causes also the link to AUTOSAR 4.0.3 because *PerInstanceParameters* within BSWMD files are possible for the first time with that release of AUTOSAR. At last the *RTE* also considers *DSERAP* concepts for calibration.

Figure 56 Overview for Specification of PerInstanceParameters



Example for the definition of calibration parameters as part of the BswInternalBehavior:

```

<INTERNAL-BEHAVIORS>
  <BSW-INTERNAL-BEHAVIOR>
    <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
    ...
    <EVENTS>
    ...
  </EVENTS>
  <PER-INSTANCE-PARAMETERS>
    <PARAMETER-DATA-PROTOTYPE>
      ...
    </PARAMETER-DATA-PROTOTYPE>
    <PARAMETER-DATA-PROTOTYPE>
      ...
    </PARAMETER-DATA-PROTOTYPE>
    ...
  </PER-INSTANCE-PARAMETERS>
</BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
  
```

<!-- PerInstanceParameters used for calib. parameters -->  
 <!-- First calibration parameter -->  
 <!-- Details are in Chapter 7.2.1 p. 192 -->  
 <!-- Second calibration parameter -->  
 <!-- Details are in Chapter 7.2.1 p. 192 -->  
 <!-- And so on -->

## Rule BSWMD\_MCSupport\_005: Naming Convention for Calibration Parameters

**Instruction** The ShortName of a calibration parameter shall be conform to following naming convention: <Component-Prefix>\_<pp><DescriptiveText>\_<EX>

With

- ▶ <ComponentPrefix> which represents the module prefix which has to be conform to rule [CDGNaming\_001] p. 80
- ▶ <pp> which represents a physical and logical type which can be picked up from appendix "Physical and Logical Types" p. 398
- ▶ <DescriptiveText> is the description of the calibration parameter
- ▶ <EX> is characteristic suffix for a calibration parameter and is conform to following table.

Table 35 Suffixes for Calibration Parameter

Suffix <EX>	Description
C	Calibration parameter

Suffix <EX>	Description
CA	Calibration parameter array
CST	Calibration parameter structure
CVB	Calibration parameter value block
ASC	Textual parameter containing ASCII characters instead of numbers
T / FT / GT	Characteristic curve / Fixed characteristic curve / Grouped characteristic curve
M / FM / GM	Characteristic map / Fixed characteristic map / Grouped characteristic map
AX	Axis definition

**Hint** Normally within BSW modules curves, maps and axis are not used.

## Rule BSWMD\_MCSupport\_006: Accessing a Calibration Parameter

**Instruction** To access a calibration parameter the following interface shall be used: SchM\_CData\_<Component-Prefix>[\_<vi>\_<ai>]\_<Name>().

Where here

- ▶ <ComponentPrefix> is the prefix of the BSW module which has to be conform to rule [\[CDGNaming\\_001\] p. 80](#)
- ▶ <Name> is the ShortName of the PerInstanceParameter, rule [\[BSWMD\\_MCSupport\\_003\]](#)
- ▶ [\_<vi>\_<ai>] is omitted where <vi> is the vendorId of the calling BSW module and <ai> is the vendorApiInfix of the calling BSW module

The SchM\_CData interface is generated from the RTE in the Module Interlink Header File SchM\_<Module>.h. This header has to be included to have access to the SchM\_CData interface.

Here is a short example:

In BSW module "Hugo" an uint8 calibration parameter "Hugo\_stValue\_C" is created conform to the naming convention for calibration parameters rule [\[BSWMD\\_MCSupport\\_005\]](#). Then the access to that calibration parameter shall be made by following code:

```
#include "SchM_Hugo.h"

uint8 Hugo_CalValue_u8;

Hugo_CalValue_u8 = SchM_Hugo_Hugo_stValue_C();
```

**Hint** It is a known redundancy that in the SchM\_CData interface the BSW module name appears two times.

## 8.3.2 Scheduling of BSW via RTE

The Basic Software Scheduler offers concepts and services to integrate Basic Software Modules. Hence, the Basic Software Scheduler embeds Basic Software Module implementations into the AUTOSAR OS context are applying data consistency mechanisms for the Basic Software Modules and communication modes between Basic Software Modules. (Due to the situation that the same OS Task might be used for the scheduling of ASW software components and basic software modules the scheduling part of the *RTE* is strongly linked with the Basic Software Scheduler and cannot be clearly separated.) and triggers main processing functions of the Basic Software Modules. Other jobs of the Basic Software Scheduler (which are not on focus of this chapter but stated for the sake of completeness.)

All Basic Software Scheduler symbols (e.g. function names, data types, etc.) belonging to the Basic Software Schedulers interfaces are using the SchM\_ prefix. The *RTE* will generate for each BSW module a Module Interlink Header File SchM\_<Module>.h. It has to be included in a BSW module to be able to access and use the Basic Software Scheduler elements. To be able to create the BSW Scheduler (SchM) header files the *RTE* generator shall get all the relevant inputs in form of a BSWMD ARXML file. Following topics are relevant for the scheduling tasks:

- ▶ BswModuleEntitys, BswModuleEntrys and BswEvents
- ▶ BSW Modes (for Mode Manager and Mode User)
- ▶ BswTriggers (External & Internal Triggers)

### 8.3.2.1 BswModuleEntitys, BswModuleEntrys and BswEvents

A BswModuleEntity describes properties of a code fragment and a BSW Entry describes the interface (i.e. the signature of a single C-function prototype) which is used to invoke a code fragment. Therefore both items belong together and it is mandatory that for each BswModuleEntity a BSW Entry exists. A BswEvent is only relevant for scheduled BswModuleEntitys and describe all kinds of events which can start a scheduled BswModuleEntity.

The lead element of these three items is the BswModuleEntity. Three different kinds of BswModuleEntitys can be distinguished:

- ▶ BswCalledEntity
- ▶ BswInterruptEntity
- ▶ BswSchedulableEntity

Following example shows the general definition of the Internal Behavior in a BSWMD ARXML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-DESCRIPTION>
              <SHORT-NAME>ModuleName</SHORT-NAME>
              ...
            <INTERNAL-BEHAVIORS>
              <BSW-INTERNAL-BEHAVIOR>
                <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                ...
              <ENTITYS>
```

```

05      <BSW-CALLED-ENTITY>
        <SHORT-NAME>BswCalledEntityName</SHORT-NAME>
        ...
      </BSW-CALLED-ENTITY>
      <BSW-INTERRUPT-ENTITY>
10      <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
        ...
      </BSW-INTERRUPT-ENTITY>
      <BSW-SCHEDULABLE-ENTITY>
        <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
15      ...
      </BSW-SCHEDULABLE-ENTITY>
    </ENTITYS>
    <EVENTS>
      ...
    </EVENTS>
20    ...
  </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
25 </AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
30

```

#### Remarks:

- ▶ BswSchedulableEntitys are scheduled by SchM which is generated by the RTE-Generator
- ▶ RTE-Generator doesn't schedule BSWCalledEntitys or BswInterruptEntitys
- ▶ Time Triggered BSW functions shall be modeled as BswSchedulableEntitys

### 8.3.2.1.1 BswCalledEntity

A BswCalledEntity is designed for a BSW function which is called from another BSW module or cluster. Two different kinds of BswCalledEntity can be modeled:

- ▶ Regular: modules API call from another BSW Module
- ▶ Callback: callback routine provided by the BSW module to other BSW modules

#### Rule BSWMD\_Sched\_001: Definition of BswCalledEntity

**Instruction** For each BSW Called Entity, a **BswCalledEntity** element shall be defined in a **BswInternalBehavior** and a **BswModuleEntry** shall be defined. The two model elements shall exist.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining BswCalledEntity.

```

60 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
65      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGES>

```

```

05    <AR-PACKAGE>
        <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        <ELEMENTS>
            <BSW-MODULE-DESCRIPTION>
                <SHORT-NAME>ModuleName</SHORT-NAME>
10            <INTERNAL-BEHAVIORS>
                <BSW-INTERNAL-BEHAVIOR>
                    <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                    ...
                <ENTITYS>
                    <BSW-CALLED-ENTITY>
15                        <SHORT-NAME>BswCalledEntityName</SHORT-NAME>
                        ...
                        <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
                            <!-- Refer to the Snippet of a BswModuleEntry -->
20                        </IMPLEMENTED-ENTRY-REF>
                        ...
                    </BSW-CALLED-ENTITY>
                </ENTITYS>
                ...
            </BSW-INTERNAL-BEHAVIOR>
25        </INTERNAL-BEHAVIORS>
        </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
30 </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

In the definition of a BswCalledEntity, there is a reference to an implemented entry. This information will reference a BswModuleEntry of the same BSW module.

In this BswModuleEntry, the tag CALL-TYPE shall be set to "REGULAR" (or "CALLBACK") and the tag EXECUTION--CONTEXT has to be set to "UNSPECIFIED", "HOOK" or "TASK".

Example of a BswModuleEntry:

```

40 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://autosar.org/schema/r4.0"
    xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
45 <AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
50            <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
            <ELEMENTS>
                <BSW-MODULE-ENTRY>
                    <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
                    ...
55                    <IS-REENTRANT>true(or false)</IS-REENTRANT>
                    <IS-SYNCHRONOUS>true(or false)</IS-SYNCHRONOUS>
                    <CALL-TYPE>REGULAR (or CALLBACK) </CALL-TYPE>
                    <EXECUTION-CONTEXT>UNSPECIFIED (or HOOK or TASK) </EXECUTION-CONTEXT>
                    <SW-SERVICE-IMPL-POLICY>STANDARD</SW-SERVICE-IMPL-POLICY>
                    ...
60                </BSW-MODULE-ENTRY>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
65 </AR-PACKAGE>
</AR-PACKAGES>

```

```
</AR-PACKAGES>
</AUTOSAR>
```

### 8.3.2.1.2 BswInterruptEntity

A BswInterruptEntity is designed for BSW Interrupt routines. Two categories of BswInterruptEntity can be defined:

- Category 1 (Cat1): Cat1 interrupt routines are not controlled by the OS and are only allowed to make a very limited selection of OS calls to enable and disable all interrupts. The BswInterruptEntity is implemented by the interrupt service routine, which is activated directly from HW (not via the OS)
- Category 2 (Cat2): Cat2 interrupt routines are controlled by the OS and they are allowed to make OS calls: The BswInterruptEntity is implemented by the interrupt handler, which is called from the OS.

Remark: It is not recommended to use Category 1 interrupts.

### Rule BSWMD\_Sched\_002: Definition of BswInterruptEntity

**Instruction** For each BSW Interrupt Entity, a **BswInterruptEntity** element shall be defined in a **BswInternalBehavior** and a **BswModuleEntry** shall be defined. The two model elements shall exist.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining BswInterruptEntity.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-DESCRIPTION>
              <SHORT-NAME>ModuleName</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <BSW-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                  ...
                  <ENTITYS>
                    <BSW-INTERRUPT-ENTITY>
                      <SHORT-NAME>BswInterruptEntityName</SHORT-NAME>
                      ...
                      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
                        <!-- Refer to the Snippet of a BswModuleEntry -->
                      </IMPLEMENTED-ENTRY-REF>
                      ...
                      <INTERRUPT-CATEGORY>CAT-2</INTERRUPT-CATEGORY>
                    </BSW-INTERRUPT-ENTITY>
                  </ENTITYS>
                  ...
                </BSW-INTERNAL-BEHAVIOR>
              </INTERNAL-BEHAVIORS>
            </BSW-MODULE-DESCRIPTION>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
```

</AUTOSAR>

In the definition of a BswInterruptEntity, there is a reference to an Implemented Entry. This information will reference a BswModuleEntry which shall be declared in the BSWMD ARXML file.

In this BswModuleEntry, the tag CALL-TYPE shall be set to "INTERRUPT" and the tag EXECUTION-CONTEXT shall be set to "INTERRUPT-CAT-2" (or "INTERRUPT-CAT-1").

Example of a BswModuleEntry:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
              <IS-REENTRANT>true(or false)</IS-REENTRANT>
              <IS-SYNCHRONOUS>true(or false)</IS-SYNCHRONOUS>
              <CALL-TYPE>INTERRUPT</CALL-TYPE>
              <EXECUTION-CONTEXT>INTERRUPT-CAT-2</EXECUTION-CONTEXT>
              <SW-SERVICE-IMPL-POLICY>STANDARD</SW-SERVICE-IMPL-POLICY>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

### 8.3.2.1.3 BswSchedulableEntity

A BswSchedulableEntity is designed for a scheduled function (so-called "main" function) which will be activated by a BswEvent and will be scheduled by SchM which is generated by RTE-Generator.

#### Rule BSWMD\_Sched\_003: Definition of BswSchedulableEntity

**Instruction** For each BswSchedulableEntity, a **BswSchedulableEntity** element shall be defined in a **BswInternal-Behavior** and a **BswModuleEntry** shall be defined. The two model elements shall exist.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining BswSchedulableEntity:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
```



```

05      <ELEMENTS>
        <BSW-MODULE-DESCRIPTION>
          <SHORT-NAME>ModuleName</SHORT-NAME>
          <INTERNAL-BEHAVIORS>
            <BSW-INTERNAL-BEHAVIOR>
10              <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                ...
            <ENTITYS>
              <BSW-SCHEDULABLE-ENTITY>
                <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
                ...
                <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
                  <!-- Refer to the Snippet of a BswModuleEntry -->
                </IMPLEMENTED-ENTRY-REF>
                ...
              </BSW-SCHEDULABLE-ENTITY>
            </ENTITYS>
            <EVENTS>
              <!-- An Event which will Activate the BswSchedulableEntity shall be defined -->
            </EVENTS>
            ...
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
  </AR-PACKAGE>
20 </AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

In the definition of a BswSchedulableEntity, there is a reference to an Implemented Entry. This information will reference a BswModuleEntry which shall be declared in the BSWMD ARXML file.

In this BswModuleEntry, the tag CALL-TYPE shall be set to "SCHEDULED" and the tag EXECUTION-CONTEXT shall be set to "TASK".

Example of a BswModuleEntry:

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
45 <AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
  </AR-PACKAGES>
  <AR-PACKAGE>
50    <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
    <ELEMENTS>
      <BSW-MODULE-ENTRY>
        <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
        ...
        <IS-REENTRANT>true(or false)</IS-REENTRANT>
        <IS-SYNCHRONOUS>true(or false)</IS-SYNCHRONOUS>
        <CALL-TYPE>SCHEDULED</CALL-TYPE>
        <EXECUTION-CONTEXT>TASK</EXECUTION-CONTEXT>
        <SW-SERVICE-IMPL-POLICY>STANDARD</SW-SERVICE-IMPL-POLICY>
        ...
      </BSW-MODULE-ENTRY>
    </ELEMENTS>
  </AR-PACKAGE>
  </AR-PACKAGES>
65 </AR-PACKAGES>
</AR-PACKAGE>

```

```
</AR-PACKAGES>
</AUTOSAR>
```

## Rule BSWMD\_Sched\_004: Definition of BswEvents

**Instruction** For each BswEvent, an **Events** element shall be defined in a **BswInternalBehavior**.

The concerned BswSchedulableEntity shall be referenced in a **StartsOnEventRef**.

The two model elements shall exist.

A BswSchedulableEntity can be only activated by the Scheduler according to the definition of a BswEvent.

Different kinds of events can be defined for a BswSchedulableEntity:

- ▶ BswBackgroundEvent
- ▶ BswExternalTriggerOccurredEvent
- ▶ BswInternalTriggerOccurredEvent
- ▶ BswModeSwitchEvent
- ▶ BswModeSwitchedAckEvent
- ▶ BswTimingEvent

For a cyclic activation of a BswSchedulableEntity, the BswTimingEvent shall be defined. Here a period shall be specified defined by a number of seconds.

As for any other kind of event, the BswSchedulableEntity to be started when the event occurs has to be defined.

Here you can find an example of definition of a timing event with a period of 100ms:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-DESCRIPTION>
              <SHORT-NAME>ModuleName</SHORT-NAME>
              <INTERNAL-BEHAVIORS>
                <BSW-INTERNAL-BEHAVIOR>
                  <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                  ...
                <ENTITYS>
                  <BSW-SCHEDULABLE-ENTITY>
                    <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
                    ...
                  <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
                    <!-- Refer to the Snippet of a BswModuleEntry -->

```

```

05         </IMPLEMENTED-ENTRY-REF>
        ...
        </BSW-SCHEDULABLE-ENTITY>
    </ENTITYS>
    <EVENTS>
        <BSW-TIMING-EVENT>
            <SHORT-NAME>BswTimingEventName</SHORT-NAME>
            ...
            <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
                <!-- Refer to the Snippet of a BswSchedulableEntity -->
            </STARTS-ON-EVENT-REF>
            ...
            <PERIOD>0.1</PERIOD>
        </BSW-TIMING-EVENT>
    </EVENTS>
    ...
    </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
25 </AR-PACKAGES>
    </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

When the Timing Event has been defined, it shall be assigned to a task (configuration files for RTE, OS and ECUC) such that the RTE generates header files containing the APIs and the code in C language.

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for the BswSchedulableEntity:

#### **FUNC(void, MODULENAME\_CODE) ModuleName\_BswModuleEntryName(void)**

```

40 /** @file      SchM_ModuleName.h
    *
    * @brief      Basic Software Scheduler Module Interlink header file
    *
    * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
    *
    * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
    */

    #ifndef SchM_ModuleName
    #define SchM_ModuleName

    #include "SchM_ModuleName_Type.h"
    #include "Rte_Intl.h"
    #ifdef __cplusplus
    extern "C" {
    #endif /* __cplusplus */

    #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
    #define RTE_PRV_ALL_API
    #endif

    /* API Mapping Macros */
    #ifndef RTE_CORE
    #endif /* RTE_CORE */

    /*****
    ***
    *** Schedulable Entity Prototypes
    ***

```

```

05 *****/

#define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
10 #define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

#ifdef __cplusplus
} /* extern C */
15 #endif /* __cplusplus */

#endif /* SchM_ModuleName */

```

The RTE generator will also generate the C code for the BSW Scheduler in form of Task bodies (resp: C-functions) to manage the cyclic scheduling of the BswSchedulable. In our case the file Task1.c will be generated and will contain the call to the BswSchedulableEntity.

### ModuleName\_BswModuleEntryName()

Example of generated scheduling code:

```

25 /** @file      Task1.c
 *
 * @brief      Task Task1 body
 *
 * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
30 * @note      Generated by ETAS GmbH  RTA-RTE v5.4.0  (R4.0 backend version: v7.1.27 (Build 31685))
 */

#define RTE_CORE

#include "Rte_Const.h"
#if !defined(RTE_VENDOR_MODE)
#pragma message "Compiling an individual task file but RTE_VENDOR_MODE
# not defined. Compiling a stale file?"
40 #endif /* !defined(RTE_VENDOR_MODE) */
#if defined(RTE_USE_TASK_HEADER)
#include "Task1.h"
#else /* defined(RTE_USE_TASK_HEADER) */
#include "Os.h"
45 #endif /* defined(RTE_USE_TASK_HEADER) */
#include "Rte.h"
#include "Rte_Int1.h"
#include "Rte_Type.h"
#include "Rte_DataHandleType.h"

50 #define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
#define MODULENAME_STOP_SEC_CODE
55 #include "MemMap.h" /*lint !e537 permit multiple inclusion */
#define RTE_START_SEC_TASKBODY
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, RTE_CODE)
Rte_task_Task1(void)
60 {
    {
        /* Box: BSWImpl0_BswImplementationName begin */
        ModuleName_BswModuleEntryName();
        /* Box: BSWImpl0_BswImplementationName end */
    }
65 } /* Task1 */

```

```
#define RTE_STOP_SEC_TASKBODY
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
```

### 8.3.2.2 BSW Modes

The BSW Modes can be defined and grouped in a Mode Declaration Group in order to have different behavior of BSW module according to the current mode and the switching from one mode to another. The modes have to be managed by a BSW module called Mode Manager which will be in charge of switching between the different defined modes. On the other side, some BSW modules can use these modes and react to some of them, they are called mode users. The usage of modes and the different events relating to modes are multiples.

A BswSchedulableEntity can be activated according to a Mode Switching (Exiting a defined Mode, Entering in a defined Mode or in case of Transition from a defined Mode to another defined Mode. For this BswModeSwitchEvent can be used.

A BswSchedulableEntity activated by any kind of event can be modeled in such a way that it can be disabled in any defined Mode(s).

For managing Modes, some specific APIs can be provided by the BSW Scheduler: SchM\_Switch and SchM\_Mode.

### Rule BSWMD\_Mode\_001: Definition of BSW Modes

**Instruction** Each BSW Mode shall be defined in a BSWMD ARXML using **ModeDeclaration** under **ModeDeclarations**.

The **InitialModeRef** shall also be added to have the complete **ModeDeclarationGroup**.

All these model elements shall exist.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining the Mode Declaration Group, it can be defined in a BSWMD ARXML file or in a separate file:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      ...
      <ELEMENTS>
        <MODE-DECLARATION-GROUP>
          <SHORT-NAME>ModeDeclarationGroupName</SHORT-NAME>
          <INITIAL-MODE-REF DEST="MODE-DECLARATION">
            <!-- Refer to the Snippet of a Mode from a ModeDeclarationGroup -->
          </INITIAL-MODE-REF>
          <MODE-DECLARATIONS>
            <MODE-DECLARATION>
              <SHORT-NAME>Mode1Name</SHORT-NAME>
              ...
            </MODE-DECLARATION>
            <MODE-DECLARATION>
              <SHORT-NAME>Mode2Name</SHORT-NAME>
              ...
            </MODE-DECLARATION>
            ...
            <MODE-DECLARATION>
              <SHORT-NAME>ModeXName</SHORT-NAME>
              ...
            </MODE-DECLARATION>
          </MODE-DECLARATIONS>
        </MODE-DECLARATION-GROUP>
      </ELEMENTS>
      ...
    </AR-PACKAGE>
```

```

05      </AR-PACKAGES>
    </AUTOSAR>

```

### 8.3.2.2.1 BSW Mode Management on the Mode Manager side

#### Rule BSWMD\_Mode\_002: Definition of BSW Mode Management on the Mode Manager side

**Instruction** For BSW Modes on the Mode Manager side the **ProvidedModeGroups** shall be defined in a BSWMD ARXML file.

The **ModeDeclarationGroupPrototype** shall be one element of the **ProvidedModeGroups**.

The **ManagedModeGroups** shall also be defined in the **BswModuleDescription** with a reference to the **ModeDeclarationGroupPrototype**.

All these model elements shall exist.

The BSW in charge of the ModeManagement on the Mode Manager side has to define the ProvidedModeGroups in the BswModuleDescription and the ManagedModeGroups in the BswSchedulableEntity (or BSWCalledEntity or BswInterruptEntity).

Example for defining BSW Mode Management on the Mode Management side (for a BswSchedulableEntity activated by a BswTimingEvent with a period of 100ms):

```

30 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://autosar.org/schema/r4.0"
    xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
35   <AR-PACKAGES>
     <AR-PACKAGE>
       <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
     <AR-PACKAGES>
       <AR-PACKAGE>
         <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
40         <ELEMENTS>
           <BSW-MODULE-ENTRY>
             <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
             ...
           </BSW-MODULE-ENTRY>
         </ELEMENTS>
       </AR-PACKAGE>
     <AR-PACKAGE>
       <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
       <ELEMENTS>
         <BSW-MODULE-DESCRIPTION>
50           <SHORT-NAME>ModuleName</SHORT-NAME>
           ...
         <PROVIDED-MODE-GROUPS>
           <MODE-DECLARATION-GROUP-PROTOTYPE>
             <SHORT-NAME>ModeDeclarationGroupPrototypeName</SHORT-NAME>
55           <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
             <!-- Refer to the Snippet of a ModeDeclarationGroup -->
             </TYPE-TREF>
           </MODE-DECLARATION-GROUP-PROTOTYPE>
         </PROVIDED-MODE-GROUPS>
         ...
       <INTERNAL-BEHAVIORS>
         <BSW-INTERNAL-BEHAVIOR>
           <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
           <ENTITYS>
65             <BSW-SCHEDULABLE-ENTITY>
               <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
               ...
             </BSW-SCHEDULABLE-ENTITY>
           </ENTITYS>
         </BSW-INTERNAL-BEHAVIOR>
       </INTERNAL-BEHAVIORS>
     </AR-PACKAGE>
   </AR-PACKAGES>

```

```

05      <MANAGED-MODE-GROUPS>
        <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
          <MODE-DECLARATION-GROUP-PROTOTYPE-REF
            DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
            <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
10          </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
          </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
        </MANAGED-MODE-GROUPS>
        ...
        <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        <!-- Refer to the Snippet of a BswModuleEntry -->
15        </IMPLEMENTED-ENTRY-REF>
        ...
        </BSW-SCHEDULABLE-ENTITY>
      </ENTITYS>
      <EVENTS>
        <BSW-TIMING-EVENT>
          <SHORT-NAME>BswTimingEventName</SHORT-NAME>
          ...
          <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
            <!-- Refer to the Snippet of a BswSchedulableEntity -->
25          </STARTS-ON-EVENT-REF>
          ...
          <PERIOD>0.1</PERIOD>
        </BSW-TIMING-EVENT>
      </EVENTS>
      ...
      </BSW-INTERNAL-BEHAVIOR>
    </INTERNAL-BEHAVIORS>
  </BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
30 </AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for the Switching Modes:

**SchM\_Switch\_ModuleName\_ModeDeclarationGroupName(data)**

And the API for the BswSchedulableEntity (activated by TimingEvent):

**FUNC(void, MODULENAME\_CODE) ModuleName\_BswModuleEntryName(void)**

```

45 /** @file      SchM_ModuleName.h
    *
    * @brief      Basic Software Scheduler Module Interlink header file
    *
    * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
    *
55  * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
    */

    #ifndef SchM_ModuleName
    #define SchM_ModuleName

60  #include "SchM_ModuleName_Type.h"
    #include "Rte_Intl.h"
    #ifdef __cplusplus
    extern "C" {
    #endif /* __cplusplus */

65  #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
    #define RTE_PRV_ALL_API

```

```

05  #endif

/* API Mapping Macros */
#ifndef RTE_CORE

#define RTE_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(VAR(Std_ReturnType, AUTOMATIC), RTE_CODE)
    SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName(VAR(uint8, AUTOMATIC) data);
#define RTE_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
15  #if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName( data )
    (SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName(data))
#endif
#endif /* RTE_CORE */

20  /*****
    ***
    *** Schedulable Entity Prototypes
    ***
    *****/

25  #define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
30  #define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

#ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

35  #endif /* SchM_ModuleName */

```

### 8.3.2.2.2 BSW Mode Management on the Mode User side

#### Rule BSWMD\_Mode\_003: Definition of BSW Mode Management on the Mode User side

**Instruction** For BSW Modes on the Mode User side the **RequiredModeGroups** shall be defined in the **BswModule-Description**.

The **ModeDeclarationGroupPrototype** shall be one element of the **RequiredModeGroups**.

A reference to the **ModeDeclarationGroup** shall also be defined in the **TypeTref** of the **ModeDeclarationGroup-Prototype**.

All these model elements shall exist.

The BSW in charge of the Mode Management on the Mode User side has to define the RequiredModeGroups in the BswModuleDescription.

Example for defining BSW Mode Management on the Mode User side (for a BswSchedulableEntity):

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://autosar.org/schema/r4.0"
    xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGE>
  </AR-PACKAGES>

```



```

05      <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-ENTRY>
          <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
          ...
        </BSW-MODULE-ENTRY>
      </ELEMENTS>
10    </AR-PACKAGE>
  </AR-PACKAGE>
  <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-DESCRIPTION>
      <SHORT-NAME>ModuleName</SHORT-NAME>
      ...
      <REQUIRED-MODE-GROUPS>
        <MODE-DECLARATION-GROUP-PROTOTYPE>
          <SHORT-NAME>ModeDeclarationGroupPrototypeName</SHORT-NAME>
          ...
          <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
            <!-- Refer to the Snippet of a ModeDeclarationGroup -->
          </TYPE-TREF>
          ...
          </MODE-DECLARATION-GROUP-PROTOTYPE>
        </REQUIRED-MODE-GROUPS>
        <INTERNAL-BEHAVIORS>
          <BSW-INTERNAL-BEHAVIOR>
            <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
            ...
            <ENTITYS>
              <BSW-SCHEDULABLE-ENTITY>
                <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
                ...
                <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
                  <!-- Refer to the Snippet of a BswModuleEntry -->
                </IMPLEMENTED-ENTRY-REF>
                ...
              </BSW-SCHEDULABLE-ENTITY>
            </ENTITYS>
            <EVENTS>
              ...
            </EVENTS>
            ...
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
      </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
  </AR-PACKAGE>
50  </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

## Rule BSWMD\_Mode\_004: Definition of BswModeSwitchEvent

**Instruction** For each BSW Mode Switch Event, a **BswModeSwitchEvent** element shall be defined in a **BswInternal-Behavior**.

The type of activation shall be defined in **Activation**, a reference to the **ModeDeclarationGroupPrototype** shall be defined in **Modelref**.

The concerned Mode(s) shall be referenced in a **TargetModeRef**.

All these model elements shall exist.

A BswSchedulable entity can be activated by a BswModeSwitchEvent according to the defined Modelref and the type of Activation, this event will be implemented by the BSW Scheduler.

For a BswModeSwitchEvent, the Activation has to be defined, 3 kinds of Activation can be defined:

- ▶ ON-ENTRY: to ModeX
- ▶ ON-EXIT: from ModeX
- ▶ ON-TRANSITION: from ModeX to ModeY

Example for defining the BswModeSwitchEvent in a BswInternalBehavior:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGE>
      <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-DESCRIPTION>
          <SHORT-NAME>ModuleName</SHORT-NAME>
          ...
        </BSW-MODULE-DESCRIPTION>
      </ELEMENTS>
      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
          ...
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
      <ENTITIES>
        <BSW-SCHEDULABLE-ENTITY>
          <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
          ...
        </BSW-SCHEDULABLE-ENTITY>
      </ENTITIES>
      <EVENTS>
        <BSW-MODE-SWITCH-EVENT>
          <SHORT-NAME>BswModeSwitchEventName</SHORT-NAME>
          ...
          <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
            <!-- Refer to the Snippet of a BswSchedulableEntity -->
          </STARTS-ON-EVENT-REF>
          ...
          <ACTIVATION>ON-ENTRY (or ON-EXIT or ON-TRANSITION)</ACTIVATION>
          <MODE-IREFS>
            <MODE-IREF>
              <CONTEXT-MODE-DECLARATION-GROUP-REF
                DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
              </CONTEXT-MODE-DECLARATION-GROUP-REF>
              <TARGET-MODE-REF DEST="MODE-DECLARATION">
                <!-- Refer to the Snippet of a Mode of a ModeDeclarationGroup -->
              </TARGET-MODE-REF>
            </MODE-IREF>
          </MODE-IREFS>
        </BSW-MODE-SWITCH-EVENT>
      </EVENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

In case of **ACTIVATION** set to **ON-TRANSITION** a second **MODE-IREF** is needed (then BswSchedulableEntity activation will take place when current mode is equal to the one stated in 1st MODE-IREF and next mode will be the one stated in the 2nd MODE-IREF):

```
<MODE-IREF>
  <CONTEXT-MODE-DECLARATION-GROUP-REF
    DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
    <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
  </CONTEXT-MODE-DECLARATION-GROUP-REF>
  <TARGET-MODE-REF DEST="MODE-DECLARATION">
    <!-- Refer to the Snippet of a Mode of a ModeDeclarationGroup -->
  </TARGET-MODE-REF>
</MODE-IREF>
```

```
</MODE-IREFS>
<!-- Here add the Snippet for a BswModeSwitchEvent -->
</BSW-MODE-SWITCH-EVENT>
</EVENTS>
```

```
...
</BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for the BswSchedulableEntity:

**FUNC(void, MODULENAME\_CODE) ModuleName\_BswModuleEntryName(void)**

```
/** @file      SchM_ModuleName.h
 *
 * @brief      Basic Software Scheduler Module Interlink header file
 *
 * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
 *
 * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
 */

#ifndef SchM_ModuleName
#define SchM_ModuleName

#include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

/* API Mapping Macros */
#ifndef RTE_CORE

#endif /* RTE_CORE */

/*****
 ***
 *** Schedulable Entity Prototypes
 ***
 *****/
```

```

05 *****/

#define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
10 #define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

#ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

15 #endif /* SchM_ModuleName */

```

For every kind of BswEvent, the generation of the event can be disabled in specific Modes of a ModeDeclarationGroup, for it the DisabledInModelref can be defined.

Example for defining the DisabledInModelref in a BswEvent:

```

25 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
      <AR-PACKAGE>
        <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        <ELEMENTS>
          <BSW-MODULE-DESCRIPTION>
            <SHORT-NAME>ModuleName</SHORT-NAME>
            ...
          <INTERNAL-BEHAVIORS>
            <BSW-INTERNAL-BEHAVIOR>
              <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
              ...
            <ENTITYS>
              <BSW-SCHEDULABLE-ENTITY>
                <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
                ...
              </BSW-SCHEDULABLE-ENTITY>
            </ENTITYS>
          <EVENTS>
            <BSW-.....-EVENT>
              <SHORT-NAME>BswEventName</SHORT-NAME>
              <DISABLED-IN-MODE-IREF>
                <CONTEXT-MODE-DECLARATION-GROUP-REF
30   DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                  <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
                </CONTEXT-MODE-DECLARATION-GROUP-REF>
                <TARGET-MODE-REF DEST="MODE-DECLARATION">
40   <!-- Refer to the Snippet of a Mode of a ModeDeclarationGroup -->
                </TARGET-MODE-REF>
              </DISABLED-IN-MODE-IREF>
            </BSW-.....-EVENT>
          </EVENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>

```

```

    </BSW-.....-EVENT>
  </EVENTS>
</BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

### 8.3.2.2.3 BSW Mode Access

#### Rule BSWMD\_Mode\_005: Definition of BSW Mode Access

**Instruction** If a BswSchedulableEntity needs to access to the current Mode the **AccessedModeGroups** shall be defined in the **BswSchedulableEntity** with a reference to the **ModeDeclarationGroupPrototype**.

If a BswSchedulableEntity needs to access to the current mode with the API SchM\_Mode, the AccessedModeGroup shall be defined.

The access to the current mode can be defined on the Mode Manager side and on the Mode User side.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining the AccessedModeGroups in a BswSchedulableEntity:

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGE>
      <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-DESCRIPTION>
          <SHORT-NAME>ModuleName</SHORT-NAME>
          ...
        <INTERNAL-BEHAVIORS>
          <BSW-INTERNAL-BEHAVIOR>
            <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
            ...
          </BSW-INTERNAL-BEHAVIOR>
        </INTERNAL-BEHAVIORS>
        <ENTITYS>
          <BSW-SCHEDULABLE-ENTITY>
            <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
            ...
            <ACCESSED-MODE-GROUPS>
              <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
                <MODE-DECLARATION-GROUP-PROTOTYPE-REF
                  DEST="MODE-DECLARATION-GROUP-PROTOTYPE">

```

```

    <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
    </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
    </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
    </ACCESSED-MODE-GROUPS>
    ...
    </BSW-SCHEDULABLE-ENTITY>
  </ENTITYS>
  ...
  </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for accessing the Current Mode:

### SchM\_Mode\_ModuleName\_ModeDeclarationGroupPrototypeName()

```

/** @file      SchM_ModuleName.h
 *
 * @brief      Basic Software Scheduler Module Interlink header file
 *
 * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
 *
 * @note      Generated by ETAS GmbH  RTA-RTE v5.4.0  (R4.0 backend version: v7.1.27  (Build 31685))
 */

#ifndef SchM_ModuleName
#define SchM_ModuleName

#include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

/* API Mapping Macros */
#ifndef RTE_CORE
#define RTE_START_SEC_VAR_8BIT
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
extern VAR(uint8, RTE_DATA) Rte_ModeCurrent_0;
#define RTE_STOP_SEC_VAR_8BIT
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
#if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BSWSE_BswModeChange_START_OnExit)
/* Inline read optimization; SchM_Mode_ModuleName_ModeDeclarationGroupPrototypeName to direct read */
#define SchM_Mode_ModuleName_ModeDeclarationGroupPrototypeName() ( Rte_ModeCurrent_0 )
#endif

#endif /* RTE_CORE */

/*****
 ***
 *** Schedulable Entity Prototypes
 ***
 *****/

```

```

05 #define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
#define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

10 #ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

15 #endif /* SchM_ModuleName */

```

### 8.3.2.3 BswTriggers

Triggers are used to pass pure trigger information without further data.

Two different kinds of BswTriggers can be defined:

- ▶ BswExternalTrigger: between different BSW modules
- ▶ BswInternalTrigger: inside a BSW module

This chapter will be detailed in the future. For the moment only simple examples are given.

#### 8.3.2.3.1 BswExternalTriggers on the Sender side

##### Rule BSWMD\_Trigger\_001: Definition of BswExternalTriggers on the Sender side

**Instruction** For each External Trigger, a **Trigger** element shall be defined in a **ReleasedTrigger** element of a **Bsw-ModuleDescription**.

For the BswSchedulableEntity, a reference to the ExternalTrigger shall be defined in **TriggerRefConditional** element in the **IssuedTrigger** element of the **BswSchedulableEntity**.

All these model elements shall exist.

Example for defining BswExternalTrigger on the Sender side:

```

45 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
          <ELEMENTS>
            <BSW-MODULE-ENTRY>
              <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
              ...
            </BSW-MODULE-ENTRY>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGE>
      <AR-PACKAGE>
        <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        <ELEMENTS>
          <BSW-MODULE-DESCRIPTION>
            <SHORT-NAME>ModuleName</SHORT-NAME>
            ...
          <RELEASED-TRIGGERS>

```

```

05      <TRIGGER>
        <SHORT-NAME>ExternalTriggerName</SHORT-NAME>
      </TRIGGER>
    </RELEASED-TRIGGERS>
  <INTERNAL-BEHAVIORS>
    <BSW-INTERNAL-BEHAVIOR>
10      <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
      ...
    <ENTITYS>
      <BSW-SCHEDULABLE-ENTITY>
15        <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
      ...
      <ISSUED-TRIGGERS>
        <TRIGGER-REF-CONDITIONAL>
          <TRIGGER-REF DEST="TRIGGER">
20            <!-- Refer to the Snippet of the ExternalTigger -->
          </TRIGGER-REF>
        </TRIGGER-REF-CONDITIONAL>
      </ISSUED-TRIGGERS>
    </BSW-SCHEDULABLE-ENTITY>
  </ENTITYS>
25  <EVENTS>
    ...
  </EVENTS>
    ...
  </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
</BSW-MODULE-DESCRIPTION>
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
35 </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for sending an External Trigger:

#### SchM\_Trigger\_ModuleName\_ExternalTriggerName()

```

45 /** @file      SchM_ModuleName.h
 *
 * @brief      Basic Software Scheduler Module Interlink header file
 *
 * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
 *
 * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
50 */

#ifndef SchM_ModuleName
#define SchM_ModuleName

55 #include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

60 #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

65 /* API Mapping Macros */
#ifndef RTE_CORE

```



```

05 #define RTE_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, RTE_CODE) SchM_Trigger_ModuleName_ExternalTriggerName(void);
#define RTE_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
10 #if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define SchM_Trigger_ModuleName_ExternalTriggerName() (SchM_Trigger_ModuleName_ExternalTriggerName())
#endif

#define RTE_CORE */
15 ...

```

### 8.3.2.3.2 BswExternalTriggers on the Receiver side

#### Rule BSWMD\_Trigger\_002: Definition of BswExternalTriggers on the Receiver side

**Instruction** For each External Trigger, a **Trigger** element shall be defined in a **RequiredTrigger** element of a **Bsw-ModuleDescription**.

On the receiver side, the BswSchedulableEntity will be activated by a BSWExternalTriggerOccuredEvent.

#### Rule BSWMD\_Trigger\_003: Definition of BSWExternalTriggerOccuredEvent

**Instruction** For each BSW External Trigger Occured Event, a **BswExternalTriggerOccuredEvent** element shall be defined in an **Events** element of a **BswInternalBehavior**.

The concerned BswSchedulableEntity shall be referenced in a **StartsOnEventRef**.

All these model elements shall exist.

Example for defining BswExternalTrigger on the Receiver side (for a BswSchedulableEntity activated by a BswExternal-TriggerOccuredEvent):

```

40 <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
45   <AR-PACKAGE>
     <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
     <AR-PACKAGES>
       <AR-PACKAGE>
         <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
50         <ELEMENTS>
           <BSW-MODULE-ENTRY>
             <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
             ...
           </BSW-MODULE-ENTRY>
         </ELEMENTS>
       </AR-PACKAGE>
       <AR-PACKAGE>
         <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
         <ELEMENTS>
60           <BSW-MODULE-DESCRIPTION>
             <SHORT-NAME>ModuleName</SHORT-NAME>
             ...
           <REQUIRED-TRIGGERS>
             <TRIGGER>
               <SHORT-NAME>ExternalTriggerName</SHORT-NAME>
65             </TRIGGER>
           </REQUIRED-TRIGGERS>

```

```

05      <INTERNAL-BEHAVIORS>
        <BSW-INTERNAL-BEHAVIOR>
          <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
          ...
          <ENTITYS>
            <BSW-SCHEDULABLE-ENTITY>
              <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
              ...
            </BSW-SCHEDULABLE-ENTITY>
          </ENTITYS>
          <EVENTS>
            <BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT>
              <SHORT-NAME>BswExternalTriggerOccurredEventName</SHORT-NAME>
              ...
              <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
                <!-- Refer to the Snippet of a BswSchedulableEntity -->
              </STARTS-ON-EVENT-REF>
              ...
              <TRIGGER-REF DEST="TRIGGER">
                <!-- Refer to the Snippet of the ExternalTigger -->
              </TRIGGER-REF>
            </BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT>
          </EVENTS>
          ...
        </BSW-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </BSW-MODULE-DESCRIPTION>
  </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
35 </AUTOSAR>

```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for the BswSchedulableEntity (activated by BswExternalTriggerOccuredEvent):

**FUNC(void, MODULENAME\_CODE) ModuleName\_BswModuleEntryName(void)**

```

40 /** @file      SchM_ModuleName.h
    *
    * @brief      Basic Software Scheduler Module Interlink header file
    *
    * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
    *
    * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
    */

50 #ifndef SchM_ModuleName
#define SchM_ModuleName

#include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

60 #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

/* API Mapping Macros */
65 #ifndef RTE_CORE

#endif /* RTE_CORE */

```

```

/*****
***
*** Schedulable Entity Prototypes
***
*****/

#define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
#define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

#ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

#endif /* SchM_ModuleName */

```

### 8.3.2.3.3 BswInternalTriggers

#### Rule BSWMD\_Trigger\_004: Definition of BswInternalTriggers

**Instruction** For each Internal Trigger, a **BswInternalTriggeringPoint** element shall be defined in a **InternalTriggeringPoints** element of a **BswInternalBehavior**.

The BswSchedulableEntity will be activated by a BswInternalTrigger Occured Event.

#### Rule BSWMD\_Trigger\_005: Definition of BswInternalTriggerOccuredEvent

**Instruction** For each BSW Internal Trigger Occured Event, a **BswInternalTriggerOccuredEvent** element shall be defined in an **Events** element of a **BswInternalBehavior**.

For the BswSchedulableEntity to be activated by the BSW Internal Trigger Occured Event, a reference to Bsw Internal Triggering Point shall be defined in the **BswInternalTriggeringPointRefConditional** of a **ActivationPoint** element of a **BswSchedulableEntity**.

The concerned BswSchedulableEntity shall be referenced in a **StartsOnEventRef**.

All these model elements shall exist.

This chapter will be detailed in the future. For the moment only simple examples are given.

Example for defining BswInternalTrigger and a BswInternalTriggerOccuredEvent (for a BswSchedulableEntity):

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGE>
    <AR-PACKAGE>
      <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-ENTRY>
          <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
          . . .
        </BSW-MODULE-ENTRY>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>

```

```

05    </AR-PACKAGE>
    <AR-PACKAGE>
        <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        <ELEMENTS>
            <BSW-MODULE-DESCRIPTION>
                <SHORT-NAME>ModuleName</SHORT-NAME>
                <INTERNAL-BEHAVIORS>
                    <BSW-INTERNAL-BEHAVIOR>
                        <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                        <INTERNAL-TRIGGERING-POINTS>
                            <BSW-INTERNAL-TRIGGERING-POINT>
                                <SHORT-NAME>BswInternalTriggeringPointName</SHORT-NAME>
                                ...
                                </BSW-INTERNAL-TRIGGERING-POINT>
                            </INTERNAL-TRIGGERING-POINTS>
                        <ENTITYS>
                            <BSW-SCHEDULABLE-ENTITY>
                                <SHORT-NAME>BswSchedulableEntityName_Sender</SHORT-NAME>
                                ...
                                <ACTIVATION-POINTS>
                                    <BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
                                        <BSW-INTERNAL-TRIGGERING-POINT-REF
                                            DEST="BSW-INTERNAL-TRIGGERING-POINT">
                                                <!-- Here add the Snippet for a BswInternalTriggeringPoint -->
                                                </BSW-INTERNAL-TRIGGERING-POINT-REF>
                                        </BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
                                    </ACTIVATION-POINTS>
                                ...
                                </BSW-SCHEDULABLE-ENTITY>
                            <BSW-SCHEDULABLE-ENTITY>
                                <SHORT-NAME>BswSchedulableEntityName_Receiver</SHORT-NAME>
                                ...
                                </BSW-SCHEDULABLE-ENTITY>
                        </ENTITYS>
                    <EVENTS>
                        <BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
                            <SHORT-NAME>BswInternalTriggerOccurredEventName</SHORT-NAME>
                            ...
                            <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
                                <!-- Refer to the Snippet of the BswSchedulableEntity (Receiver)-->
                                </STARTS-ON-EVENT-REF>
                            ...
                            <EVENT-SOURCE-REF DEST="BSW-INTERNAL-TRIGGERING-POINT">
                                <!-- Here add the Snippet for the BswInternalTriggeringPoint -->
                                </EVENT-SOURCE-REF>
                            </BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
                        </EVENTS>
                    ...
                </BSW-INTERNAL-BEHAVIOR>
            </INTERNAL-BEHAVIORS>
        </BSW-MODULE-DESCRIPTION>
    </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
60 </AUTOSAR>

```

Example of generated header files for a BSW module: SchM\_ModuleName.h

This file contains the API for sending an Internal Trigger:

**SchM\_ActMainFunction\_ModuleName\_InternalTriggerName(void)**

And the API for the BswSchedulableEntity (activated by BswInternalTriggerOccuredEvent):

**FUNC(void, MODULENAME\_CODE) ModuleName\_BswModuleEntryName\_Receiver(void)**

```

/** @file      SchM_ModuleName.h
 *
 * @brief      Basic Software Scheduler Module Interlink header file
 *
 * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
 *
 * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
 */

#ifndef SchM_ModuleName
#define SchM_ModuleName

#include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName_Sender) &&
!defined(RTE_RUNNABLEAPI_BswSchedulableEntityName_Receiver)
#define RTE_PRV_ALL_API
#endif

/* API Mapping Macros */
#ifndef RTE_CORE

#define RTE_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, RTE_CODE) SchM_ActMainFunction_ModuleName_InternalTriggerName(void);
#define RTE_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
#if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BSWSE_TriglMainFunction1)
#define SchM_ActMainFunction_ModuleName_InternalTriggerName()
(SchM_ActMainFunction_ModuleName_InternalTriggerName())
#endif

#endif /* RTE_CORE */

/*****
 ***
 *** Schedulable Entity Prototypes
 ***
 *****/

#define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName_Sender(void);
#define MODULENAME_STOP_SEC_CODE
#define MODULENAME_START_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */
FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName_Receiver(void);
#define MODULENAME_STOP_SEC_CODE
#include "MemMap.h" /*lint !e537 permit multiple inclusion */

#ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

#endif /* SchM_ModuleName */

```

### 8.3.2.4 Mapping between BSW and a corresponding SWC

Since AUTOSAR 4.0, Mappings between SWC and BSW can be defined. Mapping containers on IB level are defined between BSWMD and related service SWC, EcuAbstraction SWC or Complex Device Driver SWC.

Different kinds of mappings can be modeled:

- ▶ **SwcBswRunnableMapping** for a BSW Module which has a Bsw Module Description and a Swc Component Description (in example a SWC which is a ServiceSwComponentType or a ComplexDeviceDriverSwComponentType):  
to map a RunnableEntity (from a SWC) to be identical to a BswSchedulableEntity (from a BSW)  
Means: Model that a Runnable Entity is equal to a BswSchedulableEntity
- ▶ **SynchronizedModeGroups**: to synchronize by the Scheduler a pair of SWC and BSW Mode Group Prototypes  
Means: Model that SWC and BSW Mode Groups are handled as a common Mode Group
- ▶ **SynchronizedTriggers**: to synchronize by the scheduler a pair of SWC and BswTriggers  
Means: Model that SWC and BswTriggers are handled as common Triggers

#### Rule BSWMD\_SWC\_001: Definition of SwcBswRunnableMapping

**Instruction** For each SwcBswRunnable Mapping, a **SwcBswRunnableMappings** element shall be defined, a reference to a BswSchedulableEntity shall be defined in **BswEntityRef** and a reference to the Runnable Entity shall be defined in a **SwcRunnableRef**.

A reference to a Bsw Internal Behavior shall be defined in **BswBehaviorRef**.

And a reference to the Swc Internal Behavior shall be defined in a **SwcRunnableRef**.

All these element shall be defined in a **SwcBswMapping**.

**BswModuleEntry** shall be defined. Mapping between BSW and SWC shall be defined with the **SwcBswMapping** element in an ARXML file according to AUTOSAR specifications.

The Runnable mappings shall be defined in a **SwcBswRunnableMappings**.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
      <ELEMENTS>
        <SWC-BSW-MAPPING>
          <SHORT-NAME>SwcBswMappingName</SHORT-NAME>
          ...
          <BSW-BEHAVIOR-REF>
            <!-- Refer to the Snippet of a BswInternalBehavior -->
          </BSW-BEHAVIOR-REF>
          <RUNNABLE-MAPPINGS>
            <SWC-BSW-RUNNABLE-MAPPINGS>
              <BSW-ENTITY-REF DEST="BSW-SCHEDULABLE-ENTITY">
                <!-- Refer to the Snippet of a BswSchedulableEntity -->
              </BSW-ENTITY-REF>
              <SWC-RUNNABLE-REF DEST="RUNNABLE-ENTITY">
                <!-- Refer to the Snippet of a RunnableEntity -->
              </SWC-RUNNABLE-REF>
            </SWC-BSW-RUNNABLE-MAPPINGS>
          </RUNNABLE-MAPPINGS>
        </SWC-BSW-MAPPING>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

```

    </SWC-BSW-RUNNABLE-MAPPINGS>
  </RUNNABLE-MAPPINGS>
  <SWC-BEHAVIOR-REF>
    <!-- Refer to the Snippet of a SwcInternalBehavior -->
  </SWC-BEHAVIOR-REF>
</SWC-BSW-MAPPING>
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

## Rule BSWMD\_SWC\_002: Referencing a SwcBswMapping in a BSWMD ARXML file

**Instruction** If a Runnable Mapping has been defined in a **SwcBswRunnableMappings** element of a **SwcBswMapping**, a reference to the Swc Bsw Mapping shall be added in a **SwcBswMappingRef** element of the **BswImplementations** of the concerned BSWMD.

The same kind of reference shall be added for the concerned SWC module.

When a SwcBswMapping has been defined, this Mapping shall be referenced in the Bsw Implementation of the concerned Module (and in the SwcImplementation).

When a SwcBswMapping has been defined, this Mapping shall be referenced in the BswImplementation of the concerned Module (and in the SwcImplementation).

Example for referencing a SwcBswMapping in a BSWMD ARXML file:

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGES>
    ...
    <AR-PACKAGE>
      <SHORT-NAME>BswImplementations</SHORT-NAME>
      <ELEMENTS>
        <BSW-IMPLEMENTATION>
          <SHORT-NAME>BswImplementationName</SHORT-NAME>
          ...
          <SWC-BSW-MAPPING-REF DEST="SWC-BSW-MAPPING">
            <!-- Refer to the Snippet of a SwcBswMapping -->
          </SWC-BSW-MAPPING-REF>
          ...
        </BSW-IMPLEMENTATION>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

## Rule BSWMD\_SWC\_003: Definition of SynchronizedModeGroups between SWC and BSW

**Instruction** For each Synchronized Mode Groups a **SwcBswSynchronizedModeGroupPrototype** element shall be defined,

For BSW, a reference to a Mode Declaration Group Prototype shall be defined in **BswModeGroupIref**.

For SWC, a reference to the provided port used by the SWC in a **ContextPPortRef** and a reference to the Mode Declaration Group Prototype shall be defined in **SwcModeGroupIref**.

All these model elements shall be defined in a **SynchronizedModeGroups** element of a **SwcBswMapping**.

Example for defining SynchronizedModeGroups in a SwcBswMapping, it can be defined in a BSWMD ARXML file or in a separate file:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
    </AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
      <ELEMENTS>
        <SWC-BSW-MAPPING>
          <SHORT-NAME>SwcBswMappingName</SHORT-NAME>
          ...
          <BSW-BEHAVIOR-REF>
            <!-- Refer to the Snippet of a BswInternalBehavior -->
          </BSW-BEHAVIOR-REF>
          <SYNCHRONIZED-MODE-GROUPS>
            <SWC-BSW-SYNCHRONIZED-MODE-GROUP-PROTOTYPE>
              <BSW-MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype (for BSW) -->
              </BSW-MODE-GROUP-REF>
              <SWC-MODE-GROUP-IREF>
                <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
                  <!-- Refer to the Snippet of a ProvidedPortPrototype (for SWC) -->
                </CONTEXT-P-PORT-REF>
                <TARGET-MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                  <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype (for SWC) -->
                </TARGET-MODE-GROUP-REF>
              </SWC-MODE-GROUP-IREF>
              ...
            </SWC-BSW-SYNCHRONIZED-MODE-GROUP-PROTOTYPE>
          </SYNCHRONIZED-MODE-GROUPS>
          <SWC-BEHAVIOR-REF>
            <!-- Refer to the Snippet of a SwcInternalBehavior -->
          </SWC-BEHAVIOR-REF>
        </SWC-BSW-MAPPING>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

## Rule BSWMD\_SWC\_004: Definition of SynchronizedTriggers between SWC and BSW

**Instruction** For each Synchronized Triggers a **SwcBswSynchronizedTrigger** element shall be defined, For BSW, a reference to a Bsw Trigger shall be defined in **BswTriggerRef** For SWC, a reference to the provided port used by the Swc in a **ContextPPortRef** and a reference to the trigger used by the SWC in a **TargetTriggerRef**.

All these model elements shall be defined in a **SynchronizedTriggers** element of a **SwcBswMapping**.

Example for defining SynchronizedTriggers in a SwcBswMapping, it can be defined in a BSWMD ARXML file or in a separate file:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://autosar.org/schema/r4.0"
  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
```



```

05 <AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
        <AR-PACKAGES>
            <AR-PACKAGE>
10                <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
                <ELEMENTS>
                    <SWC-BSW-MAPPING>
                        <SHORT-NAME>SwcBswMappingName</SHORT-NAME>
                        ...
                        <BSW-BEHAVIOR-REF>
15                            <!-- Refer to the Snippet of a BswInternalBehavior -->
                        </BSW-BEHAVIOR-REF>
                        <SYNCHRONIZED-TRIGGERS>
                            <SWC-BSW-SYNCHRONIZED-TRIGGER>
                                <BSW-TRIGGER-REF DEST="TRIGGER">
20                                    <!-- Refer to the Snippet of a (External) Trigger (for BSW) -->
                                </BSW-TRIGGER-REF>
                                <SWC-TRIGGER-IREF>
                                    <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
25                                        <!-- Refer to the Snippet of a ProvidedPortPrototype (for SWC) -->
                                    </CONTEXT-P-PORT-REF>
                                    <TARGET-TRIGGER-REF DEST="TRIGGER">
                                        <!-- Refer to the Snippet of a Trigger (for SWC) -->
                                    </TARGET-TRIGGER-REF>
                                </SWC-TRIGGER-IREF>
                                ...
                            </SWC-BSW-SYNCHRONIZED-TRIGGER>
                        </SYNCHRONIZED-TRIGGERS>
                        <SWC-BEHAVIOR-REF>
                            <!-- Refer to the Snippet of a SwcInternalBehavior -->
                        </SWC-BEHAVIOR-REF>
                    </SWC-BSW-MAPPING>
                </ELEMENTS>
            </AR-PACKAGE>
        </AR-PACKAGES>
    </AR-PACKAGE>
30 </AR-PACKAGES>
</AUTOSAR>

```

### 8.3.3 BSW Documentation

Will be defined in a later version of the BSW Coding Guidelines.

## 8.3.4 BSW Module API Description (BswModuleEntry)

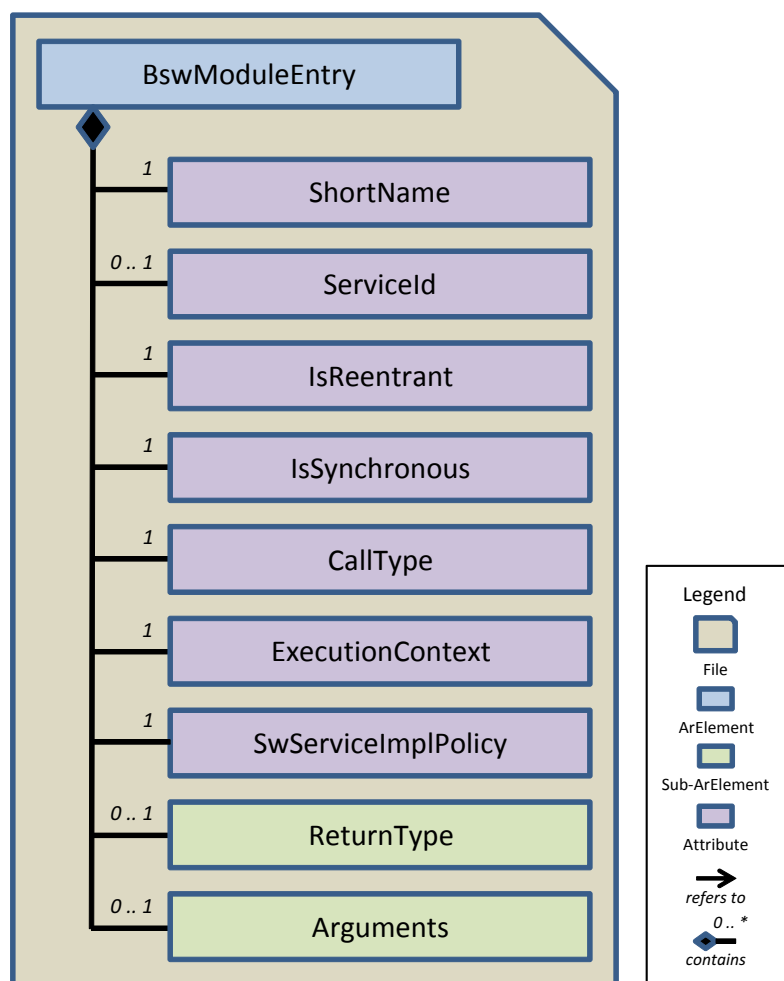
The class *BswModuleEntry* is used to model the signature of a C-function call. That means that with the *BswModuleEntry* the APIs of a BSW module can be described.

The *BswModuleEntry* is a top level ArElement and exists in parallel to the *BswModuleDescription* and the *BswImplementation*. A *BswModuleEntry* contains some common attributes which have to be specified for every API and are explained in chapter 8.3.4.1 p. 274 . How a return value and arguments of an API are specified is defined in chapter 8.3.4.2 p. 278 .

### 8.3.4.1 Common Attributes of a BswModuleEntry

This chapter gives an overview of all relevant common attributes of a *BswModuleEntry*. Rules are specified to handle the correct settings for the attributes. To have an overview of the common attributes *Figure 57* illustrates the details of a *BswModuleEntry*. Details for the definition of the return value and the arguments are defined in *Chapter 8.3.4.2 p. 278* .

Figure 57 Details of a BswModuleEntry



The following ARXML snippet shows an example for the *BswModuleEntry* of the API *Adc\_ReadGroup*.

```

...
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleEntrys</SHORT-NAME>      <!-- Rule [BSWMD_Entry_001] -->
    <ELEMENTS>
      <BSW-MODULE-ENTRY>
        <SHORT-NAME>Adc_ReadGroup</SHORT-NAME>    <!-- Rule [BSWMD_Entry_002] -->
        <SERVICE-ID>0x04</SERVICE-ID>           <!-- Rule [BSWMD_Entry_003] -->
      </BSW-MODULE-ENTRY>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
  
```

```

05      <IS-REENTRANT>true</IS-REENTRANT>          <!-- Rule [BSWMD_Entry_004] -->
      <IS-SYNCHRONOUS>true</IS-SYNCHRONOUS>        <!-- Rule [BSWMD_Entry_005] -->
      <CALL-TYPE>REGULAR</CALL-TYPE>               <!-- Rule [BSWMD_Entry_006] -->
      <EXECUTION-CONTEXT>                          <!-- Rule [BSWMD_Entry_007] -->
        UNSPECIFIED
      </EXECUTION-CONTEXT>
10      <SW-SERVICE-IMPL-POLICY>                    <!-- Rule [BSWMD_Entry_008] -->
        STANDARD
      </SW-SERVICE-IMPL-POLICY>
      <RETURN-TYPE>                                <!-- Refer to Chapter 8.3.4.2 p. 278 -->
        ...
15      </RETURN-TYPE>
      <ARGUMENTS>                                  <!-- Refer to Chapter 8.3.4.2 p. 278 -->
        ...
      </ARGUMENTS>
20      </BSW-MODULE-ENTRY>
      ...
      </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
    ...
25

```

**Hint** Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined in rule [\[BSWMD\\_Common\\_005\] p. 220](#)). Otherwise the tool cannot interpret the BSWMD file correctly.

## Rule BSWMD\_Entry\_001: ShortName of the ArPackage Kind Element BswModuleEntry

**Instruction** The ShortName of the *ArPackage* for the *BswModuleEntry* shall be set to "BswModuleEntrys".

This rule is derived from the rule [\[ARPac\\_44\] p. 301](#).

Example:

```

40      ...
      <AR-PACKAGE>
        <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
        <ELEMENTS>
          <BSW-MODULE-ENTRY>
45          ...

```

## Rule BSWMD\_Entry\_002: ShortName of the BswModuleEntry

**Instruction** The ShortName of the *BswModuleEntry* shall be set identical to the name of the C-function (but without a VendorId and a VendorApiInfix in the BSW module prefix).

The name of the C function and therefore the ShortName of the *BswModuleEntry* shall be conform to the naming convention (rule [\[CDGNaming\\_003\] p. 81](#)). The first part of a name of a C-function is the BSW module prefix. Within the ShortName no VendorId or VendorApiInfix shall be set. Possible existing VendorIds and VendorApiInfixes will be specified with special tags of the BswImplementation (see rule [\[BSWMD\\_Impl\\_003\] p. 229](#)) and are not set in the ShortName of the *BswModuleEntry*.

## Rule BSWMD\_Entry\_003: ServiceId of the BswModuleEntry

**Instruction**

**Scope:** Standardized Interfaces specified in an AUTOSAR SWS

In the tag <SERVICE-ID> the service identifier of the Standardized Interfaces specified in an AUTOSAR SWS shall be entered.

For non-standardized interfaces there is currently no handling specified to use a service identifier. Therefore the tag <SERVICE-ID> is just an optional tag and shall only be set for standardized interfaces which are specified within an AUTOSAR specification.

### Rule BSWMD\_Entry\_004: Reentrancy of a BswModuleEntry

**Instruction** The tag <IS-REENTRANT> shall be used to describe the reentrancy of the specified service.

Following two settings are possible: *true* or *false*

Explanation of the selectable settings:

Table 36 List of Settings for Definition of Reentrancy of a BswModuleEntry

Literal	Description
<i>true</i>	The service can be called again before the service has finished. The service is reentrant.
<i>false</i>	It is prohibited to call the service again before it has finished. The service is not reentrant.

The reentrancy of a service is specified within an AUTOSAR SWS. For non standardized services the information for reentrancy can still be determined and documented in this special tag.

### Rule BSWMD\_Entry\_005: Synchrony of a BswModuleEntry

**Instruction** The tag <IS-SYNCHRONOUS> shall be used to describe the synchrony of the specified service.

Following two settings are possible: *true* or *false*

Explanation of the selectable settings:

Table 37 List of Settings for Definition of Synchrony of a BswModuleEntry

Literal	Description
<i>true</i>	The service is completed when the call returns. The service is synchronous.
<i>false</i>	The service (on semantical level) may not be completed when the call returns. The service is asynchronous.

The synchrony of a service is specified within an AUTOSAR SWS. For non standardized services the information for synchrony can still be determined and documented in this special tag.

### Rule BSWMD\_Entry\_006: Call Type of the BswModuleEntry

**Instruction** With the tag <CALL-TYPE> the type of call associated with this specified service shall be set.

Following settings are possible: *CALLBACK*, *INTERRUPT*, *REGULAR* or *SCHEDULED*.

The call type denotes the mechanism by which the entry into the Bsw module shall be called. One of the following settings has to be chosen:

Table 38 List of Call Types of a BswModuleEntry

Literal	Description
<i>CALLBACK</i>	Callback (i.e. the caller specifies the signature)
<i>INTERRUPT</i>	Interrupt routine
<i>REGULAR</i>	Regular API call
<i>SCHEDULED</i>	Called by the scheduler

Consider the hint given in rule [\[BSWMD\\_Entry\\_007\] p. 277](#).

## Rule BSWMD\_Entry\_007: Execution Context of the BswModuleEntry

**Instruction** With the tag <EXECUTION-CONTEXT> the execution context required or guaranteed for the call associated with this specified service shall be set.

Following settings are possible: *HOOK*, *INTERRUPT-CAT-1*, *INTERRUPT-CAT-2*, *TASK* or *UNSPECIFIED*.

The execution context specifies which context is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this specified service. One of the following settings has to be chosen:

Table 39 List of Execution Contexts of a BswModuleEntry

Literal	Description
<i>HOOK</i>	Context of an OS "hook" routine
<i>INTERRUPT-CAT-1</i>	CAT1 interrupt context
<i>INTERRUPT-CAT-2</i>	CAT2 interrupt context
<i>TASK</i>	Task context
<i>UNSPECIFIED</i>	The execution context is not specified by the API

**Hint** Within a given *BswModuleEntry*, the following constraint holds for the attributes *CallType* and *ExecutionContext*:

- ▶ *CallType INTERRUPT* is not allowed together with *ExecutionContext TASK* or *HOOK*
- ▶ *CallType SCHEDULED* is not allowed together with *ExecutionContext INTERRUPT-CAT-1* or *INTERRUPT-CAT-2*
- ▶ All other combinations of these two attributes are allowed.

## Rule BSWMD\_Entry\_008: SwServiceImplPolicy of the BswModuleEntry

**Instruction** With the tag <SW-SERVICE-IMPL-POLICY> the implementation policy of the specified service shall be documented.

Following settings are possible: *INLINE*, *INLINE-CONDITIONAL*, *MACRO* or *STANDARD*.

The *SwServiceImplPolicy* denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. One of the following settings has to be chosen:

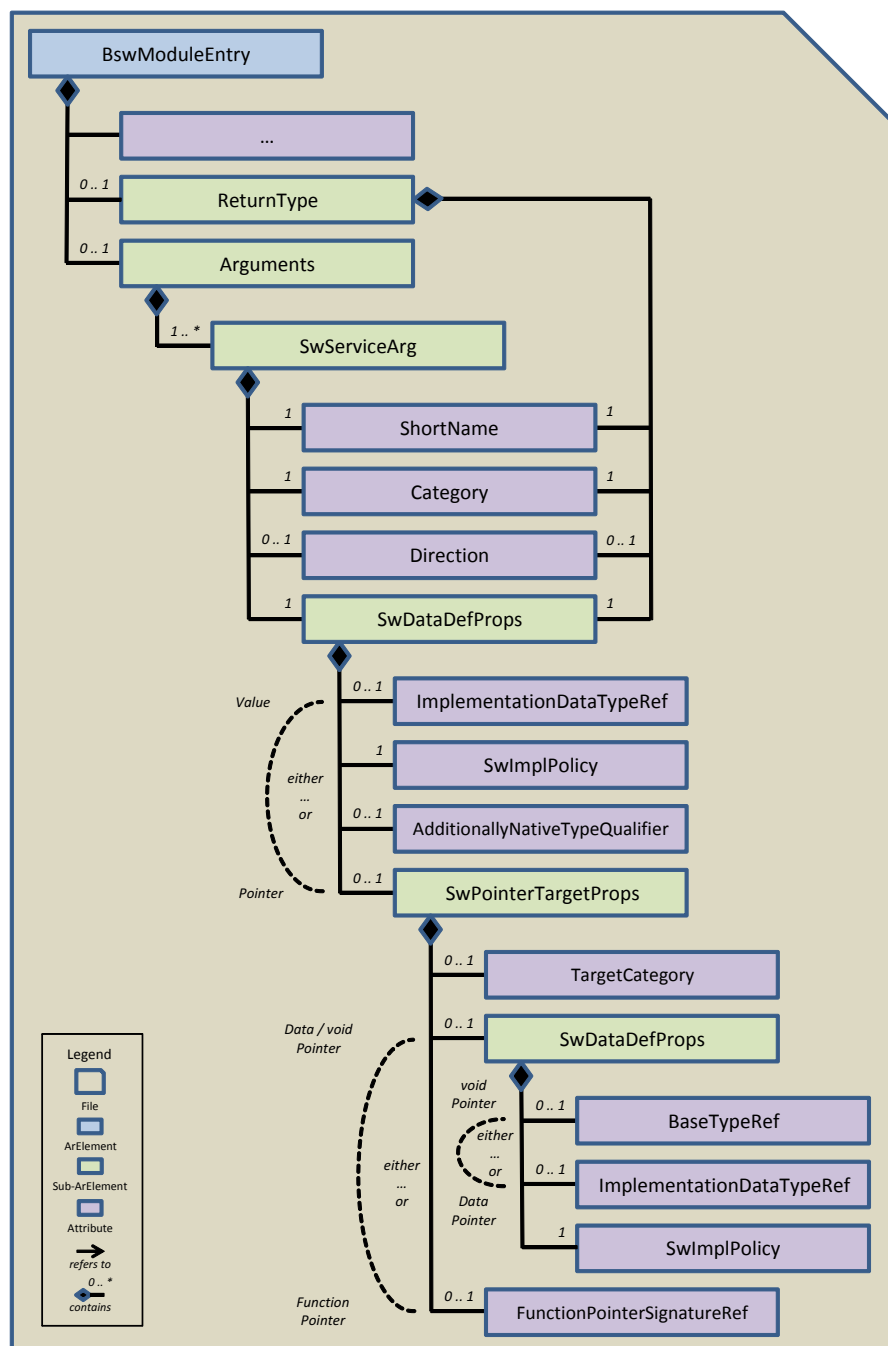
Table 40 List of SwServiceImplPolicy of a BswModuleEntry

Literal	Description
<i>INLINE</i>	Inline service definition. This service is defined using the macro <code>LOCAL_INLINE</code> . For more details refer to chapter 3.2.5 "Abstraction of Inline Functions" p. 55.
<i>INLINE-CONDITIONAL</i>	The service is implemented in a way that it either resolves to an inline function or to a standard function depending on conditions set at a later point in time (e.g. if the ECU configuration mechanism is used to inline a service depending to ECU configuration settings)
<i>MACRO</i>	Macro service definition
<i>STANDARD</i>	Standard service and default value, if nothing is defined

### 8.3.4.2 Definition of a Return Value and Arguments of a BswModuleEntry

This chapter handles how a return value and arguments have to be specified for a *BswModuleEntry*. The inner tags used to do that are similar for the definition of a return value and the definition of arguments. Only the embedded tags are different ('ReturnType' for a return type and 'Arguments' + 'SwServiceArg' for arguments). [Figure 58](#) gives an overview of the tag structure.

Figure 58 Details of a Return Value and Arguments of a BswModuleEntry



The dotted lines in the figure will show that a return value or an argument can only be a value or a pointer. And in case of a pointer it can be a data / void pointer or a function pointer. Depending on that kind of the return value or argument only the relevant attributes and elements have to be used to specify the return value or argument. For example, if an argument is only a value then the **ImplementationDataTypeRef** within the outer **SwDataDefProps** is applied but the element **SwPointerTypeProps** with all sub-elements and attributes is not used. The following examples and rules

will clarify all the details and specific issues for the definition of a return value or an argument. If the *BswModuleEntry* represents a macro the definition of a return value and arguments is easier as shown in the examples below. Refer for that use case to rule [\[BSWMD\\_EntryRetArg\\_012\] p. 287](#).

**Definition of return value, e.g. uint8:**

```
<RETURN-TYPE>
  <SHORT-NAME>ReturnValue</SHORT-NAME>                                <!-- Rule [BSWMD_EntryRetArg_001] -->
  <CATEGORY>TYPE_REFERENCE</CATEGORY>                                <!-- Rule [BSWMD_EntryRetArg_003] -->
  <DIRECTION>OUT</DIRECTION>                                         <!-- Rule [BSWMD_EntryRetArg_004] -->
  <SW-DATA-DEF-PROPS>                                                <!-- Rule [BSWMD_EntryRetArg_005] -->
    <SW-DATA-DEF-PROPS-VARIANTS>                                     <!--
      <SW-DATA-DEF-PROPS-CONDITIONAL>                                :
        <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
          /AUTOSAR_Platform/ImplementationDataTypes/uint8
        </IMPLEMENTATION-DATA-TYPE-REF>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>                   <!-- Rule [BSWMD_EntryRetArg_009] -->
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>                                     <!--
  </SW-DATA-DEF-PROPS>                                              :
  </SW-DATA-DEF-PROPS>                                              <!-- Rule [BSWMD_EntryRetArg_005] -->
</RETURN-TYPE>
```

**Hint** The ImplementationDataTypeRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementationDataTypeRef is written in three lines only for clearness.

There will be no more examples for return values because the examples for the different kinds of arguments are quite similar.

**Definition of a value as argument, e.g. (volatile uint8 Arg1\_u8):**

```
<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>Arg1_u8</SHORT-NAME>                                <!-- Rule [BSWMD_EntryRetArg_002] -->
    <CATEGORY>TYPE_REFERENCE</CATEGORY>                                <!-- Rule [BSWMD_EntryRetArg_003] -->
    <DIRECTION>IN</DIRECTION>                                         <!-- Rule [BSWMD_EntryRetArg_004] -->
    <SW-DATA-DEF-PROPS>                                                <!-- Rule [BSWMD_EntryRetArg_005] -->
      <SW-DATA-DEF-PROPS-VARIANTS>                                     <!--
        <SW-DATA-DEF-PROPS-CONDITIONAL>                                :
          <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
            /AUTOSAR_Platform/ImplementationDataTypes/uint8
          </IMPLEMENTATION-DATA-TYPE-REF>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>                   <!-- Rule [BSWMD_EntryRetArg_009] -->
          <ADDITIONAL-NATIVE-TYPE-QUALIFIER>                           <!-- Rule [BSWMD_EntryRetArg_010] -->
            volatile
          </ADDITIONAL-NATIVE-TYPE-QUALIFIER>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>                                     <!--
    </SW-DATA-DEF-PROPS>                                              :
    </SW-DATA-DEF-PROPS>                                              <!-- Rule [BSWMD_EntryRetArg_005] -->
  </SW-SERVICE-ARG>
  ...
</ARGUMENTS>
```

**Hint** The ImplementationDataTypeRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementationDataTypeRef is written in three lines only for clearness.

**Definition of a data pointer as argument, e.g. (volatile const uint8\* ptrArg\_cpu8):**

```
<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>ptrArg_cpu8</SHORT-NAME>                                <!-- Rule [BSWMD_EntryRetArg_002] -->
    <CATEGORY>DATA_REFERENCE</CATEGORY>                                <!-- Rule [BSWMD_EntryRetArg_003] -->
```

```

05  <DIRECTION>INOUT</DIRECTION>                                <!-- Rule  [BSWMD_EntryRetArg_004]  -->
    <SW-DATA-DEF-PROPS>                                         <!-- Rule  [BSWMD_EntryRetArg_006]  -->
        <SW-DATA-DEF-PROPS-VARIANTS>                           <!--           :                -->
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-IMPL-POLICY>CONST</SW-IMPL-POLICY>        <!-- Rule  [BSWMD_EntryRetArg_009]  -->
                <ADDITIONAL-NATIVE-TYPE-QUALIFIER>            <!-- Rule  [BSWMD_EntryRetArg_010]  -->
                    volatile
                </ADDITIONAL-NATIVE-TYPE-QUALIFIER>
            <SW-POINTER-TARGET-PROPS>
                <TARGET-CATEGORY>VALUE</TARGET-CATEGORY>    <!-- Rule  [BSWMD_EntryRetArg_011]  -->
            <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                    <SW-DATA-DEF-PROPS-CONDITIONAL>
                        <IMPLEMENTATION-DATA-TYPE-REF  DEST="IMPLEMENTATION-DATA-TYPE">
                            /AUTOSAR_Platform/ImplementationDataTypes/uint8
                        </IMPLEMENTATION-DATA-TYPE-REF>
                    <SW-IMPL-POLICY>                            <!-- Rule  [BSWMD_EntryRetArg_009]  -->
                        STANDARD
                    </SW-IMPL-POLICY>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
    </SW-POINTER-TARGET-PROPS>
</SW-DATA-DEF-PROPS-CONDITIONAL>
</SW-DATA-DEF-PROPS-VARIANTS>                                <!--           :                -->
</SW-DATA-DEF-PROPS>                                         <!-- Rule  [BSWMD_EntryRetArg_006]  -->
</SW-SERVICE-ARG>
...
</ARGUMENTS>

```

**Hint** The ImplementationDataTypeRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementationDataTypeRef is written in three lines only for clearness.

**Definition of a void pointer as argument, e.g. (volatile const void\* ptrArg\_cpv):**

```

40  <ARGUMENTS>
    <SW-SERVICE-ARG>
        <SHORT-NAME>ptrArg_cpv</SHORT-NAME>                   <!-- Rule  [BSWMD_EntryRetArg_002]  -->
        <CATEGORY>DATA_REFERENCE</CATEGORY>                   <!-- Rule  [BSWMD_EntryRetArg_003]  -->
        <DIRECTION>INOUT</DIRECTION>                           <!-- Rule  [BSWMD_EntryRetArg_004]  -->
        <SW-DATA-DEF-PROPS>                                     <!-- Rule  [BSWMD_EntryRetArg_007]  -->
            <SW-DATA-DEF-PROPS-VARIANTS>                       <!--           :                -->
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-IMPL-POLICY>CONST</SW-IMPL-POLICY>    <!-- Rule  [BSWMD_EntryRetArg_009]  -->
                    <ADDITIONAL-NATIVE-TYPE-QUALIFIER>        <!-- Rule  [BSWMD_EntryRetArg_010]  -->
                        volatile
                    </ADDITIONAL-NATIVE-TYPE-QUALIFIER>
                <SW-POINTER-TARGET-PROPS>
                    <TARGET-CATEGORY>VALUE</TARGET-CATEGORY> <!-- Rule  [BSWMD_EntryRetArg_011]  -->
                <SW-DATA-DEF-PROPS>
                    <SW-DATA-DEF-PROPS-VARIANTS>
                        <SW-DATA-DEF-PROPS-CONDITIONAL>
                            <BASE-TYPE-REF  DEST="SW-BASE-TYPE">
                                /AUTOSAR_Platform/SwBaseTypes/void
                            </BASE-TYPE-REF>
                        <SW-IMPL-POLICY>                            <!-- Rule  [BSWMD_EntryRetArg_009]  -->
                            STANDARD
                        </SW-IMPL-POLICY>
                    </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
        </SW-POINTER-TARGET-PROPS>
    </SW-DATA-DEF-PROPS-CONDITIONAL>

```



```

    </SW-DATA-DEF-PROPS-VARIANTS>                                <!--           :           -->
    </SW-DATA-DEF-PROPS>                                         <!-- Rule   [BSWMD_EntryRetArg_007]  -->
    </SW-SERVICE-ARG>
    ...
</ARGUMENTS>

```

**Hint** The BaseTypeRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the BaseTypeRef is written in three lines only for clearness.

#### Definition of a function pointer as argument:

```

<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>ptrFktArg_pfct</SHORT-NAME>                    <!-- Rule   [BSWMD_EntryRetArg_002]  -->
    <CATEGORY>FUNCTION_REFERENCE</CATEGORY>                   <!-- Rule   [BSWMD_EntryRetArg_003]  -->
    <DIRECTION>INOUT</DIRECTION>                              <!-- Rule   [BSWMD_EntryRetArg_004]  -->
    <SW-DATA-DEF-PROPS>                                         <!-- Rule   [BSWMD_EntryRetArg_008]  -->
      <SW-DATA-DEF-PROPS-VARIANTS>                             <!--           :           -->
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>           <!-- Rule   [BSWMD_EntryRetArg_009]  -->
          <SW-POINTER-TARGET-PROPS>
            <FUNCTION-POINTER-SIGNATURE-REF DEST="BSW-MODULE-ENTRY">
              /AUTOSAR_Adc/BswModuleEntrys/Adc_ReadGroup
            </FUNCTION-POINTER-SIGNATURE-REF>
          </SW-POINTER-TARGET-PROPS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>                             <!--           :           -->
    </SW-DATA-DEF-PROPS>                                         <!-- Rule   [BSWMD_EntryRetArg_008]  -->
  </SW-SERVICE-ARG>
  ...
</ARGUMENTS>

```

**Hint** The FunctionPointerSignatureRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the FunctionPointerSignatureRef is written in three lines only for clearness.

**Hint** Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined in rule [\[BSWMD\\_Common\\_005\] p. 220](#)). Otherwise the tool cannot interpret the BSWMD file correctly.

### Rule BSWMD\_EntryRetArg\_001: ShortName of a Return Value of a BswModuleEntry

**Instruction** The ShortName of a return value of a *BswModuleEntry* shall be set to 'ReturnValue'.

For a return value the ShortName has no functional relevance but the tag <SHORT-NAME> is still a mandatory one. Therefore the dummy-name 'ReturnValue' shall be set.

### Rule BSWMD\_EntryRetArg\_002: ShortName of an Argument of a BswModuleEntry

**Instruction** The ShortName of an argument of a *BswModuleEntry* shall be set identical to the name of the argument which is used in the C code or specified in the AUTOSAR specification.

The name of an argument is defined conform to rule [\[CDGNaming\\_006\] p. 85](#). The same identifier has to be used as ShortName of the *BswModuleEntry*.

## Rule BSWMD\_EntryRetArg\_003: Category of a Return Value or an Argument of a BswModuleEntry

**Instruction** With the tag <CATEGORY> the kind of the return value or the argument shall be specified (value, data pointer, function pointer).

Following categories are possible: *TYPE\_REFERENCE*, *DATA\_REFERENCE* or *FUNCTION\_REFERENCE*.

Dependent on a specific category appropriate SwDataDefProps shall be specified.

The category is relevant to specify the type of the return value or the argument. One of the following settings has to be chosen:

Table 41 List of Categories for a Return Value or Argument of a BswModuleEntry

Literal	Description
<i>TYPE_REFERENCE</i>	The element is defined via reference to another data type (via SwDataDefProps.implementationDataType). This category represents a value.
<i>DATA_REFERENCE</i>	Contains an address of another data prototype (whose type is given via SwDataDefProps.swPointerTargetProps). This category represents a data pointer.
<i>FUNCTION_REFERENCE</i>	Contains an address of a function prototype (whose signature is given via SwDataDefProps.-functionPointerSignature). This category represents a function pointer.

How to specify proper SwDataDefProps is explained in rules [\[BSWMD\\_EntryRetArg\\_005\]](#) , [\[BSWMD\\_EntryRetArg\\_006\]](#) , [\[BSWMD\\_EntryRetArg\\_007\]](#) and [\[BSWMD\\_EntryRetArg\\_008\]](#) .

## Rule BSWMD\_EntryRetArg\_004: Direction of a Return Type or an Argument of a BswModuleEntry

**Instruction** With the tag <DIRECTION> the kind of the data flow of the return value or the argument of a *BswModuleEntry* may be specified. The definition of the direction is optional.

Following values are possible: *IN*, *INOUT* or *OUT*.

A return value can only have the direction 'OUT'.

The direction of an argument can differ corresponding to the kind of the argument. A value argument has always the direction 'IN' while a pointer can have the direction 'IN', 'INOUT' or 'OUT' depending on the usage of the pointer argument.

The direction allows to declare the data flow of the arguments of a *BswModuleEntry*. One of the following settings has to be chosen:

Table 42 List of Direction Values for a Return Value or Argument of a BswModuleEntry

Literal	Description
<i>IN</i>	<b>Return value:</b> Not used for return values. <b>Argument:</b> The argument is only passed to the callee. That means that the argument is a value which is passed to the API.
<i>INOUT</i>	<b>Return value:</b> Not used for return values. <b>Argument:</b> The argument is passed to the callee but also passed back from the callee to the caller. This data flow direction is only possible if the argument is a pointer.
<i>OUT</i>	<b>Return value:</b> The only possible direction for a return value <b>Argument:</b> The argument is passed from the callee to the caller. This data flow direction is only possible if the argument is a pointer. Here the argument represents an additional return value. An input value is not transferred via that argument.

## Rule BSWMD\_EntryRetArg\_005: Definition of a Value as Return Value or Argument of a BswModuleEntry

**Instruction** To specify a value as return value or argument of a *BswModuleEntry* following settings shall be made:

- ▶ A reference to an *ImplementationDataType* using the tag `<IMPLEMENTATION-DATA-TYPE-REF>` shall be set within the *SwDataDefProps* tag structure
- ▶ The destination type of the *ImplementationDataTypeRef* shall be set to 'IMPLEMENTATION-DATA-TYPE'
- ▶ A complete path to a centrally or locally defined *ImplementationDataType* shall be set
- ▶ The category of the return value or argument shall be set to 'TYPE-REFERENCE' (conform to rule [\[BSWMD\\_EntryRetArg\\_003\]](#) )

The reference to an *ImplementationDataType* has to be set as shown in the following example.

```
<RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>TYPE_REFERENCE</CATEGORY>           <!-- Category for a value           -->
...
<SW-DATA-DEF-PROPS>                             <!-- SwDataDefProps tag structure       -->
  <SW-DATA-DEF-PROPS-VARIANTS>                   <!-- containing two sub tag structure  -->
    <SW-DATA-DEF-PROPS-CONDITIONAL>              <!-- levels                          -->
      <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
        /AUTOSAR_Platform/ImplementationDataTypes/uint8  <!-- Referred Impl.DataType  -->
      <IMPLEMENTATION-DATA-TYPE-REF>
        ...
      </SW-DATA-DEF-PROPS-CONDITIONAL>           <!--                          :           -->
    </SW-DATA-DEF-PROPS-VARIANTS>                <!--                          :           -->
  </SW-DATA-DEF-PROPS>                          <!-- Closing the SwDataDefProps      -->
</RETURN-TYPE> or </SW-SERVICE-ARG>
```

In the example above a reference to a centrally defined *ImplementationDataType* was used. It is also possible to refer to a locally specified *ImplementationDataType*. This *ImplementationDataType* could also be a primitive one (simple value) or a complex one (array or structure). To be able to refer to such an *ImplementationDataType* a proper and complete definition of that *ImplementationDataType* shall exist (as described in ["Rule Set: Data Description" p. 179](#)) and the path has to be set correctly too.

The *ImplementationDataTypeRef* has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the *ImplementationDataTypeRef* is written in three lines only for clearness.

## Rule BSWMD\_EntryRetArg\_006: Definition of a Data Pointer as Return Value or Argument of a BswModuleEntry

**Instruction** To specify a data pointer as return value or argument of a *BswModuleEntry* following settings shall be made:

- ▶ A reference to an *ImplementationDataType* using the tag `<IMPLEMENTATION-DATA-TYPE-REF>` shall be set within an inner *SwDataDefProps* tag structure inside of *SwPointerTargetProps* which are part of an outer *SwDataDefProps* tag structure of the return value or argument definition
- ▶ The destination type of the *ImplementationDataTypeRef* shall be set to 'IMPLEMENTATION-DATA-TYPE'
- ▶ A complete path to a centrally or locally defined *ImplementationDataType* shall be set
- ▶ The category of the return value or argument shall be set to 'DATA-REFERENCE' (conform to rule [\[BSWMD\\_EntryRetArg\\_003\]](#) )

The reference to an *ImplementationDataType* has to be set as shown in the following example.

```
<RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>DATA_REFERENCE</CATEGORY>           <!-- Category for a data pointer           -->
...
```

```

05  <SW-DATA-DEF-PROPS>                                <!-- Outer SwDataDefProps tag structure -->
      <SW-DATA-DEF-PROPS-VARIANTS>                    <!-- for the return value or argument -->
      <SW-DATA-DEF-PROPS-CONDITIONAL>                <!--                                :                                -->
      ...
      <SW-POINTER-TARGET-PROPS>                      <!-- SwPointerTargetProps structure -->
      ...
      <SW-DATA-DEF-PROPS>                            <!-- Inner SwDataDefProps tag structure -->
          <SW-DATA-DEF-PROPS-VARIANTS>                <!-- for the pointer definition -->
          <SW-DATA-DEF-PROPS-CONDITIONAL>            <!--                                :                                -->
              <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                  /AUTOSAR_Platform/ImplementationDataTypes/uint8 <!-- Ref. Impl.DataType -->
              </IMPLEMENTATION-DATA-TYPE-REF>
          ...
          </SW-DATA-DEF-PROPS-CONDITIONAL>            <!--                                :                                -->
          </SW-DATA-DEF-PROPS-VARIANTS>                <!--                                :                                -->
          </SW-DATA-DEF-PROPS>                        <!-- Closing the inner SwDataDefProps -->
      </SW-POINTER-TARGET-PROPS>                    <!-- Closing the SwPointerTargetProps -->
      </SW-DATA-DEF-PROPS-CONDITIONAL>                <!--                                :                                -->
      </SW-DATA-DEF-PROPS-VARIANTS>                  <!--                                :                                -->
      </SW-DATA-DEF-PROPS>                            <!-- Closing the outer SwDataDefProps -->
</RETURN-TYPE> or </SW-SERVICE-ARG>

```

It looks a bit complex how a pointer has to be specified within a BSWMD file. But the structure is quite logical. The Software Data Definition Properties (*SwDataDefProps*) are always needed when attributes of an element shall be specified and references to *ImplementationDataTypes* has to be set. In the example above (and also in general) the outer *SwDataDefProps* specifying the properties of the return value or argument. Because the return value or the argument shall represent a data pointer the definition of that pointer is done using the special meta-class *SwPointerTargetProps*. And because the pointer shall refer to an *ImplementationDataType* another inner *SwDataDefProps* tag structure is aggregated to set the reference to the *ImplementationDataType* (and to set additional attributes of the pointer (which are specified in rule [\[BSWMD\\_EntryRetArg\\_009\]](#))).

To complete the understanding: If the return value or argument is not a pointer but a simple value than the *SwPointerTargetProps* are not specified but instead of them a reference to an *ImplementationDataType* is set in the outer *SwDataDefProps* which is the context of rule [\[BSWMD\\_EntryRetArg\\_005\]](#). That is also the reason why in [Figure 58 "Details of a Return Value and Arguments of a BswModuleEntry" p. 278](#) the 'either ... or' decision between a simple value and the pointers is marked.

In the example above a reference to a centrally defined *ImplementationDataType* was used. It is here also possible to refer to a locally defined primitive (simple value) or complex (array or structure) *ImplementationDataType*. To be able to refer to such an *ImplementationDataType* a proper and complete definition of that *ImplementationDataType* shall exist (as described in ["Rule Set: Data Description" p. 179](#)) and the path has to be set correctly too.

The *ImplementationDataTypeRef* has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the *ImplementationDataTypeRef* is written in three lines only for clearness.

## Rule BSWMD\_EntryRetArg\_007: Definition of a Void Pointer as Return Value or Argument of a BswModuleEntry

**Instruction** To specify a void pointer as return value or argument of a *BswModuleEntry* following settings shall be made:

- ▶ A reference to a *SwBaseType* using the tag `<BASE-TYPE-REF>` shall be set within an inner *SwDataDefProps* tag structure inside of *SwPointerTargetProps* which are part of an outer *SwDataDefProps* tag structure of the return value or argument definition
- ▶ The destination type of the *ImplementationDataTypeRef* shall be set to 'SW-BASE-TYPE'
- ▶ The path '/AUTOSAR\_Platform/SwBaseTypes/void' shall be set

- The category of the return value or argument shall be set to 'DATA-REFERENCE' (conform to rule [\[BSWMD\\_EntryRetArg\\_003\]](#) )

The reference to an *ImplementationDataType* has to be set as shown in the following example.

```
<RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>DATA_REFERENCE</CATEGORY>           <!-- Category for a data pointer      -->
...
<SW-DATA-DEF-PROPS>                             <!-- Outer SwDataDefProps tag structure -->
  <SW-DATA-DEF-PROPS-VARIANTS>                   <!-- for the return value or argument  -->
    <SW-DATA-DEF-PROPS-CONDITIONAL>             <!--                               :      -->
      ...
      <SW-POINTER-TARGET-PROPS>                 <!-- SwPointerTargetProps structure  -->
        ...
        <SW-DATA-DEF-PROPS>                     <!-- Inner SwDataDefProps tag structure -->
          <SW-DATA-DEF-PROPS-VARIANTS>           <!-- for the pointer definition      -->
            <SW-DATA-DEF-PROPS-CONDITIONAL>     <!--                               :      -->
              <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_Platform/SwBaseTypes/void <!-- Ref. to SwBaseType for void -->
              </BASE-TYPE-REF>
            ...
          </SW-DATA-DEF-PROPS-CONDITIONAL>       <!--                               :      -->
        </SW-DATA-DEF-PROPS-VARIANTS>           <!--                               :      -->
      </SW-DATA-DEF-PROPS>                     <!-- Closing the inner SwDataDefProps -->
    </SW-POINTER-TARGET-PROPS>                 <!-- Closing the SwPointerTargetProps -->
  </SW-DATA-DEF-PROPS-CONDITIONAL>             <!--                               :      -->
</SW-DATA-DEF-PROPS-VARIANTS>                 <!--                               :      -->
</SW-DATA-DEF-PROPS>                         <!-- Closing the outer SwDataDefProps -->
</RETURN-TYPE> or </SW-SERVICE-ARG>
```

For void there is no *ImplementationDataType* available because void is not a memory consuming element which can be implemented. But there is a *SwBaseType* specified within the central elements who provides a type for void. This is the only case that a reference to a *SwBaseType* is made to specify a return value or an argument.

The *BaseTypeRef* has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the *BaseTypeRef* is written in three lines only for clearness.

## Rule BSWMD\_EntryRetArg\_008: Definition of a Function Pointer as Return Value or Argument of a *BswModuleEntry*

**Instruction** To specify a function pointer as return value or argument of a *BswModuleEntry* following settings shall be made:

- A reference to a *BswModuleEntry* using the tag <FUNCTION-POINTER-SIGNATURE-REF> shall be set within *SwPointerTargetProps* which are part of the *SwDataDefProps* tag structure of the return value or argument definition
- The destination type of the *FunctionPointerSignatureRef* shall be set to 'BSW-MODULE-ENTRY'
- A complete path to a *BswModuleEntry* shall be set
- The category of the return value or argument shall be set to 'FUNCTION-REFERENCE' (conform to rule [\[BSWMD\\_EntryRetArg\\_003\]](#) )

The reference to a *BswModuleEntry* (which represents a function pointer) has to be set as shown in the following example.

```
<RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>FUNCTION_REFERENCE</CATEGORY>       <!-- Category for a function pointer  -->
...
<SW-DATA-DEF-PROPS>                             <!-- SwDataDefProps tag structure  -->
  <SW-DATA-DEF-PROPS-VARIANTS>                   <!-- containing two sub tag structure -->
    <SW-DATA-DEF-PROPS-CONDITIONAL>             <!-- levels                        -->
```

```

...
<SW-POINTER-TARGET-PROPS>                                <!-- SwPointerTargetProps structure    -->
...
<FUNCTION-POINTER-SIGNATURE-REF DEST="BSW-MODULE-ENTRY">
  /AUTOSAR_Adc/BswModuleEntrys/Adc_ReadGroup  <!-- Reference to a BswModuleEntry -->
</FUNCTION-POINTER-SIGNATURE-REF>
</SW-POINTER-TARGET-PROPS>                                <!-- Closing the SwPointerTargetProps    -->
</SW-DATA-DEF-PROPS-CONDITIONAL>                    <!-- : -->
</SW-DATA-DEF-PROPS-VARIANTS>                        <!-- : -->
</SW-DATA-DEF-PROPS>                                    <!-- Closing the SwDataDefProps    -->
</RETURN-TYPE> or </SW-SERVICE-ARG>

```

Compared to the definition of a data pointer the definition of a function pointer is quite shorter. The reason is that the reference to the *BswModuleEntry* is not done via *SwDataDefProps* but with the special reference tag `<FUNCTION-POINTER-SIGNATURE-REF>`.

The *FunctionPointerSignatureRef* has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the *FunctionPointerSignatureRef* is written in three lines only for clearness.

## Rule BSWMD\_EntryRetArg\_009: Definition of an Implementation Policy for Return Values and Arguments of a BswModuleEntry

**Instruction** Within the *SwDataDefProps* an implementation policy shall be specified using the tag `<SW-IMPL-POLICY>`.

Following values are possible: *STANDARD* or *CONST*.

The tag `<SW-IMPL-POLICY>` is mandatory on each level of *SwDataDefProps*.

The implementation policy 'CONST' is only relevant in context of data pointers. In all other cases (values, function pointers) the implementation policy shall be set to 'STANDARD'.

The following table gives the description of the implementation policy:

Table 43 List of Implementation Policy Values for a Return Value or Argument of a BswModuleEntry

Literal	Description
<i>STANDARD</i>	The element is non-constant
<i>CONST</i>	The element is constant

The [Table 44](#) gives an overview of the four cases of a data pointer definition. Only in this context the implementation policy 'CONST' is relevant.

Table 44 Overview of Four Cases of Data Pointer Definition

Case	Example
Variable pointer to variable data	<code>uint8 * ptrArg_pu8</code>
Constant pointer to variable data	<code>uint8 * const ptrArg_cpu8</code>
Variable pointer to constant data	<code>uint8 const * ptrArg_pcu8</code> or <code>const uint8 * ptrArg_pcu8</code>
Constant pointer to constant data	<code>uint8 const * const ptrArg_cpcu8</code> or <code>const uint8 * const ptrArg_cpcu8</code>

The following example will show on which position the *SwImplPolicy* has to be set for the different positions of consts.

```

<RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>DATA_REFERENCE</CATEGORY>                <!-- Category for a data pointer    -->
...
<SW-DATA-DEF-PROPS>                                <!-- Outer SwDataDefProps tag structure -->
                                                    <!-- for the return value or argument -->
...
<SW-IMPL-POLICY>CONST</SW-IMPL-POLICY>            <!-- Position for a constant pointer    -->

```



```

05      ...
      <SW-POINTER-TARGET-PROPS>                                <!-- SwPointerTargetProps structure    -->
      ...
      <SW-DATA-DEF-PROPS>                                       <!-- Inner SwDataDefProps tag structure -->
                                                                <!-- for the pointer definition    -->
10      ...
      <SW-IMPL-POLICY>CONST</SW-IMPL-POLICY>                   <!-- Position for constant data      -->
      ...
      </SW-DATA-DEF-PROPS>                                       <!-- Closing the inner SwDataDefProps -->
      </SW-POINTER-TARGET-PROPS>                               <!-- Closing the SwPointerTargetProps -->
      </SW-DATA-DEF-PROPS>                                       <!-- Closing the outer SwDataDefProps -->
15 </RETURN-TYPE> or </SW-SERVICE-ARG>

```

Even if the element is not constant the SwImplPolicy has to be set with the value 'STANDARD' because the tag is a mandatory one.

## Rule BSWMD\_EntryRetArg\_010: Definition of an Additional Native Type Qualifier for Return Values and Arguments of a BswModuleEntry

**Instruction** Within the *SwDataDefProps* an additional type qualifier (e.g. "volatile") may be specified using the tag <ADDITIONAL-NATIVE-TYPE-QUALIFIER>.

The tag <ADDITIONAL-NATIVE-TYPE-QUALIFIER> is optional and shall only be used if an additional type qualifier has to be specified.

## Rule BSWMD\_EntryRetArg\_011: Definition of a Target Category for Data Pointers

### Instruction

**Scope:** Only relevant for the definition of data pointers

Within the inner *SwDataDefProps* the category of the target shall be specified using the tag <TARGET-CATEGORY>. The category of the referenced data shall be set.

Following values are possible: *VALUE*, *ARRAY* or *STRUCTURE*

Because the data pointer refers to an *ImplementationDataType* only the valid categories of an *ImplementationDataType* can be set here as target category. The specified data pointer can be a pointer to a value, an array or a structure. More details about the definition of *ImplementationDataTypes* can be found in [Chapter 7.1.2 "ImplementationDataType" p. 187](#).

## Rule BSWMD\_EntryRetArg\_012: Definition of Return Values and Arguments if the BswModuleEntry is a Macro

**Instruction** If the *BswModuleEntry* represents a macro the return type and the arguments shall be defined without *SwDataDefProps*. Only the common arguments are relevant.

Macros do not have significant data types for their return type and arguments. Therefore no reference to an *ImplementationDataType* can be specified. And because no *ImplementationDataType* can be specified also no *SwDataDefProps* has to be set. Only the common arguments ShortName, Category and Direction shall be set within the definition of a return type and the argument(s). The following example will give a short overview:

```

<BSW-MODULE-ENTRY>
  ...
  <SW-SERVICE-IMPL-POLICY>MACRO</SW-SERVICE-IMPL-POLICY>
  ...
  <RETURN-TYPE>
    <SHORT-NAME>ReturnValue</SHORT-NAME>
    <CATEGORY>TYPE_REFERENCE</CATEGORY>
    <DIRECTION>OUT</DIRECTION>
  </RETURN-TYPE>

```

**<ARGUMENTS>**

&lt;SW-SERVICE-ARG&gt;

&lt;SHORT-NAME&gt;Arg1&lt;/SHORT-NAME&gt;

&lt;CATEGORY&gt;TYPE\_REFERENCE&lt;/CATEGORY&gt;

&lt;DIRECTION&gt;IN&lt;/DIRECTION&gt;

&lt;/SW-SERVICE-ARG&gt;

...

**</ARGUMENTS>**

&lt;/BSW-MODULE-ENTRY&gt;

**Rule BSWMD\_EntryRetArg\_013: BswModuleEntry with no Return Value**

**Instruction** In case of an empty return type ("void" in C) no return value shall be specified within the *BswModuleEntry*.  
The <RETURN-TYPE> tag structure shall not be set.

To specify the empty return type as a void-type (to make a reference to a SwBaseType "void") is not applied.

**Rule BSWMD\_EntryRetArg\_014: BswModuleEntry with no Argument**

**Instruction** In case of an empty argument list ("void" in C) no arguments shall be specified within the *BswModuleEntry*.  
The <ARGUMENT> tag structure shall not be set.

To specify the empty argument list as a void-type (to make a reference to a SwBaseType "void") is not applied.



## 8.3.5 BSW Service Needs

Will be defined in a later version of the BSW Coding Guidelines.

## 8.3.6 BSW Exclusive Areas

It is not allowed to describe BSW Exclusive Areas within BSWMD files because there are some known problems in generation of SchM by the *RTE*. Use the workaround which is described in rule [\[BSW\\_Sched\\_001\]](#).

## 9 Rule Set: AUTOSAR Package Structure

### 9.1 Introduction to Use of *ARPackage*

The most important objects in an *.arxml* file are the *ARElements*. E.g. *CompuMethods*, *Units*, *SwComponentTypes* are all subclasses of *ARElement*.

All *ARElements* have to be put into an *ARPackage*.

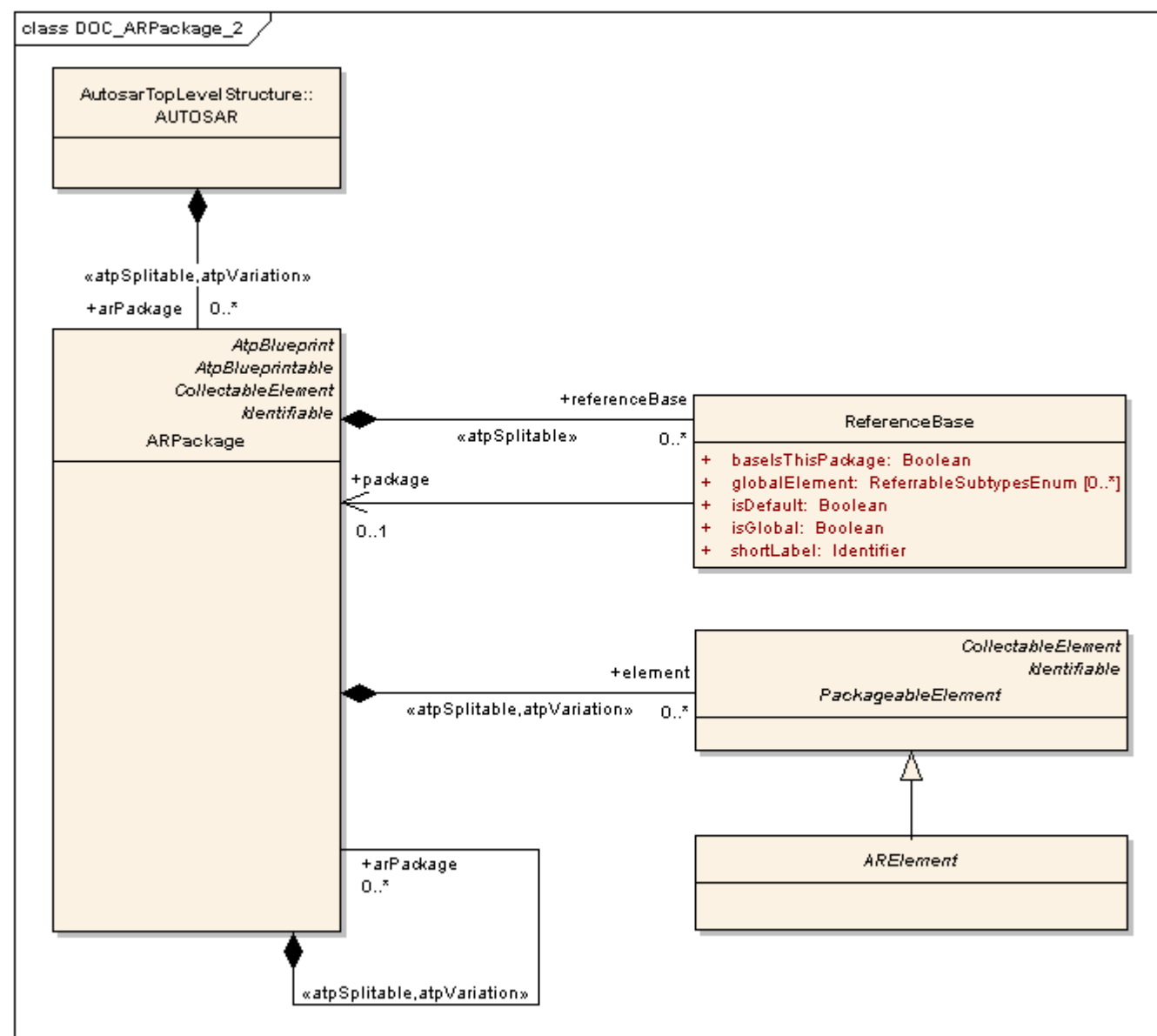
*ARPackages* define name spaces on *arxml* level. All the included *ARElements* shall have different *ShortNames*. E.g. if a *Unit* and a *CompuMethod* should have the same name you have to put them into different *ARPackages*.

*ARPackages* can be nested.

*ARPackages* can be split over several files.

The following figures show the *ARPackage* and its related objects for AUTOSAR 4.0.3.

Figure 59 AR 4.0's diagram "DOC\_ARPackage"



For more information see Chapter 4.2 (Packages in AUTOSAR) of [\[TPS\\_GST\]](#).

Here you can see an example how an *ARPackage* structure in a real *.arxml* file looks like.

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://autosar.org/schema/r4.0"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-0-3.xsd"
xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <ADMIN-DATA>
    <USED-LANGUAGES><L-10 L="EN" xml:space="preserve">EN</L-10></USED-LANGUAGES>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR_CanIf</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>DataConstraints</SHORT-NAME>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>CompuMethods</SHORT-NAME>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>Units</SHORT-NAME>
        </AR-PACKAGE>
        <AR-PACKAGE>
          <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>

```

### 9.1.1 Splitting *ARPackage* into different files

In figure 59 you see the stereotype «*atpSplitable*». This means that an *ARPackage* can be split into different files. Or, in other words, the *ARElements* of an *ARPackage* may be spread over multiple files.

Some tools still do not reflect this completely.

With status of Sept 2013 there is still no committed planning when a complete support of «*atpSplitable*» concept will be available.

Some tools (e.g. ETAS's *ISOLAR-A*) at least provide a view of merged *ARPackages* and also some support of creating split files.

As long as no sufficient tooling is available there is no common rule when to use «*atpSplitable*» and when not. You have to test tool behavior first.

### 9.1.2 Referencing *ARElements*

The reference to an *ARElement* is done by an XML-Tag, e.g. **<UNIT-REF>**. It contains an attribute **[DEST]** describing the kind of *ARElement* it is pointing to. The structure of the *ARPackages* where the referenced *ARElement* is contained has to be given in a "/" separated package path.

Example for a reference to a unit:

```
<UNIT-REF DEST="UNIT">/RB/PT/Common/CentralElements/Units/Rpm</UNIT-REF>
```

If you use a specific AUTOSAR authoring tool it typically will provide functionality for entering a reference and you don't have to type this deep path manually.

### 9.1.3 ReferenceBases

AUTOSAR 4.x defines a concept to ease the usage of references to *ARElements*. Instead of providing a deep *ARPackage* path at each reference you can define a reference base representing an *ARPackage* path. In a reference you can then use a path relative to the reference base.

If the reference base changes you have to implement this change on one single position. Till AUTOSAR release 4.0.2 the referenceBase itself is not <<splitable>>, the arxml file has to be changed even if reference bases are used. Starting with AUTOSAR release 4.0.3 the referenceBase itself is now <<splitable>>.

Example for using a *ReferenceBase*:

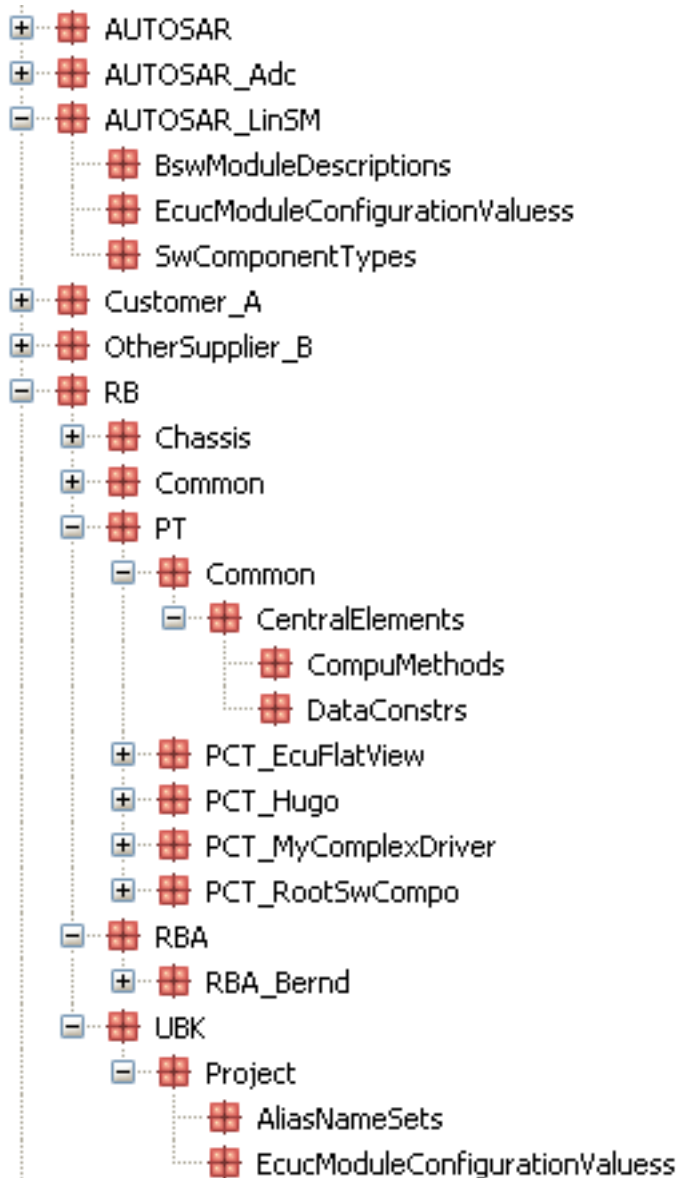
```
<UNIT-REF BASE="PT_CEL_Units" DEST="UNIT">Rpm</UNIT-REF>
```

## 9.2 Main Structure of *ARPackages*

### Overview

This picture gives an overview on the nesting of *ARPackages*. Following chapters will describe the detailed rules of the topics.

Figure 60 Overview on ARPackage structure



This shows the *ARPackage* structure of an ECU.

Of course these packages are spread over a lot of files.

## Rule ARPac\_10: Main Pattern for Package Structure

### Instruction

**Scope:** RB deliverables, except AUTOSAR specified software (e.g. BSW modules, Application Interfaces (AISpecification))

The following structure should be followed:

```

/RB
/{domain}
[/{subdomain}]*
/{block}
/{kind}[_Blueprint|_Example]
  
```

where

- ▶ RB is the top level *ARPackage* for RB, see [\[ARPac\\_11\]](#)
  - ▶ domain is an ECU domain, such as PT (power train), RBA (RB's first implementation of AUTOSAR BSW: CUBAS), see [\[ARPac\\_12\]](#)
  - ▶ subdomain gives the possibility to create a structure of sub domains
  - ▶ block is the name of a component type or a BSW module or "CentralElements" or similar
  - ▶ kind denotes the kind of the ARElement(s) which are contained, see [\[ARPac\\_14\]](#)
- For an AUTOSAR specified BSW module there is another package structure to be used. See rule [\[ARPac\\_41\]](#) .
- For ECU Configuration values a special *ARPackage*-Structure is to be used ("/RB/UBK/Project/..."). See rule [\[ARPac\\_46\]](#) and rule [\[ARPac\\_52\]](#) .

For more information see [\[TPS\\_GST\]](#) for *TPS\_GST\_00083* and *TPS\_GST\_00017*.

## Rule ARPac\_11: Top Level *ARPackage*

### Instruction

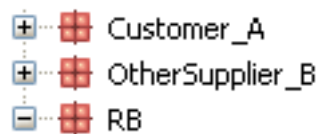
**Scope:** RB deliverables, except AUTOSAR specified software (e.g. BSW modules)

There shall be exactly one top level *ARPackage* in an .arxml file.

The top level package shall be named "RB".

Example:

Figure 61



**Note:** this figure shows the structure of *ARPackages* but they will not appear in the same file.

This avoids (or at least reduces) name clashes on AUTOSAR model level when arxml files of different suppliers or customers are combined in one system. AUTOSAR itself uses the short name "AUTOSAR" for their provided arxml files.

Please note that the rule to have exactly one top level *ARPackage* in an .arxml file is valid for RB. But files coming from outside RB may have more than one top level *ARPackage*.

## Rule ARPac\_12: Domain oriented *ARPackages*

### Instruction

**Scope:** RB deliverables, except AUTOSAR specified BSW modules

The package name on second level should be the name of the ECU-domain, as defined in [Table 45](#) .

No names derived from organisations should be used (e.g. DGS, CC-DA).

The name "AUTOSAR" and all names beginning with "AUTOSAR" are reserved.

If required subdomains may be defined.

This avoids name clashes within the different ECU domains in UBK.

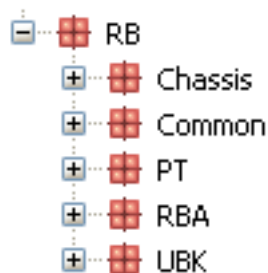
A clearing for these names should be established. Since this is not done today the following possible values are defined:

Table 45 Second Level *ARPackage*

short name	used for
PT	Power Train
--tbd--	For other ECU domains a short name has still to be defined. E.g. Chassis
UBK	an <i>ARPackage</i> for not domain specific objects
RBA	RB's first implementation of AUTOSAR BSW: CUBAS
Common	for objects which are used in all domains / sub domains

Example:

Figure 62



**Hint** Central Elements Guideline Artifact ( [\[A\\_CEL\]](#) ) defines detailed rules how to use *ARPackages* for central elements. See rules *CEL\_011* and *CEL\_020*.

Rule *ARPac\_13*: Removed

Rule *ARPac\_14*: Names for *ARElement*-kind based *ARPackages*

**Instruction** **Scope:** AR4.x

Whenever (sub-) *ARPackages* are created for the different kinds of *ARElements*, *ARPackage*'s *ShortName* shall be derived from the kind, concatenated by a plural "s". In most cases this is the name of the direct subclass of *ARElement* or *FibexElement*. Complete List is given in [Table 46 \[Recommended Packages\]](#)

If the *ARPackage* contains *blueprint elements* or *example elements* then "\_Blueprint" resp. "\_Example" has to be added to this name (e.g. "ApplicationDataTypes\_Blueprint").

Note: the usage of *examples* is currently not defined in our methods and shall not be used. Details will be defined in [\[A\\_Data\]](#) or other guidelines.

This behavior is also described in *TPS\_GST\_00049* and *TPS\_GST\_00085*.

Table 46 Recommended Packages

Element kind	<i>recommendedPackage</i>
AclObjectSet	AclObjectSets
AclOperation	AclOperations
AclPermission	AclPermissions
AclRole	AclRoles
AliasNameSet	AliasNameSets
ApplicationArrayDataType	ApplicationDataTypes
ApplicationPrimitiveDataType	ApplicationDataTypes
ApplicationRecordDataType	ApplicationDataTypes

Element kind	<i>recommendedPackage</i>
ApplicationSwComponentType	SwComponentTypes
BlueprintMappingSet	BlueprintMappingSets
BswImplementation	BswImplementations
BswModuleClientServerEntry	BswModuleEntrys
BswModuleDescription	BswModuleDescriptions
BswModuleEntry	BswModuleEntrys
BswModuleTiming	TimingExtensions
BuildActionManifest	BuildActionManifests
CalibrationParameterValueSet	CalibrationParameterValueSets
CanCluster	CommunicationClusters
CanFrame	Frames
CanTpConfig	TpConfigs
ClientServerInterface	PortInterfaces
Collection	Collections
ComplexDeviceDriverSwComponentType	SwComponentTypes
CompositionSwComponentType	SwComponentTypes
CompuMethod	CompuMethods
ConsistencyNeedsBlueprintSet	ConsistencyNeedsBlueprintSets
ConstantSpecification	ConstantSpecifications
ConstantSpecificationMappingSet	ConstantSpecificationMappingSets
CouplingElement	CouplingElements
DataConstr	DataConstrs
DataTypeMappingSet	DataTypeMappingSets
DcmIPdu	Pdus
Documentation	Documentations
EcuAbstractionSwComponentType	SwComponentTypes
EcuInstance	EcuInstances
EcuTiming	TimingExtensions
EcucDefinitionCollection	EcucDefinitionCollections
EcucModuleConfigurationValues	EcucModuleConfigurationValuess
EcucModuleDef	EcucModuleDefs
EcucValueCollection	EcucValueCollections
EndToEndProtectionSet	EndToEndProtectionSets
EthernetCluster	CommunicationClusters
EthernetFrame	Frames
EvaluatedVariantSet	EvaluatedVariantSets
FMFeature	FMFeatureModels
FMFeatureMap	FMFeatureMaps
FMFeatureModel	FMFeatureModels
FMFeatureSelectionSet	FMFeatureModelSelectionSets
FlatMap	FlatMaps
FlexrayArTpConfig	TpConfigs
FlexrayCluster	CommunicationClusters
FlexrayFrame	Frames



Element kind	<i>recommendedPackage</i>
FlexrayTpConfig	TpConfigs
Gateway	Gateways
HwCategory	HwCategorys
HwElement	HwElements
HwType	HwTypes
ISignal	ISignals
ISignalGroup	ISignalGroup
ISignalIPdu	Pdus
ISignalIPduGroup	ISignaliPduGroup
ImplementationDataType	ImplementationDataTypes
InterpolationRoutineMappingSet	InterpolationRoutineMappingSets
J1939Cluster	CommunicationClusters
J1939DcmIPdu	Pdus
J1939TpConfig	TpConfigs
KeywordSet	KeywordSets
LifeCycleInfoSet	LifeCycleInfoSets
LifeCycleStateDefinitionGroup	LifeCycleStateDefinitionGroups
LinCluster	CommunicationClusters
LinEventTriggeredFrame	Frames
LinSporadicFrame	Frames
LinTpConfig	TpConfigs
LinUnconditionalFrame	Frames
McFunction	McFunctions
ModeDeclarationGroup	ModeDeclarationGroups
ModeDeclarationMappingSet	PortInterfaceMappingSets
ModeSwitchInterface	PortInterfaces
MultiplexedIPdu	Pdus
NPdu	Pdus
NmConfig	NmConfigs
NmPdu	Pdus
NvBlockSwComponentType	SwComponentTypes
NvDataInterface	PortInterfaces
ParameterInterface	PortInterfaces
ParameterSwComponentType	SwComponentTypes
PdurIPduGroup	PdurIPduGroups
PhysicalDimension	PhysicalDimensions
PhysicalDimensionMappingSet	PhysicalDimensionMappingSets
PortInterfaceMappingSet	PortInterfaceMappingSets
PortPrototypeBlueprint	PortPrototypeBlueprints
PostBuildVariantCriterion	PostBuildVariantCriteria
PostBuildVariantCriterionValueSet	PostBuildVariantCriterionValueSets
PredefinedVariant	PredefinedVariants
RapidPrototypingScenario	RapidPrototypingScenarios
SenderReceiverInterface	PortInterfaces

Element kind	<i>recommendedPackage</i>
SensorActuatorSwComponentType	SwComponentTypes
SerializationTechnology	SerializationTechnologies
ServiceProxySwComponentType	SwComponentTypes
ServiceSwComponentType	SwComponentTypes
SoAdRoutingGroup	SoAdRoutingGroups
SwAddrMethod	SwAddrMethods
SwAxisType	SwAxisTypes
SwBaseType	BaseTypes
SwRecordLayout	SwRecordLayouts
SwSystemconst	SwSystemconsts
SwSystemconstantValueSet	SwSystemconstantValueSets
SwcBswMapping	SwcBswMappings
SwcImplementation	SwcImplementations
SwcTiming	TimingExtensions
System	Systems
SystemSignal	SystemSignals
SystemSignalGroup	SystemSignalGroups
SystemTiming	TimingExtensions
TriggerInterface	PortInterfaces
TtcanCluster	CommunicationClusters
Unit	Units
UnitGroup	UnitGroups
UserDefinedCluster	CommunicationClusters
UserDefinedIPdu	Pdus
UserDefinedPdu	Pdus
VfbTiming	TimingExtensions
ViewMapSet	ViewMapSets
XcpPdu	Pdus

This list is copied from AUTOSAR-MetaModel\_master.RecommendedPackage.xml, release 4.1.1.

## Rule ARPac\_14A: Sort kind-based *ARPackages* alphabetically

### Instruction

For better readability the kind-based *ARPackages* should be sorted alphabetically.

## Rule ARPac\_15: Omit empty *ARPackages*

### Instruction

If an *ARPackage* contains no *ARElements* you may omit it.

## Rule ARPac\_16: *ARPackage* for Common *ARElements*

### Instruction

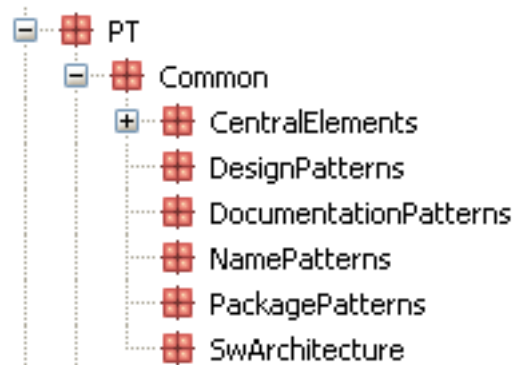
Scope: AR4.x

There are different kinds of common *ARElements*. They shall be covered by a *ARPackage* named "Common". The actually known common element types are: *CentralElements*, *DesignPatterns*, *DocumentationPatterns*, *NamePatterns*, *PackagePatterns*, *SwArchitecture*, *NamingConventions*.

The Common- *ARPackages* shall be used on that *ARPackage*-level for which the elements are common. This can be the level "RB" or the level of a domain or of a sub domain.

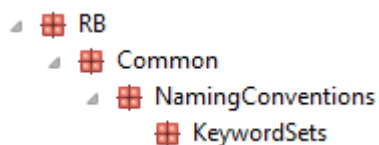
Examples for the Common- *ARPackage*:

Figure 63 Possible domain common *ARPackages*



The meaning of these *ARPackages* can be found in Guideline Artifact [\[A\\_Blueprint\]](#).

Figure 64 Possible UBK common *ARPackages*



More information on *KeywordSets* can be found in [\[A\\_AppII\]](#).

## Rule ARPac\_17: Category of *ARPackages*

### Instruction

The following *Categories* for *ARPackage* are predefined by AUTOSAR:

- ▶ *BLUEPRINT* (blueprint elements)
- ▶ *STANDARD* (standardized objects)
- ▶ *ICS* (Implementation Conformance Statement)
- ▶ *EXAMPLE* (examples how to apply Elements in *STANDARD* or *BLUEPRINT* packages)

You shall NOT define any other category for *ARPackages*.

## Rule ARPac\_18: Empty Category of *ARPackages*

**Instruction** For *ARPackages* which contain model elements of none of the categories *BLUEPRINT*, *STANDARD*, *ICS*, *EXAMPLE* you shall NOT set any category. *Category* attribute shall be omitted.

This is the standard case for most *ARPackages*.

## Rule ARPac\_191: Category *BLUEPRINT* for *ARPackages*

### Instruction

The *Category* of *ARPackages* containing blueprint elements shall be set to *BLUEPRINT*.

## Rule ARPac\_192: Category *STANDARD* for *ARPackages*

### Instruction

The *Category* of *ARPackages* containing standardized elements shall be set to *STANDARD*. This *Category* is typically created only by architects.

## Rule ARPac\_20: Category *ICS, EXAMPLE* for *ARPackages*

### Instruction

The *Category*s *ICS, EXAMPLE* shall NOT be used since the usage is not defined in our methods.

**TBC:** specific rule required for *SwSystemconsts*?

Siehe ARMETH-963.

## 9.3 *ARPackage* for Basic Software

### Rule ARPac\_41: Top level *ARPackages* for AUTOSAR Specified BSW Modules

#### Instruction

**Scope:** BSW modules specified by AUTOSAR

Each basic software module, if contained in AUTOSAR's BSW module list ( [\[TR\\_BSWModuleList\]](#) ) or in AUTOSAR's list of "virtual modules" (*TR\_PDN\_00001* in [\[TR\\_PDN\]](#) ), shall define its own *ARPackage* as top level *ARPackage*.

The *ShortName* shall be "AUTOSAR\_" + {moduleName}.

List of BSW-Modules is given in table [Table 47 \[BSWModuleNames\]](#) , taken from AUTOSAR specification 4.1.1.

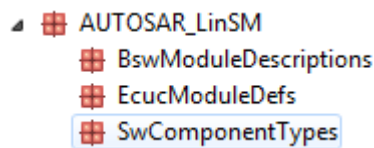
Table 47 List of AUTOSAR specified BSW-Modules (AR 4.1.1)

	list of BSW modules	source of this list
BSW module list	Adc, Com, BswM, SchM, Can, CanIf, CanNm, CanSM, CanTrcv, CanTp, ComM, CorTst, Csm, Dbg, Det, Dcm, Dem, Dlt, DoIP, Dio, EcuM, Ea, Eep, Eth, EthIf, EthTrcv, UdpNm, EthSM, Fls, Fee, FlsTst, Fr, FrIf, FrNm, FrSM, FrTrcv, FrArTp, FrTp, FiM, Gpt, Icu, IpduM, Lin, LinIf, LinNm, LinSM, LinTrcv, Mcu, MemIf, Nm, NvM, Ocu, Os, PduR, Port, Pwm, RamTst, Rte, J1939Dcm, J1939Nm, J1939Rm, J1939Tp, Sd, SoAd, Spi, StbM, TcpIp, Tm, Ttcan, TtcanIf, Wdg, WdgIf, WdgM, Xcp, Crc, Bfx, Cal, E2E, Efx, Ifl, Mfl, Mfx, Ifx	AUTOSAR_TR_BSWModuleList.xls ( <a href="#">[TR_BSWModuleList]</a> ) For more information see <i>TPS-GST_00017</i> in <a href="#">[TPS_GST]</a>
virtual modules	AISpecification, Compiler, Comtype, EcuC, GenDef, Platform, Std	<i>TR_PDN_00001</i> in <a href="#">[TR_PDN]</a>

**Note:** even though AUTOSAR has defined this list, there were AUTOSAR example files existing which were using *Standard* instead of *Std* and *PlatformTypes* instead of *Platform*.

Example:

Figure 65



Please note that also Rte is kept in this list. *McSupport* data are specified as BSW module description of Rte and so shall also follow this package structure starting with *AUTOSAR\_Rte*. *McSupport* data are generated by RTE-generator.

## Rule ARPac\_43: Top level *ARPackage*s for non AUTOSAR Specified BSW Modules

### Instruction

**Scope:** BSW modules not specified by AUTOSAR

Such modules shall be put into the top level *ARPackage* "RB".

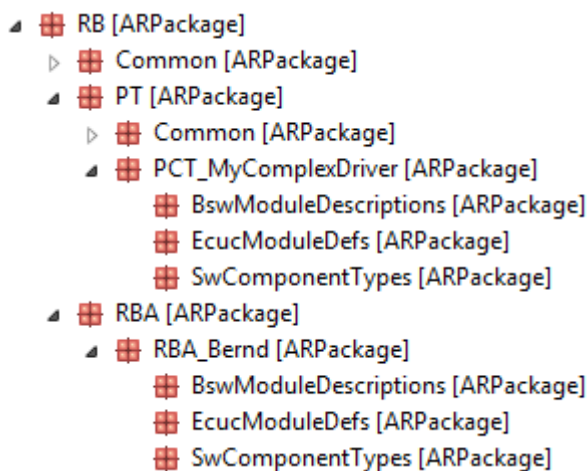
They shall be put into the second level *ARPackage* for the concerning domain. This *ARPackage* shall be either "RBA" if it is related to RBA, otherwise the ECU domain (e.g. PT for power train).

The *ARPackage* on the next level shall get the name of the concerning BSW module.

Please note that the naming convention used for "RBA"-modules recommends a prefix "rba\_" as part of the module name.

Possible examples are:

Figure 66



## Rule ARPac\_44: Kind *ARPackage*s for Basic Software Modules

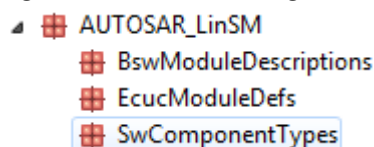
### Instruction

For BSW modules the rule "*Rule ARPac\_14: Names for ARElement-kind based ARPackages*" shall be followed.

It shall be followed by AUTOSAR specified and non AUTOSAR specified BSW modules.

Possible examples are:

Figure 67 Kind *ARPackage*s for Basic Software Modules



The package `/.../EcucModuleDefs` is to be used for parameter definitions. Note that AUTOSAR itself does not follow this rule and provides its standardized parameter definitions in a package `/AUTOSAR/EcucDefs` (see *TPS\_GST\_00082*). But this shall NOT be used in Bosch's implementations.

## Rule ARPac\_47: removed

## Rule ARPac\_46: Software Module Configuration (for protected values)

### Instruction

Protected configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following *ARPackage* structure:

```
/RB
  /UBK
    /Project
      /EcucModuleConfigurationValuess
```

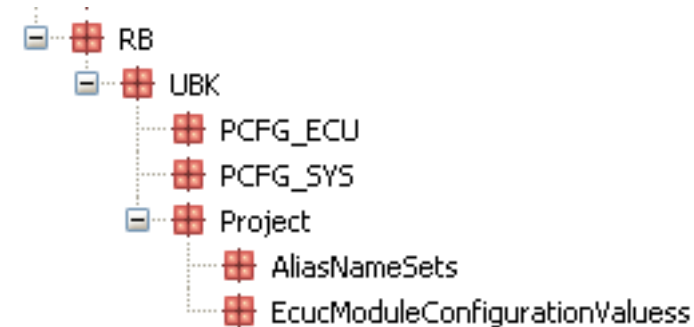
**Note:** the word "Project" is a keyword and shall not be replaced by any project's name. Reason: Tools will merge all configuration values coming from different files only if they are elements of the same *ARPackage*.

Protected configuration values shall **NOT** be put into the BSW module's *ARPackage* (e.g. `/AUTOSAR_LinSM` or `/RB/RBA-/RBA_Bernd`).

Protected values are set as fixed values by the developer of the BSW module and shall not be changed by project specific configuration values.

The structure looks as follows:

Figure 68



Note: the double "s" at the end of `EcucModuleConfigurationValuess` is not a misspelling. It is the result of the common rule to add a plural-s to the element kind.

## Rule ARPac\_45: Software Module Configuration (preconfigured or recommended values) --- currently **NOT** used

### Instruction **Scope:** upcoming

Preconfigured or recommended ecuc values for BSW modules (AUTOSAR specified or not) shall be put into the *ARPackage* of the module itself and there in a sub package *EcucModuleConfigurationValuess*.

Examples:

```
/AUTOSAR_LinSM/EcucModuleConfigurationValuess
/RB/RBA/rba_Bernd/EcucModuleConfigurationValuess
```

For preconfigured values the container's *ShortName* shall be *PreconfiguredValues*.

For recommended values the container's *ShortName* shall be *RecommendedValues*.

**Note:** Currently neither preconfigured nor recommended values are really defined. So this rule will not be used.

## 9.4 ARPackage for Project based ARElements

**TBD:** All the rules in this chapter are somewhat preliminary because there is not enough experience from practical work. So some of these rules may be changed in future.

Rule ARPac\_50: removed

Rule ARPac\_50v1: *ARPackages* for Projects

### Instruction

*ARPackages* for projects mainly shall follow [ARPac\_10] .

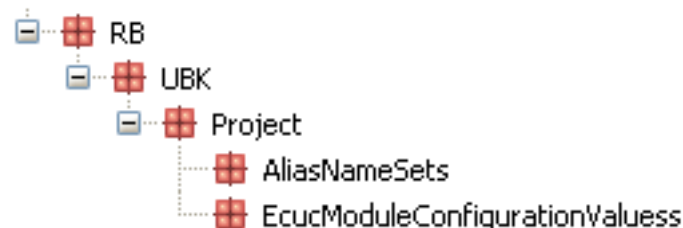
The contents of {domain} shall be set to the fixed word *UBK*.

The contents of {block} shall be set to the fixed word *Project*.

There are some other rules in this chapter which define some exceptions.

The structure may look as follows:

Figure 69



Rule ARPac\_51: *ARPackage* for top level component type

### Instruction

The name for {block}, as defined in [ARPac\_10] shall be set to the name of the top level composition type. This name will be defined by ECU software architects, e.g. to *RootSwCompo*.

A prefix may be added (e.g. *PCT\_* for DGS's power train).

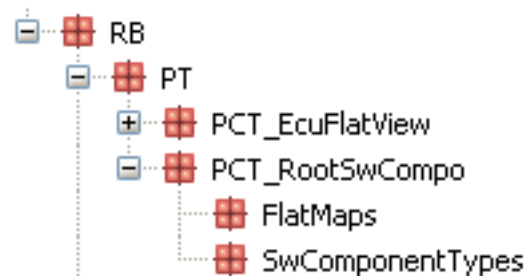
Typically there will be some sub- *ARPackages*:

- ▶ *SwComponentTypes* for the component type
- ▶ *FlatMaps* for the flat map
- ▶ *CalibrationParameterValueSets*

Also for the ECU flat view composition a name may be defined by ECU software architects, e.g. to *EcuFlatView*.

The structure may look as follows:

Figure 70



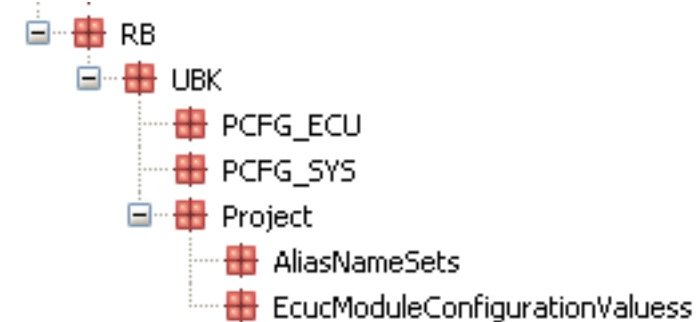
## Rule ARPac\_54: *ARPackage* for *AliasNameSets*

### Instruction

The *ARPackage* for *AliasNameSets* typically shall reside in the *ARPackage* of the project as defined in Rule [\[ARPac\\_50v1\]](#)

The structure may look as follows:

Figure 71



## Rule ARPac\_52: ECUC Software Module Configuration

### Instruction

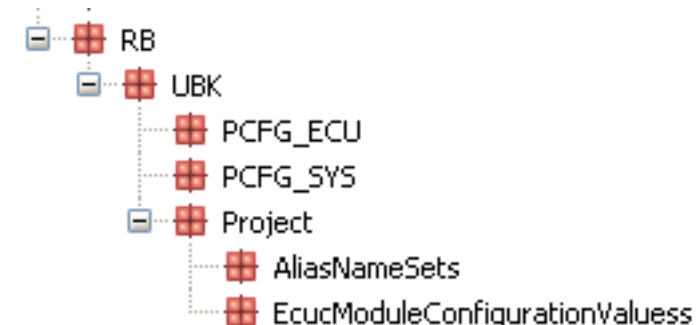
Configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following *ARPackage* structure:

```
/RB
  /UBK
    /Project
      /EcucModuleConfigurationValues
```

**Note:** the word "Project" is a keyword and shall not be replaced by the project's name. Reason: Tools will merge all configuration values coming from different files only if they are elements of the same *ARPackage*.

The structure looks as follows:

Figure 72



## Rule ARPac\_53: *ARPackage* for Configuration of ECU, RTE, OS

### Instruction

The configuration values for ECU, RTE, OS shall be handled as defined in [\[ARPac\\_52\]](#).

## Rule ARPac\_55: System and ECU based Elements

### Instruction

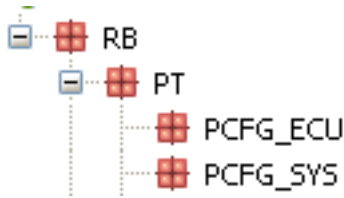
Elements based on a system or ECU should use *ARPackage* names which contain *SYS* resp. *ECU*. A prefix *PCFG\_* (for project configuration) may be used.



Examples are packages for the hardware topology, system extract, RTE container tasks. Detailed rules are defined in [\[G\\_Integ\]](#).

Example:

Figure 73



### *ARPackage for McSupport*

*McSupport* data are created by RTE generator to support the generation of A2L files for **m**asurement and **c**alibration.

In AUTOSAR specification it is handled as module description (BSWMD) of the Rte. So also for *ARPackage* it is handled as a BSW module and will follow rules [\[ARPac\\_41\]](#) and following.

No extra rule is to be defined for *McSupport*.

## 9.5 Referencing Elements

### 9.5.1 References to other *ARElements*

When referencing an *ArElement* you have to create a string which represents the *ARPackage*-Structure separated by "/".

Example for a reference to a unit:

```
<UNIT-REF DEST="UNIT">/RB/PT/Common/CentralElements/Units/Rpm</UNIT-REF>
```

If you use a specific AUTOSAR authoring tool it typically will provide functionality for entering a reference and you don't have to type this path manually.

No rules are defined in this chapter since it is defined by AUTOSAR how to use a reference.

### 9.5.2 Use of *ReferenceBase*

Since AUTOSAR release 4 a mechanism of *ReferenceBase* is provided. With this the deep *ARPackage*-paths can be shortened inside of a reference and maintainability can be eased.

If you define

```

<REFERENCE-BASES>
  <REFERENCE-BASE>
    <SHORT-LABEL>PT_CEL_Units</SHORT-LABEL>
    <PACKAGE-REF DEST="AR-PACKAGE">/RB/PT/Common/CentralElements/Units</PACKAGE-REF>
  </REFERENCE-BASE>
  ...
</REFERENCE-BASES>
  
```

inside an *ARPackage* you can refer to an object in the following form:

```
<UNIT-REF BASE="PT_CEL_Units" DEST="UNIT">Rpm</UNIT-REF>
```

## Rule ARPac\_71: Avoid usage of Element *ReferenceBase*

### Instruction

The usage of the Element *ReferenceBase* is not yet supported by all authoring tools and should NOT be used.

Exceptions are defined in subsequent rules.

## Rule ARPac\_72: Usage of *ReferenceBase*

### Instruction

In tool generated arxml files *ReferenceBase* should be used for making the files more readable.

## Rule ARPac\_73: A defined *ReferenceBase* shall always be used

### Instruction

If a *ReferenceBase* to an *ARPackage* is defined it shall be used in **all** references going to the elements in this *ARPackage*, including sub- *ARPackages*.

Otherwise it will be dangerous when someone changes the *PackageRef* of a *ReferenceBase* expecting that all references will be affected.

This would require a functionality in authoring tools that, when creating a new reference, existing reference base shall be used.

## Rule ARPac\_74: *ShortName* of *ReferenceBases*

### Instruction

*ShortName* of a *ReferenceBase* shall follow the following pattern: {*Scope*}\_{*ElementKind*} where *Scope* is the scope wherefore the objects are defined (e.g. the delivery package) and *ElementKind* shall be taken out of the list as defined in Rule [ARPac\_14] .

Examples: *PTCEL\_Units*; *CUCEL\_RecordLayouts*.

## Rule ARPac\_75: *ReferenceBases* and their usage in references shall reside in the same delivery unit

### Instruction

The definition of a *ReferenceBase* shall be in the same file as their usage in references.

**Exception:** the *ReferenceBase* for referencing central elements may be defined in central file(s).

Background of this rule is, that a delivery unit using *ReferenceBases* could change the whole behavior when the reference to other objects changes.

## Criteria for Central Elements

- ▶ Central mechanism can require a central allocation, e.g. SwSystemConst Configuration to handle variance.

This chapter identifies the kind of elements to be handled as central element.

## Rule CEL 001: Unit

- ▶ Unit

- ▶ DisplayUnit

[illegible]

## Rule CEL\_031: UnitGroup

**Instruction** The following AUTOSAR elements shall be defined centrally or locally:

- ▶ UnitGroup

## Rule CEL\_041: PhysicalDimension

**Instruction** The following AUTOSAR elements shall be defined centrally:

- ▶ PhysicalDimension

## 10.1.2 CompuMethods

### Rule CEL\_051: Arithmetic CompuMethod

**Instruction** The following AUTOSAR elements shall be defined mainly centrally:

- ▶ Arithmetic CompuMethod (categories: RAT\_FUNC, LINEAR, IDENTICAL)

In special cases, e.g. conversions for complex drivers, the CompuMethod may be allocated locally.

**Hint** If a CompuMethod is needed earlier than it is provided in a central package it can be defined locally. If in future a centrally defined CompuMethod is available the developer can switch to that CompuMethod. But it is currently not a must to switch.

### Rule CEL\_052: Verbal CompuMethod

**Instruction** The following AUTOSAR elements shall be defined centrally or locally:

- ▶ CompuMethod of category TEXTTABLE

### Rule CEL\_055: Referencing of Units by CompuMethod

**Instruction** Arithmetic and verbal CompuMethods shall refer to an Unit. If the CompuMethod has no Unit, the Unit with the shortname *NoUnit* shall be referred.

## 10.1.3 System Constants

### Rule CEL\_002: SwSystemconst

**Instruction** The following AUTOSAR elements may be defined centrally or locally:

- ▶ SwSystemconst

This elements shall be defined centrally, if the element to be used product line wide in standardized mechanisms, e.g. configuration purpose. Detailed discussion to be done in AUTOSAR TWG variant handling.

In contrast to de-central component individual usage, e.g. where an SwSystemconst represents a generic *#define*.

## 10.1.4 Address Methods

### Rule CEL\_003: SwAddrMethod

**Instruction** The following AUTOSAR elements shall be defined centrally:

- ▶ SwAddrMethod

## 10.1.5 Record Layouts

### Rule CEL\_004: SwRecordLayout

**Instruction** The following AUTOSAR elements are defined in the packages IFX and IFL, which are not part of the CEL packages but are still central elements:

- ▶ SwRecordLayout

**Hint** The SwRecordLayouts are defined within the BSW packages IFX *Document "Specification of Fixed Point Interpolation Routines (IFx Library)" [SWS\_Ifx]* and IFL *Document "Specification of Floating Point Interpolation Routines (IFl Library)" [SWS\_Ifl]* by AUTOSAR. IFX and IFL are not part of a CEL package. The SwRecordLayouts are generally valid and have the character of central elements even if they are not delivery by the central package.

AR Package Paths:

/AUTOSAR\_Ifx/SwRecordLayouts/... (for IFX)

/AUTOSAR\_Ifl/SwRecordLayouts/... (for IFL)

## 10.1.6 Keywords

### Rule CEL\_005: Keywords

**Instruction** The following AUTOSAR elements shall be defined centrally for all RB product lines:

- ▶ KeywordSets

**Hint:** The KeywordSets contain the Keywords.

## 10.1.7 Data types

### Rule CEL\_006: ImplementationDataType

**Instruction** The following AUTOSAR elements may be defined centrally and locally:

- ▶ ImplementationDataType

### Rule CEL\_061: ApplicationDataType

**Instruction** The following AUTOSAR elements may be defined centrally or locally:

- ▶ ApplicationDataType

## 10.1.8 SwBaseType

### Rule CEL\_062: SwBaseType

**Instruction** The following AUTOSAR elements shall be defined centrally.

- ▶ SwBaseType

That means that SwBaseTypes shall not be defined outside of the central elements. Only references to SwBaseTypes are allowed and needed from such other AUTOSAR elements.

## 10.1.9 Data Constraints

### Rule CEL\_063: DataConstraints

**Instruction** The following AUTOSAR elements shall be defined centrally, if the AUTOSAR element, which is referencing it, is defined centrally, too.

- ▶ DataConstraints

## 10.1.10 DataTypeMappingSet

### Rule CEL\_064: DataTypeMappingSet

**Instruction** The following AUTOSAR elements are defined centrally and locally:

- ▶ DataTypeMappingSet

It may only be defined centrally if the referenced ApplicationDataType AND the referenced ImplementationDataType are defined centrally, too. In this case the DataTypeMappingSet has the same parent AR package as the Application-DataType (and the same delivery unit).

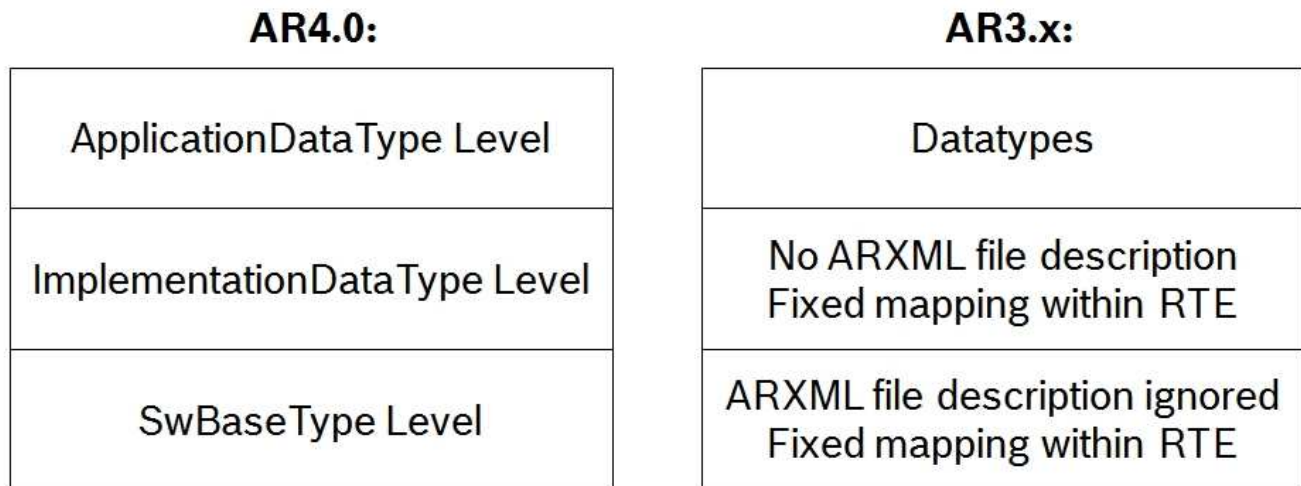
A user of a centrally defined ApplicationDataType shall refer to the centrally defined DataTypeMappingSet and shall not create an own local DataTypeMappingSet.

## 10.2 Data Types

### 10.2.1 ApplicationDataTypes and ImplementationDataTypes

Comparison Structure of Types between AR4.x and AR 3.x.

Figure 74 Comparison Between Data Types

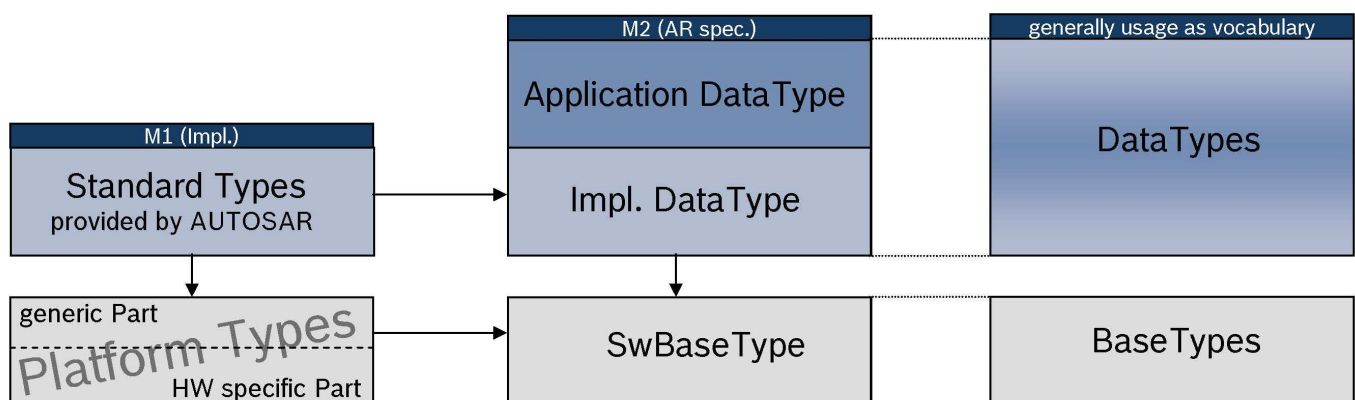


Datatypes in AR3.x do correspond to level of ApplicationDataType of AR4.x.

In AR 4.x the details on DataTypes are described in AUTOSAR Chapter 5.1 [Document "Software Component Template" \[TPS\\_SWCT\]](#). The concept of AUTOSAR Standard Types is defining data types which could reference to a Implementation Type this represents a platform type.

General concept of platform types. The following diagram illustrates the relation of types between the AUTOSAR specification (M2 definitions which provides the meta model) and its instantiation, the implementation level for platform and standard types.

Figure 75 Overview on AUTOSAR Types



The major aspect splitting the platform types into a platform (processor/microcontroller specific) part and into a platform independent part. This allows stable and independent layers on top.

#### Rule CEL\_008: Constraints Referenced by ApplicationDataTypes and ImplementationDataTypes

**Instruction** For all centrally defined ApplicationDataTypes and ImplementationDataTypes the referenced data constraint shall be defined centrally too.

Please consider also rule [\[CEL\\_063\]](#).

## Rule CEL\_050: Reuse of ImplementationDataTypes

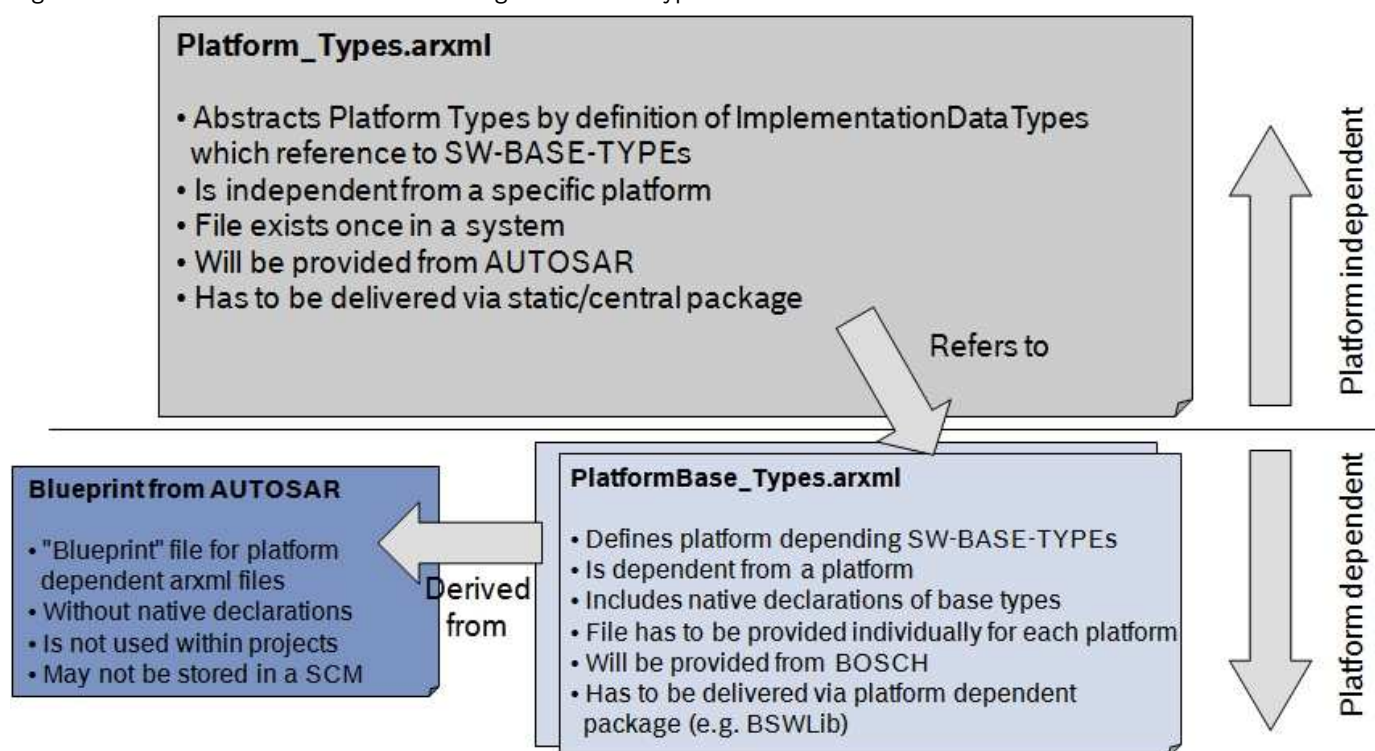
**Instruction** The reuse of ImplementationDataTypes is encouraged. This implies that there should be no 1:1 mapping between application and implementation data types.

## Rule CEL\_054: ImplementationDataType References

**Instruction** ImplementationDataTypes shall not be referenced directly, but via a DataTypeMappingSet.

## 10.2.2 AUTOSAR Platform Types

Figure 76 Common Overview on Handling of Platform Types



The same short name is used for IMPLEMENTATION-DATA-TYPE and SW-BASE-TYPE. The native declaration can be easily adapted without changing the short name. Hence the references are stable.



Figure 77 Detailed Overview of Handling of Platform Types Based on ARXML Files for a Specific Controller

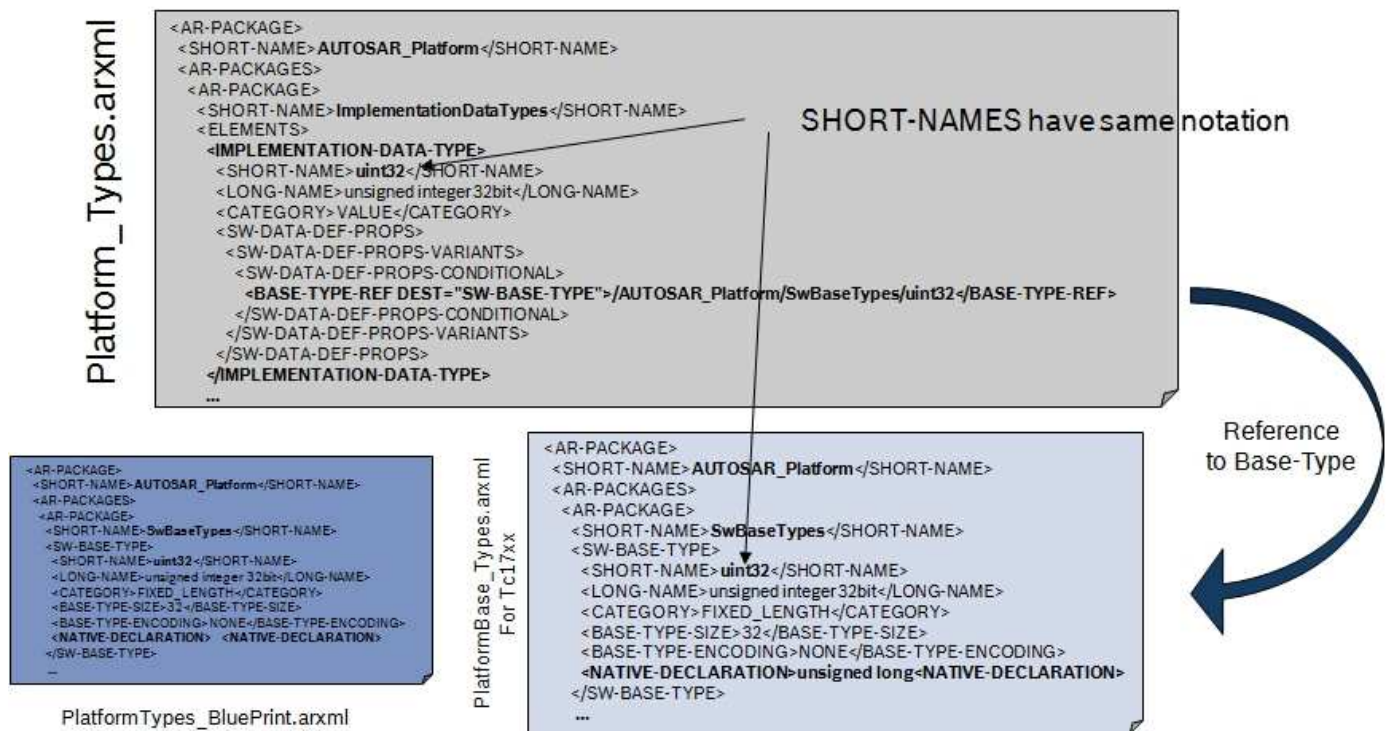
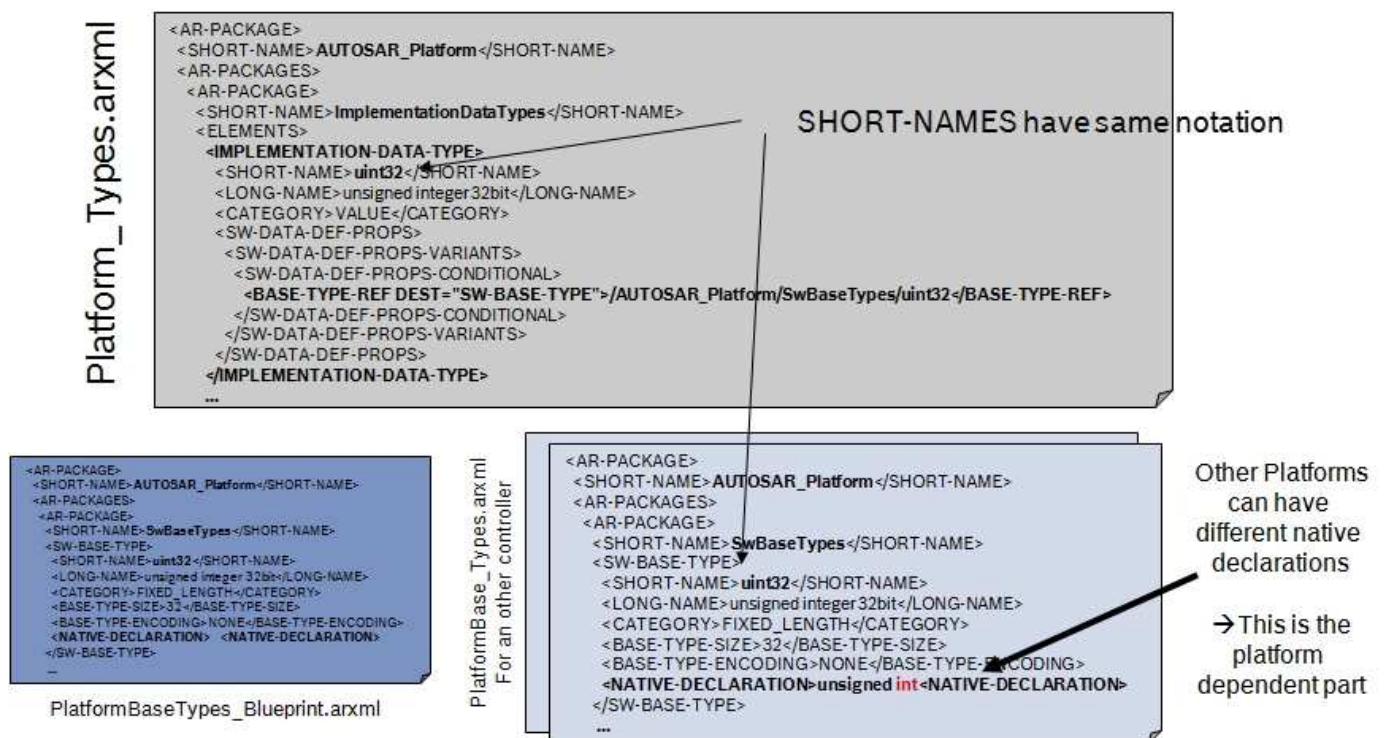


Figure 78 Detailed Overview of Handling of Platform Types Based on ARXML Files for an Other Controller



## AR 4.x:

All central types and symbols which are used on ApplicationDataType level are defined as ImplementationDataTypes. To reference to them a correct path has to be set in the ARXML file.

## Examples:

To refer to ImplementationDataType "uint32" following path has to be set:

```

<IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"/>AUTOSAR_Platform
/ImplementationDataTypes/uint32</IMPLEMENTATION-DATA-TYPE-REF>

```

Reference to ImplementationDataType "Std\_ReturnType" :  
`<IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">/AUTOSAR_Std  
 /ImplementationDataTypes/Std_ReturnType</IMPLEMENTATION-DATA-TYPE-REF>`

## Rule CEL\_007: AUTOSAR PlatformTypes

**Instruction** PlatformTypes are defined in the following file artefacts:

- ▶ Platform\_Types.arxml (Hardware independent)
- ▶ PlatformBase\_Types.arxml (Hardware dependent)

The ASW may only reference the PlatformTypes, but never the PlatformBaseTypes.

## Rule CEL\_025: Reference Base AUTOSAR Platform Types

**Instruction** Reference base concept should be used instead of full qualified package path.

With the reference base concept it is possible to define references which substitutes the path of the AUTOSAR Platform Types.

With a reference base "Platform\_ImplementationDataTypes" (which is conform to the rules for reference bases in AR package guideline, Rule ARPac\_71 to Rule ARPac\_75) the definition can be written in following form:

```
<IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"
BASE= "Platform_ImplementationDataTypes">uint32</IMPLEMENTATION-DATA-TYPE-REF>
```

## 10.2.3 AUTOSAR Standard Types

### Rule CEL\_091: AUTOSAR Standard Types

**Instruction** AUTOSAR Standard Types are elements of ARPackage /AUTOSAR\_Std/ImplementationDataTypes. The Category of this ARPackage is "STANDARD".

Example:

```
<AR-PACKAGE>
  <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
  <CATEGORY>STANDARD</CATEGORY>
```

### Rule CEL\_026: Reference Base AUTOSAR Standard Types

**Instruction** Reference base concept should be used instead of full qualified package path.

With the reference base concept it is possible to define references which substitutes the path of the AUTOSAR Standard Types.

With a reference base "Std\_ImplementationDataTypes" (which is conform to the rules for reference bases in AR package guideline) the definition can be written in following form:

```
<IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"
BASE= "Std_ImplementationDataTypes">E_OK</IMPLEMENTATION-DATA-TYPE-REF>
```

## 10.2.4 ComStack Types

ComStack Types define only own symbols which refer to ImplementationDataTypes, which are defined as Platform Types. These elements are defined centrally and delivered by CUCEL.

## 10.2.5 Other Types

### Rule CEL\_009: RB Standard Types

**Instruction** For RB additional standard types besides the AUTOSAR standard types can be defined. They are called RB Standard Types.

## 10.3 AR Package Structure for Central Elements

### Rule CEL\_011: AR Package Structure for Central Elements

**Instruction** Path declaration

- ▶ for UBK ASW: /RB/Common/CentralElements/{kind}s
- ▶ for BSW/CUBAS + ASW: /RB/RBA/Common/CentralElements/{kind}s
- ▶ for Powertrain ASW: /RB/PT/Common/CentralElements/{kind}s
- ▶ for Chassis ASW: /RB/Chassis/Common/CentralElements/{kind}s
- ▶ for BSW but not CUBAS: **under discussion**.

### Rule CEL\_020: Referencing to Central Elements

**Instruction** An ASW product line shall not use central elements of another product line. An ASW product line may use UBK common central elements, CUBAS central elements, their own central elements and the AUTOSAR central elements.

E. g. ASW PT may use /RB/Common/CentralElements, /RB/RBA/Common/CentralElements and /AUTOSAR\_\*/ and /RB/PT/Common/CentralElements.

E. g. ASW PT shall not use /RB/Chassis/Common/CentralElements.

BSW may access to /AUTOSAR/, /AUTOSAR\_\* and /RB/RBA, but not to /RB/Common/CentralElements.

Reason: Self-Containment of CUBAS, because CUBAS should be delivered as separate product.

### Rule CEL\_013: Package Path for AUTOSAR SwBaseTypes

**Instruction** Path for SwBaseTypes according to AUTOSAR:

/AUTOSAR\_Platform/SwBaseTypes/{NameOfSwBaseType}

The category of these elements is STANDARD.

In the package CUCEL following SwBaseTypes are provided. The type in the table is the replacement for {NameOfSwBaseType}. Please also consider rule [\[CEL\\_062\]](#).

Table 48 List of SwBaseTypes

Type	Description
<b>boolean</b>	boolean unsigned integer
<b>uint8</b>	8 bit unsigned integer
<b>sint8</b>	8 bit signed integer
<b>uint16</b>	16 bit unsigned integer
<b>sint16</b>	16 bit signed integer

Type	Description
<b>uint32</b>	32 bit unsigned integer
<b>sint32</b>	32 bit signed integer
<b>float32</b>	Single precision (32 bit) floating point number (consider 2nd hint below)
<b>float64</b>	Double precision (64 bit) floating point number (consider 2nd hint below)
<b>uint8_least</b>	At least 8 bit unsigned integer
<b>sint8_least</b>	At least 8 bit signed integer
<b>uint16_least</b>	At least 16 bit unsigned integer
<b>sint16_least</b>	At least 16 bit signed integer
<b>uint32_least</b>	At least 32 bit unsigned integer
<b>sint32_least</b>	At least 32 bit signed integer

For more details refer to the delivery notes of package CUCEL. Here SwBaseTypes are provided from the module "Platform".

## Rule CEL\_012: Package Path for AUTOSAR Standard Types

**Instruction** Path for standard types according to AUTOSAR:

/AUTOSAR\_Std/ImplementationDataTypes/{NameOfStandardType}

The category of these elements is STANDARD.

In the package CUCEL following AUTOSAR Standard Types are provided which are specified in [Document "Standard Types" \[TR\\_Std\]](#). The type in the table is the replacement for {NameOfStandardType}.

Table 49 List of AUTOSAR Standard Types

Type	Description
<b>Std_ReturnType</b>	Type of a standard return type. Std_ReturnType is based on data type uint8.
<b>Std_VersionInfoType</b>	Type to request the version of a BSW module. Std_VersionInfoType is a structure containing following elements: <ul style="list-style-type: none"><li>▶ uint16 vendorID</li><li>▶ uint16 moduleID</li><li>▶ uint8 sw_major_version</li><li>▶ uint8 sw_minor_version</li><li>▶ uint8 sw_patch_version</li></ul>

For more details refer to the delivery notes of package CUCEL. Here AUTOSAR Standard Types are provided from the module "Standard".

## Rule CEL\_112: Package Path for AUTOSAR Platform Types

**Instruction** Path for platform types according to AUTOSAR:

/AUTOSAR\_Platform/ImplementationDataTypes/{NameOfPlatformType}

The category of these elements is STANDARD.

In the package CUCEL following AUTOSAR Platform Types are provided which are specified in [Document "Platform Types" \[TR\\_Platform\]](#). The type in the table is the replacement for {NameOfPlatformType}.

Table 50 List of AUTOSAR Platform Types

Type	Description
<b>boolean</b>	boolean unsigned integer
<b>uint8</b>	8 bit unsigned integer
<b>sint8</b>	8 bit signed integer
<b>uint16</b>	16 bit unsigned integer
<b>sint16</b>	16 bit signed integer
<b>uint32</b>	32 bit unsigned integer
<b>sint32</b>	32 bit signed integer
<b>float32</b>	Single precision (32 bit) floating point number (consider hint below)
<b>float64</b>	Double precision (64 bit) floating point number (consider hint below)
<b>uint8_least</b>	At least 8 bit unsigned integer
<b>sint8_least</b>	At least 8 bit signed integer
<b>uint16_least</b>	At least 16 bit unsigned integer
<b>sint16_least</b>	At least 16 bit signed integer
<b>uint32_least</b>	At least 32 bit unsigned integer
<b>sint32_least</b>	At least 32 bit signed integer

The list is similar to the list of SwBaseTypes in rule [\[CEL\\_013\]](#). But in normal case a user has to refer to an ImplementationDataType of the AUTOSAR Platform Types. Therefore the user shall use the AR package path defined shown above. References to SwBaseTypes are only needed in exceptional cases (e.g. definition of ImplementationDataTypes or RecordLayouts).

For more details refer to the delivery notes of package CUCEL. Here AUTOSAR Platform Types are provided from the module "Platform".

**Hint** On different  $\mu$ Cs often no floating point unit is available. Therefore float data types (float32 and float64) have to be used carefully. It has to be ensured that float can be used. Additionally float64 is more critical than float32 because  $\mu$ C supports more often float32 than float64.

Float data types shall not be used if the developed SW has a focus on complete HW and compiler independence. In such a use case float data types are not portable.

## Rule CEL\_053: Package Path for AUTOSAR ComStackTypes

**Instruction** Path for ComStackTypes according to AUTOSAR:

/AUTOSAR\_Comtype/ImplementationDataTypes/{NameOfComtype}

The category of these elements is STANDARD.

In the package CUCEL following AUTOSAR ComStackTypes are provided which are specified in [Document "Communication Stack Types" \[TR\\_Comtype\]](#). The type in the table is the replacement for {<NameOfComtype>}.

Table 51 List of AUTOSAR ComStackTypes

Type	Description
<b>PduIdType</b>	Used to define variables for unique identifiers for PDUs
<b>PduLengthType</b>	Used to define length information of a PDU which is provided in number
<b>PduInfoType</b>	Used to store the basic information about a PDU
<b>TpDataState</b>	Used to define variables to store the state of a Transport Protocol (TP)
<b>NotifResultType</b>	Used to define variables to store the result status of a notification (confirmation or indication)



Type	Description
<b>TPParameterType</b>	Used to specify the type of parameter of a Transport Protocol (TP)
<b>BufReq_ReturnType</b>	Used to store the result of a buffer request
<b>BusTrcvErrorType</b>	Used to return the bus status evaluated by a transceiver
<b>TpDataStateType</b>	Used to store the state of Transport Protocol buffer
<b>RetryInfoType</b>	Used to store the information about Transport Protocol buffer handling
<b>NetworkHandleType</b>	Used to store the identifier of a communication channel
<b>PNCHandleType</b>	Used to store the identifier of a partial network cluster

For more details refer to the delivery notes of package CUCEL. Here AUTOSAR ComStackTypes are provided from the module "ComStack".

## Rule CEL\_016: Package Path for RB Standard Types

**Instruction** Package path for RB standard types:

/RB/Common/CentralElements/ImplementationDataTypes or

/RB/{domain}/Common/CentralElements/ImplementationDataTypes.

The Category of the leaf ARPackage "ImplementationDataTypes" is "STANDARD".

Currently no RB Standard Types are specified within CUCEL or UBKCEL.

## Rule CEL\_113: Package Path for UBK CompuMethods

**Instruction** Path for centrally defined CompuMethods:

/RB/RBA/Common/CentralElements/CompuMethods/{NameOfCompuMethod}

In the package CUCEL following CompuMethods are provided. The type in the table is the replacement for {NameOfCompuMethod}.

Table 52 List of CUBAS Computation Methods

Type	Description
<b>Identical</b>	The internal value and the physical one are identical. No physical unit available. This computation method shall only be used within CUBAS. For more details refer to the delivery notes of package CUCEL. The computation method is provided from the module "rba_CUCELSwCompuMethods".

In future more computation methods will be defined.

## Rule CEL\_114: Package Path for SwAddrMethods

**Instruction** Path for centrally defined SwAddrMethods:

/RB/RBA/Common/CentralElements/SwAddrMethods/{NameOfSwAddrMethod}

In the package CUCEL following SwAddrMethods are provided. The type in the table is the replacement for {NameOfSwAddrMethod}.

Table 53 List of AUTOSAR SwAddrMethods

Type	Description
<b>INTERNAL_VAR_INIT</b>	Addressing method for global or static variables accessible from a calibration tool that are initialized with values after every reset
<b>INTERNAL_VAR_CLEARED</b>	Addressing method for global or static variables accessible from a calibration tool that are cleared to zero after every reset
<b>INTERNAL_VAR_POWER_ON_CLEARED</b>	Addressing method for global or static variables accessible from a calibration tool that are cleared to zero only after power on reset
<b>CALIB</b>	Addressing method for calibration constants
<b>CALIB_FAST</b>	Addressing method for calibration constants with frequently usage. This addressing method is an extension to AUTOSAR
<b>CALIB_SLOW</b>	Addressing method for calibration constants with rarely usage. This addressing method is an extension to AUTOSAR
<b>CODE</b>	Addressing method for code (application code, BSW code, boot code)
<b>CONFIG_DATA</b>	Addressing method for constants with attributes that show that they reside in one segment for link-time module configuration
<b>CONFIG_DATA_FAST</b>	Addressing method for constants with attributes that show that they reside in one segment for link-time module configuration. This addressing method is an extension to AUTOSAR
<b>CONFIG_DATA_SLOW</b>	Addressing method for constants with attributes that show that they reside in one segment for link-time module configuration. This addressing method is an extension to AUTOSAR
<b>CONST</b>	Addressing method for global or static constants
<b>CONST_FAST</b>	Addressing method for global or static constants with frequently usage. This addressing method is an extension to AUTOSAR
<b>CONST_SLOW</b>	Addressing method for global or static constants with rarely usage. This addressing method is an extension to AUTOSAR
<b>VAR_INIT</b>	Addressing method for global or static variables that are initialized with values after every reset
<b>VAR_CLEARED</b>	Addressing method for global or static variables that are cleared to zero after every reset
<b>VAR_POWER_ON_CLEARED</b>	Addressing method for global or static variables that are cleared to zero only after power on reset
<b>VAR_FAST_INIT</b>	Addressing method for global or static variables which are frequently used and that are initialized with values after every reset
<b>VAR_FAST_CLEARED</b>	Addressing method for global or static variables which are frequently used and that are cleared to zero after every reset
<b>VAR_FAST_POWER_ON_CLEARED</b>	Addressing method for global or static variables which are frequently used and that are cleared to zero only after power on reset
<b>VAR_SLOW_INIT</b>	Addressing method for global or static variables that are initialized with values after every reset. This addressing method is an extension to AUTOSAR
<b>VAR_SLOW_CLEARED</b>	Addressing method for global or static variables that are cleared to zero after every reset. This addressing method is an extension to AUTOSAR
<b>VAR_SLOW_POWER_ON_CLEARED</b>	Addressing method for global or static variables that are cleared to zero only after power on reset. This addressing method is an extension to AUTOSAR

For more details refer to the delivery notes of package CUCEL. Here SwAddrMethods are provided from the module "rba\_CUCELSwAddrMethods".

## Rule CEL\_115: Package Path for AUTOSAR KeywordSets

**Instruction** ARPackagePath for AUTOSAR KeywordSets:

00

05

/AUTOSAR\_AISpecification/KeywordSets/{NameOfKeywordSet} derived from /AUTOSAR/AISpecification/Keyword-Sets\_Blueprint/KeywordList

ARPackagePath for UBK KeywordSets:

/RB/Common/NamingConventions/KeywordSets/{NameOfKeywordSet}

10

Currently no AUTOSAR or UBK KeywordSets are specified within CUCEL or UBKCEL.

15

20

25

30

35

40

45

50

55

60

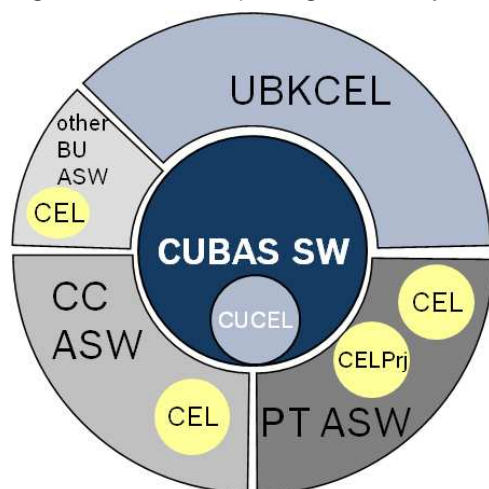
65

70



## 10.4 Delivery Aspects

Figure 79 Central packages/delivery units of UBK common software



### Rule CEL\_018: Delivery Units

**Instruction** A general assumption for the delivery unit names is done to refer always on common names within this guideline. Further this names can be mapped to SCM specific implementations.

The delivery unit for CUBAS central elements is CUCEL. CUCEL provides central elements for BSW but also elements which are relevant for BSW and ASW (e.g. PlatformTypes and StandardTypes).

The delivery unit for ASW central elements of AUTOSAR WP 10.x and defined for UBK are called UBKCEL.

The delivery unit for business unit specific central elements are the business unit specific CELs.

CUBAS software contains a delivery "CUCEL", which contains the central elements. This CUBAS specific central elements delivery unit contains the required central elements to development and deliver a self contained CUBAS software. CUBAS software should not depend on UBKCEL. UBKCEL contains the central elements for ASW of AUTOSAR WP 10.x and UBK. Hence CDG provides all UBK common central elements and AUTOSAR WP 10.x to the UBK customers. The process for the aspect of clearing is not part of this guideline.

**Under discussion** about assignment to a delivery unit: Elements for BSW, but not for CUBAS.

### Rule CEL\_019: Additional Central Element in Business Unit

**Instruction** Business units may define additional delivery units for central elements.

### Rule CEL\_027: Downward Compatibility of CUCEL and UBKCEL

**Instruction** CUCEL and UBKCEL are always downward compatible.

# 11 Rule Set: CLF (Classification File)

## 11.1 Introduction

The CLF (Classification File) technology was integrated in the projects of CDG departments to provide a simple mechanism which allows adaption of conventional resource systems (file and folder) by additional information as well as to support File and Folder based variant handling.

Since several years the CDG tool environment follows the concept to adapt file and folder informations. So it is possible to classify files and folders for specific Use Cases (Processors, Autosar, MSR, Documentation, etc. ). These additional informations can be provided by several project technologies like LWS-Catalog, the File Pattern Matcher technology (FPM) which is already used at CDG, as well as with CLF.

The file and folder based variant handling is a requirement of CDG department and the major reason to integrate the CLF concept at CDG. Therewith it is possible to work on a non variant project and to extract and test variant parts of it. Such a mechanism is very helpful especially in an area where the projects have to deliver to several customers and in several variants.

### 11.1.1 CLF projects overview

The additional information for files and folders and the variant mechanism are provided at so called \*.clf Files. At these files it is possible to define Packages, Variants and Rules.

*Packages* are objects which act as grouping elements. They are used to provide the entry point for the CLF technology (PAC, PAC\_D) as well as to define PAC, COMP, ST\_COMP and ECU\_CFG.

*Variants* are objects to provide the file and folder based variant handling at projects. They can be placed at each Package object and allow therewith to pass the variant handling from top (PAC/PAC\_D) to down (COMP, ECU\_CFG).

The files of projects get their additional information from the *Rule* objects and will be marked as relevant for processing tools like BCT-Build, CDG-Build and BlueDoc. These rules will be provided using file path and file naming pattern in a common way.

Additionally it is possible to ignore files and folders explicitly using *Ignore-Rule* objects. These objects use (as well as the Rule objects) Path and naming pattern of files and folders to ignore files or an entire part of project. For more details about the common usage and possibilities see "CLF Project User Guide".

### 11.1.2 Tool Environment

To configure the \*.clf files of a CLF project, it is recommended to use the latest version of *Autosar Workbench* (AWB) and ECU.WorX. In this tool the configuration of CLF-Files is supported by several UI elements like Metadata Navigator and CLF Editor.

The *Metadata Navigator* shows the defined CLF structure of the active project variant. Whenever CLF files will be configured and saved the changes of the structure will be visible at the navigator. The classification defined at \*.CLF files will be shown at the elements of Metadata Navigator as well as at the Filesystem Navigator (decoration of files and folders). Any CLF file will be opened with the *CLF Editor* by default. The Editor supports several helper functionalities like outline view, syntax highlighting, error highlighting, content assistance, code folding and templates.

More details about the UI elements can be read in the specific parts of Autosar Workbench help.

### 11.1.3 References

The following documents have to be considered together with this document.

For CLF there is a reference available in the Intranet:

#### ► CLF Documentation

The CLF documentation can be found in the Inside-Wiki of CDG-SMT [Document "CLF Documentation" \[CLFDocu\]](#)

## 11.2 Guidelines

This chapter defines the guidelines for editing and using CLF

### 11.2.1 Structural Conventions

#### Rule CLF\_Structure\_001: CLF for Packages, components and configuration

**Instruction** Each CUBAS package (PAC), structural component (STCOMP), component (COMP), as well as the corresponding configuration container (ECU\_CFG), shall have its own CLF file.

CLF files of components need to be included in the parent CLF package, e.g. structural component or package.

Examples: PAC Mcal and COMP Dio

CLF Name: Mcal.clf:

```
package Mcal
{
    class = PAC

    package-includes = Adc_Stc,
                      Dio,
                      Port
                      rba_Gtm
                      // more components separated by comma
}
```

CLF Name: Dio.clf:

```
package Dio
{
    class = COMP
}
```

Examples: ECU\_CFG for PAC Mcal and COMP Dio

CLF Name: Mcal\_Cfg.clf:

```
package Mcal_Cfg
{
    class = PAC

    package-includes = Adc_Stc_Cfg,
                      Dio_Cfg,
                      Port_Cfg
                      rba_Gtm_Cfg
                      // more components separated by comma
}
```

CLF Name: Dio\_Cfg.clf:

```
package Dio_Cfg
{
    class = ECU_CFG
}
```

## Rule CLF\_Structure\_002: Project

**Instruction** There shall be only one top-level CLF file that defines the project specific includes and variants.

Example: MyProject.clf

```
package MyProject
{
    class = PAC
    domain = CUBAS

    package-includes = Rules, MyProject_Rules // and other general CLF packages required
                                                // by the project, separated by comma

    variant PROJECT_VARIANT_ONE
    {
        // variant specific includes
    }

    variant PROJECT_VARIANT_TWO
    {
        // variant specific includes
    }

    // ...
}
```

**Hint** It is recommended to name the top level CLF file identical to the project name.

## Rule CLF\_Structure\_003: Package Classes

**Instruction** The classes to be used for CLF packages are the same as eASee.BASD container classes in CDG context:  
PAC, PAC\_D, STCOMP, COMP, ECU\_CFG, TEST

## Rule CLF\_Structure\_004: File Classes

**Instruction** The classes to be used for files are defined in a standard rules set from CDG-SMT/EMT (Rules.clf).

If the project needs additional rules which are not covered by the standard rules, a project specific rules file can be used. This rule set shall be included in the top level CLF file (e.g. MyProject.clf).

Rule files shall reside in the top level folder of the project, aside MyProject.clf

The standard Rules set is provided by CDG-SMT/EMT, it can be found in the Inside.Wiki of CDG-SMT: [\[Document CLF Documentation / URL: https://inside-wiki.bosch.com/confluence/display/CDGSMT/CLF+documentation\]](https://inside-wiki.bosch.com/confluence/display/CDGSMT/CLF+documentation)

Standard rules and project specific rules have to be included like this:

```
package-includes = Rules, MyProject_Rules // and other general CLF packages required
                                                // by the project, separated by comma
```

## Rule CLF\_Structure\_005: Exclusivity

**Instruction** In CUBAS packages, structural components, components and configuration there shall be only one CLF file in one folder.

To avoid conflicts or mutual exclusion with included rules, only one CLF file is permitted in each folder. Exception is the top level folder.

## Rule CLF\_Structure\_006: Variants

**Instruction** To switch between different configurations at the same project, e.g. for different micro controllers or test boards or ECU, the CLF variant handling shall be used.

Variants that are to be used across the whole project shall be defined in the top level project CLF file.

However, in order to be selectable by AWB, cdgb and ECU.Worx, those top level variants shall be defined in the top level project CLF file, e.g. MyProject.clf.

Any other variant in subjacent CLF packages cannot be selected at top level, they need to be derived from the top level variants.

However it is recommend to use only the top level variants instead of prodiving too many package local variants.

A top level variant excludes all the other available variants.

Using variants, folders or individual files of the project can be selected or discarded at any level of the CLF package structure.

Example: MyProject.clf

```
package MyProject
{
    class = PAC
    domain = CUBAS

    package-includes = Rules,
                     MyProject_Rules,
                     PackageOne

    variant PROJECT_VARIANT_ONE
    {
        // variant specific includes
        package-includes = PackageTwo(PROJECT_VARIANT_ONE),
                          PackageThree(PROJECT_VARIANT_ONE),
                          PackageFour,
                          PackageFive(PACKAGEFIVE_VARIANT_A)
    }

    variant PROJECT_VARIANT_TWO
    {
        // variant specific includes
        package-includes = PackageTwo(PROJECT_VARIANT_TWO),
                          PackageThree(PROJECT_VARIANT_TWO),
                          PackageFive(PACKAGEFIVE_VARIANT_B)
    }

    // ...
}
```

Only PROJECT\_VARIANT\_ONE and PROJECT\_VARIANT\_TWO are top level variants which can be selected as active project variant.

PackageOne has no variant at all, it is included at the top level directly.

PackageTwo and PackageThree have the top level variants inside.

PackageFour does not have variants inside, it is included only for specific top level variants.

PackageFive has local variants which are specific to that package, and which cannot be selected as global variants.

PackageTwo.clf with project global variants:

```
package PackageTwo
{
```

```

05     class = PAC

    variant PROJECT_VARIANT_ONE
    {
        // variant specific includes
        package-includes = ComponentOne (PROJECT_VARIANT_ONE),
                                ComponentTwo (PROJECT_VARIANT_ONE)
    }

    variant PROJECT_VARIANT_TWO
    {
        // variant specific includes
        package-includes = ComponentOne (PROJECT_VARIANT_TWO),
                                ComponentTwo (PROJECT_VARIANT_TWO)
    }

20     // ...
    }

```

PackageFive.clf with package local variants:

```

25 package PackageFive
    {
        class = PAC

        package-includes = ComponentSeven

30     variant PACKAGEFIVE_VARIANT_A
    {
        // variant specific includes
        package-includes = ComponentEight (PACKAGEFIVE_VARIANT_A),
                                ComponentNine (PACKAGEFIVE_VARIANT_A)
35     }

    variant PACKAGEFIVE_VARIANT_B
    {
        // variant specific includes
        package-includes = ComponentEight (PACKAGEFIVE_VARIANT_B),
                                ComponentNine (PACKAGEFIVE_VARIANT_B)
40     }

    // ...
45 }

```

## 11.2.2 Naming Conventions

The following rules define naming conventions for CLF files and their structure.

### Rule CLF\_Naming\_001: Domain

**Instruction** The domain shall be used only once in the top level CLF file, and it shall have the value CUBAS.

### Rule CLF\_Naming\_002: Name uniqueness

**Instruction** The CLF file name and the containing package name shall be identical. CLF files and package names shall be unique across the whole project

The name of a CLF file and of the package in the CLF file can be used only once in a project. If a package name exists several times the CLF structure cannot be resolved properly.

## Rule CLF\_Naming\_003: Packages and components

**Instruction** The name of the CLF file shall have the name of the respective CUBAS package or component.

The same name shall be used as package name inside the CLF file.

Example: PAC Mcal

CLF Name: Mcal.clf:

```
package Mcal
{
    class = PAC

    package-includes = Adc_Stc,
                      Dio,
                      Port
                      rba_Gtm
                      // more components separated by comma
}
```

Example: COMP Dio

Dio.clf:

```
package Dio
{
    class = COMP
}
```

## Rule CLF\_Naming\_004: Structural components

**Instruction** CLF files for structural components shall have the name of the respective structural component extended by \_Stc.

Example: STCOMP Adc

CLF Name: Adc\_Stc.clf

```
package Adc_Stc
{
    class = STCOMP

    package-includes = Adc,
                      rba_IoMcuAdc
}
```

## Rule CLF\_Naming\_005: Packages and components configuration

**Instruction** CLF files for configuration shall have the name of their respective package or component, extended by \_Cfg.

Example: ECU\_CFG for PAC Mcal

CLF Name: Mcal\_Cfg.clf

```
package Mcal_Cfg
{
    class = ECU_CFG

    package-includes = Adc_Cfg,
                      Dio_Cfg,
                      // more Mcal configurations
}
```

Example: ECU\_CFG for COMP BswM

CLF Name: BswM\_Cfg.clf

```
package BswM_Cfg
{
    class = ECU_CFG
}
```

## Rule CLF\_Naming\_006: Structural components configuration

**Instruction** CLF files for configuration of structural components shall have the name of the respective structural component, extended by \_Stc\_Cfg.

Example: ECU\_CFG for STCOMP Adc

CLF Name: Adc\_Stc\_Cfg.clf

```
package Adc_Stc_Cfg
{
    class = ECU_CFG

    package-includes = Adc_Cfg,
                      rba_IoMcuAdc_Cfg
}
```

## Rule CLF\_Naming\_007: Variant Names

**Instruction** Variant names shall use capital letters and underscore as special character as delimiter for name parts.

They have to be unique and self-explanatory.

Variant names for microcontrollers have to be defined like this: <MANUFACTURER>\_<DEVICE>\_<HWPACKAGE>

This helps to differentiate variant names from CLF package names which are usually written in CamelCase.

Example: IFX\_DEV3\_LQFP

# 11.3 Examples for Variants Usage

CLF variant handling can be used to enable or disable individual files or complete folders.

## 11.3.1 Simple enabling / disabling of folders

Folders can be hidden from the project by using a variant with an ignore-rule that excludes everything inside of the component. The second variant is empty, this means the folder content is taken into account.

Example: COMP rba\_IoExtCj135

CLF Name: rba\_IoExtCj135.clf

```
package rba_IoExtCj135
{
    class = COMP

    variant USE {}

    variant IGNORE {
```



```

05         ignore-rule {
            path = **
        }
    }

```

}  
 rba\_IoExtCj135 is used in the upper level CLF, in the present example in IoExtDev rba\_IoExtCj135 is enabled by using the variant USE, and rba\_IoExtCj721 is disabled with the variant IGNORE:

```

package IoExtDev
{
    class = PAC

    // use either USE or IGNORE as variant
    package-includes = rba_IoExtCj135 (USE),
                                rba_IoExtCj721 (IGNORE)
}

```

## 11.3.2 Using configuration variants

Usually there are many configuration variants, e.g. for different CPU devices, ECUs, customer projects, etc.

To help reduce the complexity, inclusion of ready-made rules for each variant is simpler than specifying each rule in every variant in every file.

Either the complete variant is passed to the next CLF package, e.g. IFX\_DEV3\_LQFP or JDP\_DEV3\_LQFP.

Or only the manufacturer type IFX or JDP, if there is no dependency to the individual microcontroller device.

Or no variant is passed at all if the component is independent of any hardware variant.

All this will be shown in the following examples.

Let's take a snapshot of a project, and look at the following folder structure for a few packages and components:

```

\BootCtrl
\BootCtrl\rba_BootCtrl
\BSMM
\BSMM\BswM
\CUCEL
\CUCEL\Compiler
\CUCEL\ComStack
\CUCEL\EcuC
\CUCEL\Platform
\CUCEL\Standard

```

The corresponding configuration folder structure is the following:

```

\Conf
\Conf\BootCtrl
\Conf\BootCtrl\rba_BootCtrl
\Conf\BSMM
\Conf\BSMM\BswM
\Conf\CUCEL
\Conf\CUCEL\Compiler
\Conf\CUCEL\ComStack
\Conf\CUCEL\EcuC
\Conf\CUCEL\Platform
\Conf\CUCEL\Standard

```

Now for each variant there are additional folders which contain the actual configuration files, example with rba\_BootCtrl:

```
\Conf
\Conf\BootCtrl
\Conf\BootCtrl\rba_BootCtrl
\Conf\BootCtrl\rba_BootCtrl\IFX
\Conf\BootCtrl\rba_BootCtrl\IFX\IFX_Common\rba_BootCtrl_EcucValues.arxml
\Conf\BootCtrl\rba_BootCtrl\JDP
\Conf\BootCtrl\rba_BootCtrl\JDP\JDP_DEV2_LQFP_144\rba_BootCtrl_EcucValues.arxml
\Conf\BootCtrl\rba_BootCtrl\JDP\JDP_DEV3_LQFP\rba_BootCtrl_EcucValues.arxml
\Conf\BootCtrl\rba_BootCtrl\JDP\JDP_DEV4_BGA_416\rba_BootCtrl_EcucValues.arxml
```

The main configuration is defined in Conf\Conf.clf.

```
package Conf
{
    class = PAC
    ignoreUpperRules = true

    variant IFX_DEV3_LQFP
    {
        package-includes = BootCtrl_Cfg(IFX_DEV3_LQFP), // pass specific CPU or testboard variant
                        BSMM_Cfg, // no variant - independent of configured variants
                        CUCEL_Cfg(IFX) // only depends on manufacturer
    }

    variant JDP_DEV3_LQFP
    {
        package-includes = BootCtrl_Cfg(JDP_DEV3_LQFP), // pass specific CPU or testboard variant
                        BSMM_Cfg, // no variant - independent of configured variants
                        CUCEL_Cfg(JDP) // only depends on manufacturer
    }
}
```

CLF for configuration of BootCtrl: Conf\BootCtrl\BootCtrl\_Cfg.clf

```
package BootCtrl_Cfg
{
    class = ECU_CFG

    variant IFX_DEV3_LQFP
    {
        package-includes = rba_BootCtrl_Cfg(IFX_DEV3_LQFP)
    }

    variant JDP_DEV3_LQFP
    {
        package-includes = rba_BootCtrl_Cfg(JDP_DEV3_LQFP)
    }
}
```

CLF for configuration of rba\_BootCtrl: Conf\BootCtrl\rba\_BootCtrl\rba\_BootCtrl\_Cfg.clf

```
package rba_BootCtrl_Cfg
{
    class = ECU_CFG

    // rules for common folder
    package-includes = Project_Rules_Conf_Common

    // Variants supported by this component
```

```

variant IFX_DEV3_LQFP
{
    package-includes = Project_Rules_Conf_IFX_Common, Project_Rules_Conf_IFX_DEV3_LQFP
}

```

```

variant JDP_DEV3_LQFP
{
    package-includes = Project_Rules_Conf_JDP_Common, Project_Rules_Conf_JDP_DEV3_LQFP
}
}

```

CLF for configuration of BSMM: Conf\BSMM\BSMM\_Cfg.clf

```

package BSMM_Cfg
{
    class = ECU_CFG

    package-includes = BswM_Cfg
}

```

CLF for configuration of BswM: Conf\BSMM\BswM\BswM\_Cfg.clf

As there is no dependency, only the common rules are included.

```

package BswM_Cfg
{
    class = ECU_CFG

    // rules for common folder
    package-includes = Project_Rules_Conf_Common
}

```

CUCEL has only two variants inside: one for IFX, one for JDP:

Cucel\_Cfg.clf

```

package CUCEL_Cfg
{
    class = ECU_CFG

    variant IFX
    {
        package-includes = EcuC_Cfg,
                           Compiler_Cfg,
                           ComStack_Cfg,
                           Standard_Cfg,
                           Platform_Cfg(IFX)
    }

    variant JDP
    {
        package-includes = EcuC_Cfg,
                           Compiler_Cfg,
                           ComStack_Cfg,
                           Standard_Cfg,
                           Platform_Cfg(JDP)
    }
}

```

Now to the included rule packages.

Since in Conf.clf the global rules are disabled by the keyword IgnoreUpperRules = true each CLF package for configuration now has to include a specific rule set.

In the example there is one for common files (independent of any variant), one for manufacturer dependent configuration, and one for each specific variant (here: microcontroller variant).

Those rules have to reside on top level directory aside the main project.clf. Unfortunately all relevant file classes for the configuration folders have to be copied here from the global Rules.clf file.

#### Project\_Rules\_Conf\_Common.clf

```
package Project_Rules_Conf_Common
{
    rule {
        class = CONFDATA
        path = **/Common/*.arxml
        path = **/Common/*_confdata.xml
    }

    rule {
        class = SWHDR
        path = **/Common/**/*.*h
    }

    rule {
        class = SWSRC
        path = **/Common/**/*.*c
    }
}
```

#### Project\_Rules\_Conf\_IFX\_Common.clf

```
package Project_Rules_Conf_IFX_Common
{
    rule {
        class = CONFDATA
        path = **/IFX_Common/*.arxml
        path = **/IFX_Common/*_confdata.xml
    }

    rule {
        class = SWHDR
        path = **/IFX_Common/**/*.*h
    }

    rule {
        class = SWSRC
        path = **/IFX_Common/**/*.*c
    }
}
```

#### Project\_Rules\_Conf\_JFP\_Common.clf

```
package Project_Rules_Conf_JDP_Common
{
    rule {
        class = CONFDATA
        path = **/JDP_Common/*.arxml
        path = **/JDP_Common/*_confdata.xml
    }

    rule {
        class = SWHDR
        path = **/JDP_Common/**/*.*h
    }

    rule {
        class = SWSRC
    }
}
```

```
    path = **/JDP_Common/**/*.*
}
```

Finally, as example for variant specific, Project\_Rules\_Conf\_JFP\_Common.clf

```
package Project_Rules_Conf_IFX_DEV3_LQFP
{
    rule {
        class = CONFDATA
        path = **/IFX_DEV3_LQFP/*.arxml
        path = **/IFX_DEV3_LQFP/*_confdata.xml
    }

    rule {
        class = SWHDR
        path = **/IFX_DEV3_LQFP/*.h
    }

    rule {
        class = SWSRC
        path = **/IFX_DEV3_LQFP/*.c
    }
}
```

So the rule of thumb is: Project\_Rules\_Conf\_<VariantName>.clf

```
package Project_Rules_Conf_<VariantName>
{
    rule {
        class = CONFDATA
        path = **/<VariantName>/*.arxml
        path = **/<VariantName>/*_confdata.xml
    }

    rule {
        class = SWHDR
        path = **/<VariantName>/*.h
    }

    rule {
        class = SWSRC
        path = **/<VariantName>/*.c
    }
}
```

If necessary, other file classes can be added.

# A Rules summary

Table 54 General C90 Language Topics

Id	Rule <i>Chapter 3.1.1</i>
<i>[CCode_001]</i>	<b>Conformity to C Standard</b> All software written in C language shall be conform to ISO/IEC 9899: 1990 (ISO C90) "Programming languages – C" amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/-AMD1:1995, and ISO/IEC 9899/- COR2:1996. Additionally some selected deviations and extensions are permitted to ensure the applicability of C language for automotive applications.
<i>[CCode_002]</i>	<b>Character Set, Escape Sequences and Trigraphs</b> A basic set of characters and escape sequences conform to ISO C90 shall be used. Usage of trigraphs is not allowed.
<i>[CCode_003]</i>	<b>Octal Constants and Escape Sequences</b> Octal constants (other than zero) and octal escape sequences shall not be used.
<i>[CCode_004]</i>	<b>Usage and Handling of Bit Fields</b> Bit fields are allowed but shall only be defined based on "unsigned int" or "signed int" data types. Additionally a bit field based on signed type shall be at least 2 bits long. Bit fields shall not be used in APIs and inside of unions. The size of bit fields is limited to 16 bits. Finally no certain order of the bits inside the bit field shall be assumed.
<i>[CCode_005]</i>	<b>Packing of Bit Fields</b> If it is relied on, the implementation-defined behaviour and packing of bit fields shall be documented in the chapter for integration in the product documentation.

Table 55 MISRA and HIS Metrics Conformity

Id	Rule <i>Chapter 3.1.2</i>
<i>[CCode_Misra-HIS_001]</i>	<b>Conformity to MISRA Standard and HIS Metrics</b> All software modules written in C language shall conform to the accepted set of rules of the MISRA C Standard and shall be conform to the accepted set of HIS metrics.
<i>[CCode_Misra-HIS_002]</i>	<b>Commendation of MISRA Violations in C Code</b> In technically reasonable, exceptional cases MISRA violations are permissible and shall be documented with a comment within the source file.
<i>[CCode_Misra-HIS_003]</i>	<b>Commendation of MISRA Violations for a Complete Module</b> In technically reasonable, exceptional cases component wide MISRA rules violations are permissible and shall be documented in the chapter for integration in the product documentation.
<i>[CCode_Misra-HIS_004]</i>	<b>Commendation HIS Metric Violations in C Code</b> In technically reasonable, exceptional cases HIS Metrics limit violations are permissible and shall be documented with a comment within the source file.

Table 56 Initializations

Id	Rule <i>Chapter 3.1.3</i>
<i>[CCode_Inits_001]</i>	<b>Initialization of Variables</b> All variables shall have been assigned a value before they are being used.
<i>[CCode_Inits_002]</i>	<b>Initialization of Enumerators</b> In an enumerator list, the '=' construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.

Table 57 Expressions

Id	Rule <i>Chapter 3.1.4</i>
<i>[CCode_Expr_001]</i>	<b>Limited Dependence</b> Limited dependence should be placed on C's operator precedence rules in expressions.

Id	Rule <i>Chapter 3.1.4</i>
<i>[CCode_Expr_-002]</i>	<b>Value of Expressions</b> The value of an expression shall be the same under any order of evaluation that the standard permits.
<i>[CCode_Expr_-003]</i>	<b>Usage of Sizeof Operator</b> The sizeof operator shall not be used on expressions that contain side effects.
<i>[CCode_Expr_-004]</i>	<b>Logical and Conditional Operators without Side Effects</b> The right hand operand of a logical '&&' or '  ' operator and of a conditional operator '? :' shall not contain side effects.
<i>[CCode_Expr_-005]</i>	<b>Logical Operator Operands as Primary Expressions</b> The operands of a logical '&&' or '  ' operators shall be primary-expressions or extra parentheses are recommended.
<i>[CCode_Expr_-006]</i>	<b>Usage of Logical Operators</b> The operands of logical operators ('&&', '  ' and '!') should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than '&&', '  ', '!', '=', '==', '!=' and '? :'.
<i>[CCode_Expr_-007]</i>	<b>Type of Bitwise Operators</b> Bitwise operators shall not be applied to operands whose type is a signed integer.
<i>[CCode_Expr_-008]</i>	<b>Usage of Shift Operator</b> The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.
<i>[CCode_Expr_-009]</i>	<b>Usage of Unary Minus Operator</b> The unary minus operator shall not be applied to an expression whose type is an unsigned integer.
<i>[CCode_Expr_-010]</i>	<b>Usage of Comma Operator</b> The comma operator shall not be used, except in the control expression of a "for" loop and within macros.
<i>[CCode_Expr_-011]</i>	<b>Usage of Constant Unsigned Integer Expression</b> Evaluation of constant unsigned integer expressions should not lead to wrap-around.
<i>[CCode_Expr_-012]</i>	<b>Usage of Increment and Decrement Operator</b> The increment '++' and decrement '--' operators should not be mixed with other operators in an expression.
<i>[CCode_Expr_-013]</i>	<b>Value of Floating Complex Expression</b> The value of a complex expression of floating type shall only be cast to a floating type which is narrower or of the same size.
<i>[CCode_Expr_-014]</i>	<b>Implicit Integer Type Conversions</b> Implicit type conversions without a cast of integer types shall only be done from smaller to bigger types where no information is lost.
<i>[CCode_Expr_-015]</i>	<b>Explicit Integer Type Casts</b> Explicit type casts of integer types shall only be done if an explicit conversion to a narrower type is intended.

Table 58 Control Statement Expressions

Id	Rule <i>Chapter 3.1.5</i>
<i>[CCode_Control_-001]</i>	<b>Boolean Expressions</b> Assignment operators shall not be used in expressions that yield a Boolean value.
<i>[CCode_Control_-002]</i>	<b>Value Test Expression</b> Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.
<i>[CCode_Control_-003]</i>	<b>Test of Floating-Point Expressions</b> Floating-point expressions shall not be tested for equality or inequality. A check for equality or inequality to zero is allowed.

Id	Rule <i>Chapter 3.1.5</i>
<i>[CCode_Control_004]</i>	<b>For Loop Expression</b> The three expressions of a for statement shall be concerned only with loop control and shall not contain any objects of floating type.
<i>[CCode_Control_005]</i>	<b>For Loop Iteration</b> Numeric variables being used within a "for" loop for iteration counting shall not be modified in the body of the loop.

Table 59 Control Flow

Id	Rule <i>Chapter 3.1.6</i>
<i>[CCode_Cntr-Flow_001]</i>	<b>Unreachable Code</b> There shall be no unreachable code.
<i>[CCode_Cntr-Flow_002]</i>	<b>Duplication of Code</b> The duplication of code should be avoided.
<i>[CCode_Cntr-Flow_003]</i>	<b>Non-Null Statements</b> All non-null statements shall either a) have at least one side-effect however executed, or b) cause control flow to change.
<i>[CCode_Cntr-Flow_004]</i>	<b>Null Statement</b> Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.
<i>[CCode_Cntr-Flow_005]</i>	<b>Goto and Continue</b> The 'goto' and the 'continue' statements shall not be used. Additionally goto labels shall not be defined.
<i>[CCode_Cntr-Flow_006]</i>	<b>Iteration Statement</b> For any iteration statement there shall be at most one break statement used for loop termination.
<i>[CCode_Cntr-Flow_007]</i>	<b>If ... Else</b> All if ... else if constructs shall be terminated with an else clause.

Table 60 Switch Statements

Id	Rule <i>Chapter 3.1.7</i>
<i>[CCode_Switch_001]</i>	<b>Prohibition of Declarations or Definitions between Case and Default Clauses</b> The case and default clauses in the body of a switch statement shall not be preceded by declarations or definitions.
<i>[CCode_Switch_002]</i>	<b>Position of Switch Labels</b> A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.
<i>[CCode_Switch_003]</i>	<b>Usage of Break Statement</b> An unconditional break statement shall terminate every non-empty switch clause even it is not used because of optimized programming.
<i>[CCode_Switch_004]</i>	<b>Usage of Default Clauses</b> The final clause of a switch statement shall be the default clause.
<i>[CCode_Switch_005]</i>	<b>Condition for Switch Expression Values</b> A switch expression shall not represent a value that is effectively Boolean.
<i>[CCode_Switch_006]</i>	<b>Usage of Switch Statements</b> Every switch statement shall have at least one case clause.



Table 61 Structures and Unions

Id	Rule <i>Chapter 3.1.8</i>
<i>[CCode_Struct_001]</i>	<b>Usage of Unions</b> Unions are allowed but they shall not be used to access to sub-parts of objects or to pack or unpack objects. An object shall not be assigned to an overlapping object.
<i>[CCode_Struct_002]</i>	<b>Typedef Declarations for Structures and Unions</b> All structure and unions should base on specific typedef declarations.
<i>[CCode_Struct_003]</i>	<b>Completion of Structure and Union Types</b> All structure and union types shall be complete at the end of a translation unit.
<i>[CCode_Struct_004]</i>	<b>Avoidance of Alignment Gaps inside Structures</b> Elements of structures shall be arranged in that way to avoid alignment gaps.

Table 62 Preprocessing Directives

Id	Rule <i>Chapter 3.1.9</i>
<i>[CCode_Prepro_001]</i>	<b>Handling of #include Statements</b> #include statements in a file should only be preceded by other preprocessor directives or comments.
<i>[CCode_Prepro_002]</i>	<b>Prohibition of Non-Standard Characters in #include Directives</b> Non-standard characters should not occur in header file names in #include directives.
<i>[CCode_Prepro_003]</i>	<b>#include followed by a File Name</b> The #include directive shall be followed by either a <filename> or "filename" sequence.
<i>[CCode_Prepro_004]</i>	<b>Prohibition of #undef</b> #undef shall not be used.
<i>[CCode_Prepro_005]</i>	<b>Handling of Preprocessor Operator "defined"</b> The "defined" preprocessor operator shall only be used in one of the two standard forms.
<i>[CCode_Prepro_006]</i>	<b>Correctness of Preprocessing Directives</b> Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.
<i>[CCode_Prepro_007]</i>	<b>Hold Preprocessor Directives together</b> All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
<i>[CCode_Prepro_008]</i>	<b>Macro Identifiers in Preprocessor Directives</b> All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.

Table 63 Macros

Id	Rule <i>Chapter 3.1.10</i>
<i>[CCode_Macro_001]</i>	<b>Handling of C Macros</b> C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.
<i>[CCode_Macro_002]</i>	<b>Validity of C Macros</b> Macros shall not be #define'd or #undef'd within a block.
<i>[CCode_Macro_003]</i>	<b>Handling of Function-like Macros</b> In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##.
<i>[CCode_Macro_004]</i>	<b>Calling of Function-like Macros</b> A function-like macro shall not be invoked without all of its arguments.
<i>[CCode_Macro_005]</i>	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.

Table 64 Main Rule

Id	Rule <i>Chapter 3.2.1</i>
<i>[Abstr_Main_001]</i>	<b>Main Rule of Compiler Abstraction</b> The source code of software modules (at least above the $\mu$ C Abstraction Layer (MCAL)) shall be neither processor-dependent nor compiler-dependent. No reliance shall be placed on undefined or unspecified behaviour of the compiler.

Table 65 Prohibition of Compiler Specifics

Id	Rule <i>Chapter 3.2.2</i>
<i>[Abstr_Main_002]</i>	<b>Prohibition of Compiler Specific Headers and Libraries</b> Compiler specific header files, libraries and intrinsic functions shall not be used.
<i>[Abstr_Main_003]</i>	<b>Prohibition of Compiler Specific Keywords</b> Compiler specific keywords and internal defines shall not be used.
<i>[Abstr_Main_004]</i>	<b>Prohibition of Redefinition of Reserved Identifiers, Macros and Functions</b> Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined. The names of standard library macros, objects and functions shall not be reused.

Table 66 Abstraction of Addressing Keywords

Id	Rule <i>Chapter 3.2.3 [Abstr_AddrKeywords]</i>
<i>[Abstr_NearFar_001]</i>	<b>Prohibition of Usage of Addressing Keywords</b> Direct use of compiler and platform specific keywords for addressing of data and code like "_near" or "_far" is not allowed.
<i>[Abstr_NearFar_002]</i>	<b>AUTOSAR 16 bit Microcontroller Abstraction</b> To encapsulate 16 bit microcontroller the AUTOSAR standardized #defines shall be used.
<i>[Abstr_NearFar_003]</i>	<b>Usage of Module Name of near/far Addressing Keywords</b> For declarations the module name of the defining module has to be used for the near/far addressing keywords.

Table 67 Abstraction of Memory Mapping

Id	Rule <i>Chapter 3.2.4 [Abstr_MemMap]</i>
<i>[Abstr_MemMap_001]</i>	<b>Usage of Memory Mapping Concept instead of #pragma Keywords</b> Direct use of compiler and platform specific "#pragma" keywords is not allowed. Memory mapping of code, variables and constants shall be done by using the memory mapping concept.
<i>[Abstr_MemMap_002]</i>	<b>Structure of Memory Mapping Concept</b> Declarations and definitions of code, variables and constants shall be wrapped with a start symbol (syntax: <MODULE> START_SEC <NAME>) and stop symbol (syntax: <MODULE> STOP_SEC <NAME>) and includes of memory mapping headers (" <i>&lt;Mip&gt;</i> _MemMap.h" for BSW modules and "<SW-C>_MemMap.h" for ASW components).
<i>[Abstr_MemMap_003]</i>	<b>Exception of Memory Mapping Concept: Function Local Variables</b> For function local variables within a C function no memory mapping is possible therefore memory mapping concept shall not be used. Memory mapping header files shall not be included inside the body of a function.
<i>[Abstr_MemMap_005]</i>	<b>No Usage inside Function Bodies</b> Memory mapping header files shall not be included inside the body of a function.
<i>[Abstr_MemMap_006]</i>	<b>Usage of Module Name of Memory Mapping Keywords</b> For declarations the module name of the defining module has to be used for the memory mapping keywords.

Table 68 Abstraction of Inline Functions

Id	Rule <i>Chapter 3.2.5</i>
<i>[Abstr_Inline_001]</i>	<b>Usage of LOCAL_INLINE</b> Direct use of compiler and platform specific inline keywords like "__inline__" or "_inline" are not allowed. To define an inline function macro "LOCAL_INLINE" shall be used.
<i>[Abstr_Inline_002]</i>	<b>Inlines Without RTE APIs</b> Within inline functions no RTE APIs shall be called.

Table 69 Handling of Assembler Instructions

Id	Rule <i>Chapter 3.2.6</i>
<i>[Abstr_Asm_001]</i>	<b>Usage of Assembler Code</b> Assembler instructions inside C code shall not be used unless they are encapsulated and isolated for special cases.

Table 70 Prohibition of Dynamic Memory

Id	Rule <i>Chapter 3.2.7</i>
<i>[Abstr_DynMem_001]</i>	<b>Prohibition of Dynamic Heap Memory</b> Dynamic heap memory allocation shall not be used.
<i>[Abstr_DynMem_002]</i>	<b>Usage of Memory Areas</b> An area of memory shall not be reused for unrelated purposes.

Table 71 Pointer Type Conversions

Id	Rule <i>Chapter 3.2.8</i>
<i>[Abstr_PtrConv_001]</i>	<b>Prohibited Function Pointer Conversions</b> Conversions shall not be performed between a pointer to a function and any type other than an integral type.
<i>[Abstr_PtrConv_002]</i>	<b>Prohibited Object Pointer Conversions</b> Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.
<i>[Abstr_PtrConv_003]</i>	<b>Prohibition of Casts between Pointer and Integral Types</b> A cast should not be performed between a pointer type and an integral type.
<i>[Abstr_PtrConv_004]</i>	<b>Prohibition of Casts between Object Pointer Types</b> A cast should not be performed between a pointer to object type and a different pointer to object type. Sole exception is a cast to an uint8 object type because uint8 has the smallest alignment.
<i>[Abstr_PtrConv_005]</i>	<b>Preservation of Pointer Qualifications</b> A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.

Table 72 Pointer Arithmetic and Arrays

Id	Rule <i>Chapter 3.2.9</i>
<i>[Abstr_PtrArith_001]</i>	<b>Allowed Form of Pointer Arithmetic</b> Pointer arithmetic shall only be applied to pointers that address an array or array element.
<i>[Abstr_PtrArith_002]</i>	<b>Allowed Form of Pointer Subtraction</b> Pointer subtraction shall only be applied to pointers that address elements of the same array.
<i>[Abstr_PtrArith_003]</i>	<b>Allowed Form of Comparison of Pointers</b> >, >=, <, <= shall not be applied to pointer types except where they point to the same array.
<i>[Abstr_PtrArith_004]</i>	<b>Validity of Pointer Objects</b> The address of an object with automatic storage shall not be assigned to another object that may be used after the first object has ceased to exist.

Id	Rule <i>Chapter 3.2.9</i>
<i>[Abstr_PtrArith_005]</i>	<b>Maximum Number of Pointer Indirection</b> The declaration of objects should contain no more than 2 levels of pointer indirection.

Table 73 Ensure Usability of BSW Modules in C++ Environments

Id	Rule <i>Chapter 3.2.10</i>
<i>[Abstr_CPP_002]</i>	<b>Type of Character Literal</b> Externally usable macros and inline functions shall not depend on the type of character literal.
<i>[Abstr_CPP_003]</i>	<b>Use Cast to Assign String Literals</b> Externally usable macros and inline functions shall use a cast to non-const type when a string literal is assigned to a pointer.
<i>[Abstr_CPP_004]</i>	<b>Don't use Doubled Definition</b> Externally usable macros and inline functions shall not use repeated type definitions.
<i>[Abstr_CPP_004]</i>	<b>Don't Refer to Structs Defined in Structs from Outside</b> Externally usable macros and inline functions shall not refer to structs, enumerations or enumerator names outside the struct in which those are defined.
<i>[Abstr_CPP_006]</i>	<b>Declare const Objects with External Linkage in C code Explicitly extern</b> Externally usable macros and inline functions shall declare <i>const</i> objects with external linkage in C code explicitly <i>extern</i> .
<i>[Abstr_CPP_007]</i>	<b>Use Cast to Convert void* to a Pointer Type</b> Externally usable macros and inline functions shall use a cast to convert a void* to a pointer type.
<i>[Abstr_CPP_008]</i>	<b>Only Pointers to non-const and non-volatile Objects may be Implicitly Converted to void*</b> Within externally usable macros and inline functions only pointers to non-const and non-volatile objects may be implicitly converted to void*.
<i>[Abstr_CPP_009]</i>	<b>const Objects shall be Initialized</b> Externally usable macros and inline functions shall initialize const objects.
<i>[Abstr_CPP_010]</i>	<b>Don't Use auto as a Storage Class Specifier</b> Externally usable macros and inline functions shall not use auto as a storage class specifier.
<i>[Abstr_CPP_011]</i>	<b>Assign only Values of the same Type to Enumeration Objects</b> Within externally usable macros and inline functions only values of the same type shall be assigned to enumeration objects.
<i>[Abstr_CPP_012]</i>	<b>Don't let the Code Depend on the Type and Size of Enumeration Objects</b> Within externally usable macros and inline functions code shall not depend on the type and size of enumeration objects.
<i>[Abstr_CPP_013]</i>	<b>Use a Cast or Separate Implementations to Copy Volatile Structs</b> Externally usable macros and inline functions code shall use a cast when copying volatile structs to non-volatile structs.
<i>[Abstr_CPP_014]</i>	<b>Don't use __STDC__</b> Within externally usable macros and inline functions the value of __STDC__ shall not be used.
<i>[Abstr_CPP_015]</i>	<b>Encapsulate C code Conflicting with C++ and Provide C++ Substitution Code</b> If usage of C code conflicting with C++ cannot be avoided then that conflicting code shall be encapsulated and C++ compliant code shall also be provided.

Id	Rule <i>Chapter 3.2.10</i>
[Rule Abstr_CPP_-001]	<p><b>Don't use C++ Keywords not Defined as Keywords in C</b></p> <p>C++ defines new keywords in addition to C. Externally usable macros and inline functions shall not use those keywords. The respective keywords are:</p> <ul style="list-style-type: none"> <li>▶ alignas alignof and and_eq asm</li> <li>▶ bitand bitor bool</li> <li>▶ catch char16_t char32_t class compl constexpr const_cast</li> <li>▶ decltype delete dynamic_cast</li> <li>▶ explicit export</li> <li>▶ false friend</li> <li>▶ inline</li> <li>▶ mutable</li> <li>▶ namespace new noexcept not not_eq nullptr</li> <li>▶ operator or or_eq</li> <li>▶ private protected public</li> <li>▶ reinterpret_cast</li> <li>▶ static_assert static_cast</li> <li>▶ template this thread_local throw true try typeid typename</li> <li>▶ using</li> <li>▶ virtual</li> <li>▶ wchar_t</li> <li>▶ xor xor_eq</li> </ul>

Table 74 Prohibition of Open Source Software

Id	Rule <i>Chapter 3.2.11</i>
[Abstr_OSS_001]	<p><b>Prohibition of Open Source Software</b></p> <p>Open Source Software (OSS) shall not be used within BSW modules.</p>

Table 75 Base Integer and Float Data Types

Id	Rule <i>Chapter 3.3.1</i>
[CCode_Types_-001]	<p><b>Platform Integer and Float Data Types</b></p> <p>Following common platform integer and float data types are allowed and can be used within SW and APIs: boolean, uint8, sint8, uint16, sint16, uint32, sint32, float32, float64.</p> <p>The float data types shall be used carefully and their usability shall be ensured.</p>
[CCode_Types_-002]	<p><b>Prohibition of Plain C Data Types</b></p> <p>The usage of plain C data types "int", "short" and "long" is forbidden.</p>
[CCode_Types_-003]	<p><b>Usage of Char Data Type</b></p> <p>The plain "char" data type shall only be used for the storage and use of character values or data.</p>
[CCode_Types_-004]	<p><b>Handling of 64 Bit Data Types</b></p> <p>64 bit integer data types are no base data types and shall not be used in APIs. A local definition is allowed (e.g. libraries) but shall only used in exceptional cases.</p>

Id	Rule <i>Chapter 3.3.1</i>
<i>[CCode_Types_005]</i>	<b>Handling of Own Defined Data Types</b> Do not define own data types based on base data types if this is not necessary and the data width is known at specification time.

Table 76 Optimized Integer Data Types

Id	Rule <i>Chapter 3.3.2</i>
<i>[CCode_Types_006]</i>	<b>Common Optimized Integer Data Types</b> Following common optimized integer data types are allowed and can be used in a local scope inside a module but not in public APIs: uint8_least, sint8_least, uint16_least, sint16_least, uint32_least, sint32_least.
<i>[CCode_Types_007]</i>	<b>Handling of Optimized Integer Data Types</b> Operations on the optimized integer data types (<typename>_least) shall not expect a specific size of this type. The size specified by the name is guaranteed, but can be larger. It is not allowed to use rollover mechanisms during counting and shifting.

Table 77 Standard Symbols

Id	Rule <i>Chapter 3.3.3</i>
<i>[CCode_Symbols_001]</i>	<b>TRUE and FALSE</b> TRUE is defined as "1" and FALSE is defined as "0" can be used in conjunction with the standard data type "boolean" and should not be redefined.
<i>[CCode_Symbols_002]</i>	<b>CPU Specific Symbols</b> Following CPU specific common symbols are available: CPU_TYPE and CPU_BYTE_ORDER.
<i>[CCode_Symbols_003]</i>	<b>Common Symbols</b> Following other common symbols are usable: E_OK, E_NOT_OK, STD_HIGH, STD_LOW, STD_ACTIVE, STD_IDLE, STD_ON, STD_OFF.
<i>[CCode_Symbols_004]</i>	<b>Standard Version Info Type</b> Type "Std_VersionInfoType" shall be used to request the version of a BSW module.
<i>[CCode_Symbols_005]</i>	<b>Usage of Null Pointer</b> For null pointers "NULL_PTR" shall be used.

Table 78 Specific Types and Symbols for Communication Software

Id	Rule <i>Chapter 3.3.4</i>
<i>[CCode_ComStackTypes_001]</i>	<b>Type for PDU Identifiers</b> Type "PduldType" has to be used to define variables for unique identifiers for PDUs.
<i>[CCode_ComStackTypes_002]</i>	<b>Handling of Types for PDU Identifiers</b> Variables of type "PduldType" shall be zero-based and consecutive, in order to be able to perform table-indexing within a software module.
<i>[CCode_ComStackTypes_003]</i>	<b>Type for Length Information of a PDU</b> Type "PduLengthType" shall be used to define length information of a PDU which is provided in number of bytes.
<i>[CCode_ComStackTypes_004]</i>	<b>Type for Basic Information of a PDU</b> Type "PduInfoType" has to be used to store the basic information about a PDU.
<i>[CCode_ComStackTypes_005]</i>	<b>Type for Result of a Buffer Request</b> Type "BufReq_ReturnType" shall be used to define variables to store the result of a buffer request.
<i>[CCode_ComStackTypes_006]</i>	<b>Type for Result Status of a Notification</b> Type "NotifResultType" shall be used to define variables to store the result status of a notification (confirmation or indication).

Id	Rule <i>Chapter 3.3.4</i>
<i>[CCode_Com-StackTypes_007]</i>	<b>Naming of Return Codes of a Notification</b> Return codes of a notification shall be named as follows: NTFRSLT_E_<Communication System Abbreviation>_<Error Code Name>.
<i>[CCode_Com-StackTypes_008]</i>	<b>Type for Bus Status Evaluated by a Transceiver</b> Type "BusTrcvErrorType" shall be used to define variables to return the bus status evaluated by a transceiver.
<i>[CCode_Com-StackTypes_009]</i>	<b>Naming of Return Codes of a Bus Status Notification</b> Return codes of a bus status notification shall be named as follows: BUSTRCV_E_<Communication System Abbreviation>_<Error Code Name>.
<i>[CCode_Com-StackTypes_010]</i>	<b>Type for State of a Transport Protocol Buffer</b> Type "TpDataStateType" shall be used to define variables to store the state of a Transport Protocol (TP) buffer.
<i>[CCode_Com-StackTypes_011]</i>	<b>Type for Transport Protocol Buffer Handling</b> Type "RetryInfoType" shall be used to define variables to store the information about TP buffer handling.
<i>[CCode_Com-StackTypes_012]</i>	<b>Type for Identifiers of Communication Channels</b> Type "NetworkHandleType" shall be used to define variables to store the identifier of a communication channel.
<i>[CCode_Com-StackTypes_013]</i>	<b>Type to Specify a Parameter</b> Type "TpParameterType" shall be used in "ChangeParameter" interfaces to specify the parameter to which the value has to be changed.

Table 79 Module Specific Add Ons

Id	Rule <i>Chapter 3.3.5</i>
<i>[SpecificTypes_5]</i>	<b>Handling of Additional Types and Symbols</b> Module specific symbols may be defined and used if a module needs more symbols than are provided by standard symbols or communication software specific symbols.

Table 80 Rule Set: Naming Convention

Id	Rule <i>Chapter 3.4</i>
<i>[CDGNaming_001]</i>	<b>Definition of a Component Prefix</b> Every name with a global visibility shall be prefixed with a component prefix.
<i>[CDGNaming_002]</i>	<b>Macros</b> Macro names shall be defined in following form: <COMPONENT PREFIX>{_IDENTIFIER}_<UPPERTEXT>.
<i>[CDGNaming_003]</i>	<b>C-Identifiers with File Scope</b> C-identifiers with file scope shall be defined in following form: <Component prefix>{_identifier}_<pp><DescriptiveText>{_Typesuffix}.
<i>[CDGNaming_004]</i>	<b>Typedefs</b> A typedef name shall be a unique identifier.
<i>[CDGNaming_005]</i>	<b>Tag Names</b> A tag name shall be a unique identifier.
<i>[CDGNaming_006]</i>	<b>C-Identifiers with Block Scope</b> C-identifiers with block scope or function prototype scope shall be defined in following form: {s_}<pp><DescriptiveText>_<Typesuffix>.
<i>[CDGNaming_007]</i>	<b>Usage of Object of Function Identifier</b> No object or function identifier with static storage duration should be reused.
<i>[CDGNaming_008]</i>	<b>Spelling of Identifiers in Different Name Spaces</b> No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.



Id	Rule <i>Chapter 3.4</i>
<i>[CDGNaming_009]</i>	<b>Identifiers of Inner and Outer Scope</b> Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.
<i>[CDGNaming_010]</i>	<b>Maximum Length of Identifiers</b> Identifiers (internal and external) shall not rely on the significance of more than 60 characters.
<i>[CDGNaming_011]</i>	<b>File and Directory Names</b> File and directory names shall be defined in a specified form.
<i>[CDGNaming_012]</i>	<b>SymbolicNameValue</b> The values of configuration parameters which are defined as symblicNameValue = true shall be generated into the header file of the declaring module as #define. The symbol shall be composed of <ul style="list-style-type: none"> <li>▶ the component prefix of the declaring component followed directly by the literal "Conf_" followed by</li> <li>▶ the shortName of the EcucParamConfContainerDef of the declaring module followed by "_" followed by</li> <li>▶ the shortName of the EcucContainerValue container which holds the symblicNameValue configuration parameter value.</li> </ul>

Table 81 Basis Set of Module Files

Id	Rule <i>Chapter 3.5.1</i>
<i>[BSW_Files_001]</i>	<b>Basic Set Of Module Files</b> All modules shall provide at least the following files: "<Module>.c", "<Module>.h" and "<Module>_BSWMD.arxml". Other files have to be provided if they are needed.
<i>[BSW_Files_002]</i>	<b>Additional Module C Files</b> If a module provides several functions and processes additional module source files "<Module>_<Sub>.c" may be used. Name <Sub> can be chosen freely.
<i>[BSW_Files_003]</i>	<b>Header for Callback Functions</b> Declarations of callback functions shall be grouped and out-sourced in a separate header file "<Module>_Cbk.h".
<i>[BSW_Files_004]</i>	<b>Files for ECU Configuration</b> Configuration parts shall strictly be separated from implementation of functionality. Configuration data (not to be modified after compile time) shall be grouped and out-sourced to following configuration files: "<Module>_Cfg.c", "<Module>_Cfg_<Sub>.c", "<Module>_Cfg.h" and "<Module>_Cfg_<Sub>.h" for pre-compile configuration data, "<Module>_Lcfg.c", "<Module>_Lcfg_<Sub>.c", "<Module>_Lcfg.h" and "<Module>_Lcfg_<Sub>.h" for link time configuration data and "<Module>_PBcfg.c", "<Module>_PBcfg_<Sub>.c", "<Module>_PBcfg.h" and "<Module>_PBcfg_<Sub>.h" for post build configuration data.
<i>[BSW_Files_005]</i>	<b>Header for Module Specific Types</b> Module specific types and symbols can be defined in header "<Module>_Types.h".
<i>[BSW_Files_006]</i>	<b>Header for Exclusive Interfaces for another Module</b> Interfaces which are provided exclusively for one module should be separated into a dedicated header file "<Module>_<User>.h".
<i>[BSW_Files_007]</i>	<b>Additional Module Headers</b> To structure headers of a module additional sub header(s) can be used: "<Module>_<Sub>.h" and "<Module>_<Sub>_Inl.h" to structure "<Module>.h" and "<Module>_Prv_<Sub>.h" and "<Module>_Prv_<Sub>_Inl.h" to structure "<Module>_Prv.h". Name <Sub> can be chosen freely.
<i>[BSW_Files_008]</i>	<b>Private Module Header</b> Elements which shall not be exported via module header file "<Module>.h" shall be placed in private header "<Module>_Prv.h".



Table 82 Header Include Concept

Id	Rule <i>Chapter 3.5.2</i>
<i>[BSW_HeaderInc_001]</i>	<b>Definition of BSW Header Include Concept</b> Following header include concept has to be followed.
<i>[BSW_HeaderInc_002]</i>	<b>Include Order of Module Header</b> "<Module>.h" header has to be included as first header in every c file (functional and configuration) of a module.
<i>[BSW_HeaderInc_004]</i>	<b>Include Order of Types Header inside Module Header</b> Header for standard base data types and symbols shall be included in "<Module>.h" as first header. Communication related modules include "ComStack_Types.h", all other non communication related modules include "Std_Types.h".
<i>[BSW_HeaderInc_005]</i>	<b>Include Order of other Headers inside Module Header</b> Further preferred include order of headers in module header file: Header for module specific types ("<Module>_Types.h"), optional RTE generated header ("Rte_<Module>.h"), optional external module headers ("<Extmodule>.h", "<Extmodule>_Cbk.h", "<Extmodule>_<Module>.h"), configuration header ("<Module>_Cfg.h", "<Module>_Cfg_<Sub>.h"), optional other configuration headers ("<Module>_Lcfg.h", "<Module>_Lcfg_<Sub>.h", "<Module>_PBcfg.h", "<Module>_PBcfg_<Sub>.h"), finally structural sub headers ("<Module>_Sub.h", "<Module>_Sub_Inl.h").  Only that headers shall be included which shall be exported with the module header. The include order can be changed if the visibility and dependency of contents of headers needs another include order than the preferred one.
<i>[BSW_HeaderInc_006]</i>	<b>Include Order of Private Headers</b> Include order of headers in private module header file "<Module>_Prv.h" is: "<Module>_Prv_<Sub>.-h" for structural private sub headers of the module and "<Module>_Prv_<Sub>_Inl.h" for structural private headers containing inline functions.
<i>[BSW_HeaderInc_007]</i>	<b>Module Export Headers</b> "<Module>.h", "<Module>_Cbk.h" and "<Module>_<User>.h" are module export headers to be included in other modules. These headers shall only export that kind of information which is explicitly needed by other modules.
<i>[BSW_HeaderInc_008]</i>	<b>Import of Headers from Other Modules</b> A module shall only import the necessary header files from other modules which are required to fulfill the modules functional requirements ("<Extmodule>.h" and/or "<Extmodule>_Cbk.h" and/or "<Extmodule>_<Module>.h").
<i>[BSW_HeaderInc_009]</i>	<b>Protection against Multiple Inclusion</b> <b>Scope:</b> All headers except MemMap headers Each header file shall protect itself against multiple inclusion.
<i>[BSW_HeaderInc_010]</i>	<b>Preprocessor Check for AUTOSAR Headers</b> A pre-processor check shall be performed for all included AUTOSAR based header files (Inter Module Check).
<i>[Rule BSW_HeaderInc_003]</i>	<b>Include Order after Module Header</b> Further preferred include order of headers in module c files: BSW scheduler header ("SchM_<Module>.h"), module callback header ("<Module>_Cbk.h"), friends header ("<Module>_<User>.h"), RTE generated header ("Rte_<Module>.h"), external module headers ("<Extmodule>.h", "<Extmodule>_Cbk.h", "<Extmodule>_<Module>.h"), configuration header files ("<Module>_Lcfg.h", "<Module>_Lcfg_<Sub>.h", "<Module>_PBcfg.h", "<Module>_PBcfg_<Sub>.h") private header file ("<Module>_Prv.h").  Only that headers shall be included which are needed from the corresponding module c file. The include order can be changed if the visibility and dependency of contents of headers needs another include order than the preferred one.

Table 83 BSW Service Module with and without RTE

Id	Rule <i>Chapter 3.5.3</i>
[BSW_ServiceRTE_001]	<p><b>BSW Service Module with and without RTE: Conditions for RTE headers</b></p> <p><b>Scope:</b> BSW service modules which provide a SWCD file and have a close connection to the RTE generator</p> <p>To support that a BSW service module can work with and without a RTE generator following RTE specific files shall be provided with following content:</p> <ul style="list-style-type: none"> <li>▶ For the header "Rte_&lt;Module&gt;_Type.h" a header template "Rte_&lt;Module&gt;_Type.h_tpl" shall be provided including all the types and macros required from the BSW service module (all elements which are specified in the SWCD file)</li> <li>▶ For the header "Rte_&lt;Module&gt;.h" also a header template "Rte_&lt;Module&gt;.h_tpl" shall be provided containing only the include of "Rte_&lt;Module&gt;_Type.h", but nothing more</li> <li>▶ The "&lt;Module&gt;_SWCD.arxml" file shall be provided even if no RTE is available, but in case of an existing RTE generator this file is anyway mandatory</li> </ul>
[Rule BSW_ServiceRTE_002]	<p><b>BSW Service Module with and without RTE: Conditions for Module Header</b></p> <p><b>Scope:</b> BSW service modules which provide a SWCD file and have a close connection to the RTE generator</p> <p>To support that a BSW service module can work with and without a RTE generator the module header / the module types header shall fulfill following conditions:</p> <ul style="list-style-type: none"> <li>▶ The module header file "&lt;Module&gt;.h" and the header file "&lt;Module&gt;_Types.h" (this is only an optional header) shall contain only that elements which are needed from the BSW service module itself and which are not part of the RTE based headers</li> <li>▶ Function prototypes shall be placed in "&lt;Module&gt;.h" even if they are also generated in "Rte_&lt;Module&gt;.h". Here a crosscheck in the "&lt;Module&gt;{&lt;Sub&gt;}.c" file is intended.</li> <li>▶ The module header file "&lt;Module&gt;.h" shall not include the header "Rte_&lt;Module&gt;.h"</li> <li>▶ The module header file "&lt;Module&gt;.h" shall include the header "Rte_&lt;Module&gt;_Type.h"</li> <li>▶ The module c file "&lt;Module&gt;{&lt;Sub&gt;}.c" shall include "&lt;Module&gt;.h" and "Rte_&lt;Module&gt;.h"</li> </ul>

Table 84 Design of Processes and Interrupt Service Routines

Id	Rule <i>Chapter 3.5.5</i>
[BSW_ProcISR_001]	<p><b>Return Value of Initialization Processes</b></p> <p>The return type of initialization processes shall be void.</p>
[BSW_ProcISR_002]	<p><b>Parameters of Initialization Processes</b></p> <p>If post build selectable configuration is enabled for a module, the initialization processes shall have a pointer to it as parameter, otherwise there shall be no parameter.</p>
[BSW_ProcISR_003]	<p><b>Interface of Scheduled Processes</b></p> <p>Scheduled processes shall have no parameters and no return value.</p>
[BSW_ProcISR_004]	<p><b>Internal Design of Scheduled Processes</b></p> <p>Scheduled processes shall be designed to run in parallel with APIs and ISRs on other cores.</p>
[BSW_ProcISR_005]	<p><b>Interface of Interrupt Service Routines</b></p> <p>Interrupt Service Routines shall have no parameters and no return value.</p>
[BSW_ProcISR_006]	<p><b>Internal Design of Interrupt Service Routines</b></p> <p>Interrupt Service Routines shall be executable on any core.</p>

Table 85 Definition and Declaration of Functions and Objects

Id	Rule <i>Chapter 3.5.6</i>
<i>[BSW_FuncObj_001]</i>	<b>Definitions only in C Files</b> There shall be no definitions of objects or functions (except inline functions) in a header file. Definitions shall take place in a c file.
<i>[BSW_FuncObj_002]</i>	<b>Explicit Functions and Objects</b> Whenever an object or function is declared or defined, its object or return type shall be explicitly stated. Functions with no explicit return type shall be defined as void function.
<i>[BSW_FuncObj_003]</i>	<b>Void Functions</b> Function with no parameters shall be declared and defined with the parameter list void.
<i>[BSW_FuncObj_004]</i>	<b>Singulary Definitions of Objects and Functions</b> An object or function with external linkage or global visibility shall have exactly one definition in one file.
<i>[BSW_FuncObj_005]</i>	<b>Scope of Functions</b> Functions shall be declared at file scope and not on block scope.
<i>[BSW_FuncObj_006]</i>	<b>Prototype Declarations</b> External functions, inline functions and external objects shall have prototype declarations in one and only one header file. The prototypes shall be visible 1st at the function and object definition and 2nd the function call or object use. Static functions or static objects shall not have a prototype declaration in a header file. Static functions or static objects do not need a prototype.
<i>[BSW_FuncObj_007]</i>	<b>Equality of Declaration and Definition of a Function</b> For each function parameter the type given in the declaration and definition shall be identical, and the return types shall be identical too. Additional parameter names shall be given for all parameters in the function prototype declaration and they shall be identical to the parameter names of the function definition.
<i>[BSW_FuncObj_008]</i>	<b>Static Objects and Functions</b> All declarations and definitions of objects or functions at file scope shall have internal linkage using the static storage class specifier unless external linkage is required.
<i>[BSW_FuncObj_009]</i>	<b>Constant Pointer Parameters</b> A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.
<i>[BSW_FuncObj_010]</i>	<b>Compatible Types of multiple declared Objects of Functions</b> If objects or functions are declared more than once their types shall be compatible.
<i>[BSW_FuncObj_011]</i>	<b>Global Constants</b> Global visible constant data shall be indicated with read-only purposes by explicitly assigning the const keyword.

Table 86 Providing of Version Information

Id	Rule <i>Chapter 3.5.8</i>
<i>[BSW_VersionInfo_001]</i>	<b>Provision Version Information</b> <b>Scope:</b> Software based on an AUTOSAR SWS Each AUTOSAR based BSW module shall provide information to identify vendor, module and SW version.
<i>[BSW_VersionInfo_002]</i>	<b>Module Vendor Identifier</b> <b>Scope:</b> Software based on an AUTOSAR SWS A module vendor identifier "<MODULE>_VENDOR_ID" shall be provided in the module header file.
<i>[BSW_VersionInfo_003]</i>	<b>Module Identifier</b> <b>Scope:</b> Software based on an AUTOSAR SWS A module identifier "<MODULE>_MODULE_ID" shall be provided in the module header file.

Id	Rule <i>Chapter 3.5.8</i>
<i>[BSW_VersionInfo_004]</i>	<b>Software Version Number and AUTOSAR Specification Version Number</b> <b>Scope:</b> Software based on an AUTOSAR SWS A software version number and an AUTOSAR specification version number shall be provided in the module header file. The software version number shall be updated even before a new version of the software will be released. The AUTOSAR specification version number shall be updated even if the module supports a new version of an AUTOSAR specification.
<i>[BSW_VersionInfo_005]</i>	<b>Provision of GetVersionInfo Interface</b> <b>Scope:</b> Software based on an AUTOSAR SWS Each AUTOSAR based BSW module shall provide a function <code>&lt;Module&gt;_GetVersionInfo</code> to read out published parameters.
<i>[BSW_VersionInfo_006]</i>	<b>Usage of GetVersionInfo Interface</b> The function <code>&lt;Module&gt;_GetVersionInfo</code> shall not be called within any SW of an <i>ECU</i> (maybe to check if another module exists or to get the version of another module). These functions are only provided to be called from offline tools.

Table 87 BSW Scheduler and Exclusive Areas

Id	Rule <i>Chapter 3.5.9</i>
<i>[BSW_Sched_001]</i>	Direct access of interrupt suspend/resume functions is not allowed. Instead the AUTOSAR Standard Interfaces <code>SchM_Enter_&lt;Module&gt;_&lt;Name&gt;</code> and <code>SchM_Exit_&lt;Module&gt;_&lt;Name&gt;</code> shall be used.

Table 88 Common File Style

Id	Rule <i>Chapter 3.6.1</i>
<i>[CCode_Style_001]</i>	<b>Unique Style</b> Each C file, header and script source file shall follow common settings to get an unique style.
<i>[CCode_Style_002]</i>	<b>Standard File Header</b> Every C and header source file shall be headed by a standard file header. Excluded are generated C and header files which are not stored in the SCM.
<i>[CCode_Style_003]</i>	<b>Special Comment Blocks</b> To structure source files special comment blocks may be used to cluster header includes, definitions, declaration and code.
<i>[CCode_Style_004]</i>	<b>Line Length</b> The line length shall not exceed 120 characters.
<i>[CCode_Style_005]</i>	<b>Indentation</b> Each new instruction block shall be indented by 4 whitespaces within C and header files. The "Tab" character (ASCII code 09) is not permitted in the code and in scripts. Instead of a tabulator 4 whitespaces have to be used.
<i>[CCode_Style_006]</i>	<b>Definition of Variables</b> Only one variable shall be defined in a single line. A multi line or comma separated definition is not allowed.
<i>[CCode_Style_007]</i>	<b>Initialization of Arrays and Structures</b> Braces shall be used to indicate and match the structure in a non-zero initialization of arrays and structures.
<i>[CCode_Style_008]</i>	<b>Operators and Whitespaces</b> Mathematical, logical, comparison and conditional operators shall be embedded in whitespaces. This rule does not apply to brackets and operators like <code>!</code> , <code>++</code> and <code>--</code> .
<i>[CCode_Style_009]</i>	<b>Nested Preprocessor Commands</b> Nested preprocessor commands should use a special form of indentation.

Id	Rule <i>Chapter 3.6.1</i>
<i>[CCode_Style_010]</i>	<b>Display History Information</b> To display history information from the SCM system a special comment block shall be placed at end of a file.
<i>[CCode_Style_011]</i>	<b>Display Header File Name</b> The name of a header file should be named within a comment at the end of each header file.
<i>[CCode_Style_012]</i>	<b>New Line at End of File</b> Each c file and header shall end with a new line to avoid compiler warnings.
<i>[CCode_Style_013]</i>	<b>Avoiding of trailing spaces</b> Trailing spaces shall be avoided.

Table 89 Block Style

Id	Rule <i>Chapter 3.6.2</i>
<i>[CCode_BlockStyle_001]</i>	<b>Common Block Style Layout</b> A block style layout shall be used for all functions and type definitions and all instruction blocks like for-loop, if-else, do-while, while and switch-case. The statement forming the body of an instruction shall be a compound statement.
<i>[CCode_BlockStyle_002]</i>	<b>Block Brackets are Single Characters in a Line</b> The beginning of a block "{" and the end of a block "}" shall be at a new line and shall be the only character in this line (Except the character indicates the end of a structure definition or is followed by a while command. Here the name of the structure or the while command can stand after the end of block bracket. Additional comments are allowed.) After a block a new line should be entered (Except the block ends before an else or break command.).
<i>[CCode_BlockStyle_003]</i>	<b>Horizontal Position of Brackets of Blocks</b> "{" and "}" shall be placed at the same horizontal position.
<i>[CCode_BlockStyle_004]</i>	<b>Instruction Block with One Instruction is a Block</b> If an instruction block only contains one instruction, the block shall nevertheless be embraced with "{" and "}".

Table 90 Comments

Id	Rule <i>Chapter 3.6.3</i>
<i>[CCode_Comments_001]</i>	<b>Usage of Comments</b> Code which is not self-explanatory shall be extended by a meaningful comment. Also workarounds and rule violations shall be commented, too.
<i>[CCode_Comments_002]</i>	<b>Allowed Comment Marker</b> It is allowed to use "/* ... */" or "//" as comment marker styles to write a comment.
<i>[CCode_Comments_003]</i>	<b>Non-Nested Comments</b> Comments shall not be nested and comment styles shall not be mixed.
<i>[CCode_Comments_004]</i>	<b>Sections of Code Should Not be Commented Out</b> Sections of code should not be commented out.
<i>[CCode_Comments_005]</i>	<b>Language of Comments is English</b> Comments shall be written in English.
<i>[CCode_Comments_006]</i>	<b>Position of Comments</b> It is allowed to write a comment behind code but it is not allowed to write code behind a comment.
<i>[CCode_Comments_007]</i>	<b>Markers for Requirements Tracing</b> A special marker may be used within comments to trace requirements from AUTOSAR: TRACE[<AUTOSAR SWS Trace ID>].

Id	Rule <i>Chapter 3.6.3</i>
<i>[CCode_Comments_008]</i>	<b>Publishable Contents of Comments</b> All contents of comments shall be publishable. That means that comments ... <ul style="list-style-type: none"> <li>▶ shall not contain customer names, product names or customer specific wordings</li> <li>▶ shall not contain names of persons, names of departments or Bosch specific wordings</li> <li>▶ shall not contain four-letter words or other inappropriate phrases like "dirty hack", "bug", "quick and dirty" or "this can be a reason for a big recall"</li> </ul>

Table 91 Authoring of ECUC artifacts

Id	Rule <i>Chapter 4.1.1</i>
<i>[ECUC_001]</i>	<b>AUTOSAR conformance</b> Within BSW, only the ECUC formats defined by AUTOSAR are allowed to be used. In particular, MSR Conf is not to be used within BSW (but is allowed in legacy application software and adapters to legacy software). All AUTOSAR ECUC artifacts shall be conforming to the corresponding AUTOSAR specifications.
<i>[ECUC_002]</i>	<b>AUTOSAR version</b> On the head of the main development branch, all AUTOSAR ECUC artifacts shall be based on AUTOSAR release 4.0.2 or higher.

Table 92 ECUC Values: Content Responsibility

Id	Rule <i>Chapter 4.2.1</i>
<i>[ECUC_V001]</i>	<b>Content responsibility</b> Customer projects are responsible for ECUC values files unless this responsibility has explicitly been delegated. Project-specific ECUC values files are named <Comp>_EcucValues.arxml <sup>8</sup> .
<i>[ECUC_V002]</i>	<b>Protected configuration</b> Protected ECUC values shall not be changed by any developer who receives these files as a deliverable <sup>9</sup> . Protected ECUC values files are named <Comp>_Prot_EcucValues.arxml.
<i>[ECUC_V003]</i>	<b>Configuration examples</b> Configurable SW components shall also deliver example ECUC values files (except where these files would only contain values which are identical to their default values). Example ECUC values files are named <Comp>_EcucValues.arxml.
<i>[ECUC_V004]</i>	<b>Release conditions</b> ECUC values may only be released if they have been successfully checked, reviewed, and tested (see below for details). ECUC value examples need only be formally checked but not tested.

Table 93 ECUC Values: Editing, Checking, and Reviewing

Id	Rule <i>Chapter 4.2.2</i>
<i>[ECUC_V005]</i>	<b>Editing</b> ECUC values shall be edited with BCT.

<sup>8</sup> See also the guideline on file naming rules for an overview of all file naming conventions.

<sup>9</sup> Protected ECUC values are a concept which go beyond of what is currently defined by AUTOSAR itself. But this concept does also not violate any definition imposed by AUTOSAR.



Id	Rule <i>Chapter 4.2.2</i>
[ECUC_V006]	<b>Formal checks</b> ECUC values have to be formally checked before delivery by processing it in a suitable project context.
[ECUC_V007]	<b>Reviewing</b> Protected ECUC values shall be reviewed before releasing the SW component version which contains these values. Project-specific ECUC values shall be reviewed by a project expert before releasing the project version which contains these values.

Table 94 ECUC Values: Testing

Id	Rule <i>Chapter 4.2.3</i>
[ECUC_V008]	<b>Testing</b> Protected ECUC values shall be tested before SW component delivery as part of the unit test for this SW component. Project-specific ECUC values are tested in the context of the corresponding project.

Table 95 ECUC Values: Content-related Procedures

Id	Rule <i>Chapter 4.2.4</i>
[ECUC_V009]	<b>Container short names</b> The short name of each container in the ECUC values files shall be identical to the short name of the corresponding container definition <b>if</b> it is not allowed to exist more than once. E.g., the short name of the "DioGeneral" container shall be "DioGeneral". No ECUC processor may rely on this convention, though. For containers which may exist more than once according to the ECUC ParamDef (i.e. UpperMultiplicity > 1), no general conventions apply.
[ECUC_V010]	<b>ArPackage usage</b> All ECU configuration values shall be within the ArPackage hierarchy <b>/RB/UBK/Project/EcucModuleConfigurationValues.</b> See the guideline rules for ArPackage usage for details.
[ECUC_V011]	<b>EcucValueCollection usage</b> No BSW module specific ECUC values file shall contain a <b>&lt;ECUC-VALUE-COLLECTION&gt;</b> . Not more than one centrally maintained ECUC values file per project shall contain this collection.

Table 96 ECUC ParamDefs: Content Responsibility

Id	Rule <i>Chapter 4.3.1</i>
[ECUC_PD001]	<b>Content responsibility</b> ECUC parameter definitions are in the sole responsibility of the developer whose SW component is configurable by means of the ECU configuration methodology. They shall be named <b>&lt;Comp&gt;_Ecuc-ParamDef.xml</b> .

Table 97 ECUC ParamDefs: Editing, Checking, and Reviewing

Id	Rule <i>Chapter 4.3.2</i>
[ECUC_PD002]	<b>Editing</b> ECUC parameter definitions shall be changed in a backwards-compatible way whenever possible. Incompatible changes shall be documented.
[ECUC_PD003]	<b>Formal checks</b> ECUC parameter definitions shall be formally checked for validity against the corresponding version of the AUTOSAR schema before delivery.

Id	Rule <i>Chapter 4.3.2</i>
<a href="#">[ECUC_PD004]</a>	<b>Reviewing</b>  After a successful formal check, the parameter definitions shall be reviewed against the requirements which triggered their creation or update.

Table 98 ECUC ParamDefs: Content-related Procedures

Id	Rule <i>Chapter 4.3.4</i>
<a href="#">[ECUC_PD005]</a>	<b>Vendor-specific derivation</b>  If the AUTOSAR standard specifies ECUC parameters for the particular SW component, then the ECUC parameter definition file for this SW component shall adhere to the "vendor-specific derivation rules" provided in the AUTOSAR ECUC specification in chapter 5.
<a href="#">[ECUC_PD006]</a>	<b>One module definition per file</b>  No ECUC parameter definition file shall contain the definition of more than one <b>&lt;ECUC-MODULE-DEF&gt;</b> .
<a href="#">[ECUC_PD007]</a>	<b>English language</b>  The only language to be used is English.
<a href="#">[ECUC_PD008]</a>	<b>EcucDefinitionCollection usage</b>  No BSW module specific ECUC parameter definition file shall contain a <b>&lt;ECUC-DEFINITION-COLLECTION&gt;</b> . Not more than one centrally maintained ECUC parameter definition file per project shall contain this collection.
<a href="#">[ECUC_PD009]</a>	<b>Naming conventions for short names</b>  The contents of the <b>&lt;SHORT-NAME&gt;</b> of each parameter definition (incl. containers and references) shall adhere to the AUTOSAR naming convention of ECUC parameters: upper camel case, starting with the owning SW component's name (e.g. "DioGeneral"). If the component's name contains underscores, the actual parameter name shall be separated from the SW component name by another underscore (e.g. "rba_loSigDio_General"). Bosch-specific parameters in SW components specified by AUTOSAR shall be marked by an "Rb" infix directly after the SW component name, e.g. "DioRbLegacyApi".
<a href="#">[ECUC_PD010]</a>	<b>Order of parameters</b>  The <b>&lt;SHORT-NAME&gt;</b> s of all items inside a container shall be in the following order: <ol style="list-style-type: none"> <li>1. all parameters in alphabetical order</li> <li>2. all references in alphabetical order</li> <li>3. all sub-containers in alphabetical order.</li> </ol>
<a href="#">[ECUC_PD011]</a>	<b>Long name of parameters</b>  Each ECUC parameter (incl. containers and references) shall provide a human-readable <b>&lt;LONG-NAME&gt;</b> . This is typically used as a display text in configuration editors. If providing a unit is helpful for the user, then it shall be appended behind the actual parameter long name, e.g. "Maximum delay [us]".
<a href="#">[ECUC_PD012]</a>	<b>Description of parameters</b>  Each parameter (incl. containers and references) shall provide a proper description of this parameter in the <b>&lt;DESC&gt;</b> element. If one paragraph of description is not sufficient, it shall be continued in the <b>&lt;INTRODUCTION&gt;</b> element.
<a href="#">[ECUC_PD013]</a>	<b>Specification of parameter origin</b>  The <b>&lt;ORIGIN&gt;</b> of any parameter not defined by AUTOSAR shall be set to "RB", possibly followed by a version and/or date specifier.



Id	Rule <i>Chapter 4.3.4</i>
[ECUC_PD014]	<b>ArPackage usage</b> All ECUC parameter definitions shall be either within the ArPackage hierarchy <b>/AUTOSAR_&lt;Module&gt;/EcucModuleDefs</b> (for BSW components where AUTOSAR defines a standardized ECUC parameter definition) or <b>/RB/RBA/&lt;Module&gt;/EcucModuleDefs</b> (for all other BSW components). See the guideline rules for ArPackage usage for details.
[ECUC_PD015]	<b>Anticipation of parameters from future AUTOSAR releases</b>
[ECUC_PD016]	<b>Multiplicity handling</b> If the multiplicities of a standardized ECUC parameter shall be changed in a vendor-specific derivation, the following rules apply: <ul style="list-style-type: none"> <li>▶ The multiplicity interval defined in a vendor-specific ECUC parameter definition can be made smaller than its standardized counterpart without violating the standard as long as the standardized interval limits are not crossed. Nevertheless, this fact shall be documented as a deviation from the AUTOSAR specification.</li> <li>▶ If the multiplicity interval defined in a vendor-specific ECUC parameter definition goes beyond at least one interval limit defined in the standardized ECUC parameter definition, then this vendor-specific derivation violates the AUTOSAR standard. As such, it must be justified and well documented if it is not avoidable.</li> </ul>
[ECUC_PD017]	<b>Default values for mandatory parameters</b> Default values for mandatory parameters are a mere editing aid with the semantics of a "typical value". The existence of a default value for a parameter does not make this parameter implicitly optional.

Table 99 ECUC Processors: Content Responsibility and Language

Id	Rule <i>Chapter 4.4.1</i>
[ECUC_P001]	<b>Content responsibility</b> ECUC processors and the related helper files (libraries, templates, etc.) are in the sole responsibility of the SW developer whose SW component is configurable by means of the ECU configuration methodology.
[ECUC_P002]	<b>Language</b> The only allowed languages for ECUC processors are oAW (openArchitectureWare) and Perl. No other languages are allowed. The preferred language is oAW. Inside CDG, this language is to be used for all SW components for which no explicit allowance of using Perl has been defined by CDG management. This exception is granted for all SW components belonging to: <ul style="list-style-type: none"> <li>▶ MemStack</li> <li>▶ IoStack</li> <li>▶ MCAL layer except MCAL SW components belonging to the ComStack</li> <li>▶ Calibration Software</li> <li>▶ ECUSec</li> <li>▶ Boot Control</li> <li>▶ BSW Library.</li> </ul> CDG packages shall not contain a mix of ECUC processor languages.

Table 100 ECUC Processors: Editing, Checking, and Reviewing

Id	Rule <i>Chapter 4.4.2</i>
<i>[ECUC_P003]</i>	<b>Reviewing</b>  Before delivery, ECUC processors shall be reviewed against the corresponding requirements and existing content-related procedures (see below).

Table 101 ECUC Processors: Testing

Id	Rule <i>Chapter 4.4.3</i>
<i>[ECUC_P004]</i>	<b>Testing</b>  ECUC processors are tested as part of the SW component's unit test. Ideally, all branches of the ECUC processors should be covered by these tests. It shall be tested that correct ECUC value input produces correct ECUC processor output (including logs, reports, and ECUC processor messages) according to the requirements. To address missing checks for existence of optional parameters, there shall be at least one test case where all optional parameters are not existing in the ECUC values.

Table 102 ECUC Processors: Documentation

Id	Rule <i>Chapter 4.4.4</i>
<i>[ECUC_P005]</i>	<b>Documenting</b>  Apart from line-oriented documentation inside the ECUC processor code itself, also the overall ECUC processor concept shall be documented unless it is a straightforward 1:1 implementation of the AUTOSAR specification of the related BSW module.

Table 103 ECUC Processors: General Content-related Procedures

Id	Rule <i>Chapter 4.4.5</i>
<i>[ECUC_P006]</i>	<b>Allowed output</b>  Only artifacts of the own component (this includes AUTOSAR interface definitions) are allowed to be generated. Exceptions from this rule must be approved before implementation and they must be documented by the process responsible for ECU configuration <sup>10</sup> . In addition to code artifacts and meta information (such as fragments of BSW module descriptions or SW component descriptions), only documentation fragments, configuration reports and configuration exports may be generated. For generated code and meta information, the same quality demands and coding guidelines apply as for manually created code.
<i>[ECUC_P007]</i>	<b>Stable output</b>  In order to provide reproducible output and to allow the easy comparison of the files generated by an ECUC processor, the following rules shall be obeyed: <ul style="list-style-type: none"><li>▶ Generated files are not allowed to contain path, date, time, or user information.</li><li>▶ No ECUC processor may rely on a particular order of non-ordered source data (in AUTOSAR, all multiple instances of the same container, parameter, or reference are non-ordered). It may even happen that the order of these elements as seen by the ECUC processor varies from processor run to processor run, even with unchanged source data. The ECUC processor shall establish its own reproducible order in these cases, e.g. by alphabetical sorting of the container instances by the short name.</li><li>▶ Generated files need not contain any SCM header because they are typically not checked in.</li></ul>

<sup>10</sup> Within CDG, the only approved exception are the components rba\_Wdg\* which are allowed to generate artifacts of components Wdg\_6\*.

Id	Rule <i>Chapter 4.4.5</i>
<i>[ECUC_P008]</i>	<b>Allowed input</b> <p>Main input of each ECUC processor shall be the ECUC values belonging to the ECUC ParamDefs delivered in the same SW component as the ECUC processor.</p> <p>Additional input from other SW components is allowed if and only if this is technically necessary. In case of ECUC parameters standardized by AUTOSAR, only parameters marked with scope "global" or "ECU" in the AUTOSAR specification of these other SW components are allowed to be taken as additional input.</p> <p>Due to maintenance issues, a global configuration values section (also known as GLOBDATA) is not allowed.</p>
<i>[ECUC_P009]</i>	<b>Independence from input origin</b> <p>ECUC processors are not allowed to react differently in case of</p> <ul style="list-style-type: none"> <li>▶ manually created input ECUC values resp.</li> <li>▶ forwarded ECUC values.</li> </ul>
<i>[ECUC_P010]</i>	<b>Independence from container short names in ECUC Values</b> <p>The short names of containers in ECUC Values shall only be used for identification purposes. No actual SW functionality may be influenced depending on short name contents except for the creation of #defines corresponding to ECUC parameters with the SymbolicNameValue attribute set to true.</p>
<i>[ECUC_P011]</i>	<b>(Removed)</b> <p>Removed.</p>
<i>[ECUC_P012]</i>	<b>Naming conventions for input files</b> <p>There shall be a clearly evident relation from input templates to the corresponding output files. E.g., if a generated file is called Hugo_Cfg.h, then</p> <ul style="list-style-type: none"> <li>▶ the corresponding oAW template shall be called Hugo_Cfg_h.xpt and</li> <li>▶ the corresponding Perl template shall be called Hugo_Cfg.ht.</li> </ul>
<i>[ECUC_P013]</i>	<b>Naming conventions for output files</b> <p>The following naming conventions apply for files generated by ECUC processors:</p> <ul style="list-style-type: none"> <li>▶ Files containing PreCompile-time information shall be named <code>&lt;Comp&gt;_Cfg[_&lt;Sub&gt;].&lt;ext&gt;</code></li> <li>▶ Files containing link time information shall be named <code>&lt;Comp&gt;_Lcfg[_&lt;Sub&gt;].&lt;ext&gt;</code></li> <li>▶ Files containing PostBuild-time information shall be named <code>&lt;Comp&gt;_PBcfg[_&lt;Sub&gt;].&lt;ext&gt;</code></li> <li>▶ Files defining AUTOSAR interfaces to ASW shall be named <code>&lt;Comp&gt;_Cfg[_&lt;Sub&gt;]_SWCD.xml</code></li> <li>▶ Files defining (fragments of) BSW module descriptions shall be named <code>&lt;Comp&gt;_Cfg[_&lt;Sub&gt;]_BSWMD.xml</code></li> <li>▶ Report files shall be named <code>&lt;Comp&gt;_Report.txt</code></li> <li>▶ Export files shall be named <code>&lt;Comp&gt;_Export.xml</code></li> </ul>

Id	Rule <i>Chapter 4.4.5</i>
<i>[ECUC_P014]</i>	<p><b>Clear separation of actions</b></p> <p>ECUC processors fulfil the following tasks (not necessarily all of them):</p> <ul style="list-style-type: none"> <li>▶ Validation: checks input ECUC values for potential semantic problems.</li> <li>▶ File generation: creates output files depending on input ECUC values.</li> <li>▶ Forwarding: creates output ECUC values depending on input ECUC values.</li> </ul> <p>These aspects shall be clearly separated in different processor actions. In particular, forwarders and generators shall not be mixed in one processor action. In oAW, also the validators shall be separated from the forwarders and generators. In Perl, separating the validators is often not a feasible approach, and hence validation is typically part of the forwarder and/or generator actions.</p>
<i>[ECUC_P015]</i>	<p><b>Validation responsibility</b></p> <p>No assumption about the correctness of input ECUC values shall be made in the ECUC processors. If a forwarder or generator action only produces valid output if some conditions on the input side are met, then the fulfillment of these conditions must be verified before actually forwarding or generating output in this action.</p> <p>To meet the general single maintenance goals, also each validation code shall only be written once if technically feasible. In particular, validations centrally carried out by the configuration framework shall not be repeated in SW component-specific code.</p>
<i>[ECUC_P016]</i>	<p><b>Problem reporting</b></p> <p>Problem reporting in ECUC processors shall meet the following expectations:</p> <ul style="list-style-type: none"> <li>▶ If no error is reported, then the configured SW component works according to its specification.</li> <li>▶ Suspicious configurations which are typically caused by wrong ECUC values but still result in error-free operation (see above) shall trigger a warning.</li> <li>▶ Merely informational items shall be provided via report files and logs only.</li> </ul>
<i>[ECUC_P017]</i>	<p><b>Logging</b></p> <p>Logging shall only be used for analyzing the trace of operations in the ECUC processors. The intended audience of log files are the developers of ECUC processors, not ECUC users. Hence, customer-relevant information shall not be logged but issued via reports and/or generated documentation instead.</p>
<i>[ECUC_P018]</i>	<p><b>Generating configuration documentation and reports</b></p> <p>Currently, the effective configuration of a SW component is documented by plain text files (aka report files). No report-specific guidelines are defined yet.</p> <p>Eventually, these reports will be replaced with generated documentation fragments according to the standard AUTOSAR documentation concepts.</p>

Id	Rule <i>Chapter 4.4.5</i>
<i>[ECUC_P019]</i>	<p data-bbox="343 264 726 293"><b>Handling of PostBuild data sets</b></p> <p data-bbox="343 309 1476 398">The following conventions apply for multiple configuration containers (used for post-build selectable configuration, marked as <b>&lt;MULTIPLE-CONFIGURATION-CONTAINER&gt;</b> set to <i>true</i> in the ECUC parameter definition):</p> <ul style="list-style-type: none"> <li data-bbox="343 421 534 450">▶ Configuration:           <ul style="list-style-type: none"> <li data-bbox="375 465 1476 645">– The short name of the multiple configuration container shall start with the name of the configuration container itself plus an optional suffix identifying the data set, separated by an underscore. For example, if the multiple configuration container for SW component "Hugo" is called "HugoConfig", then "HugoConfig" or "HugoConfig_4Cyl" or "HugoConfig_8Cyl" are valid short names of these container values while "Hugo_4Cyl" or "HugoConfig8Cyl" are not. This convention shall be checked by the corresponding ECUC processor.</li> <li data-bbox="375 667 1476 757">– If forwarding takes place to a forwarding target which has no post-build capabilities at all, then a technically feasible superset of all data sets of the forwarding origin shall be pushed (e.g. when forwarding data from powerstage drivers to the OS).</li> </ul> </li> <li data-bbox="343 779 566 808">▶ Implementation:           <ul style="list-style-type: none"> <li data-bbox="375 824 1476 913">– For each variant-specific configuration data set (e.g. "HugoConfig_4Cyl", "HugoConfig_8Cyl"), a corresponding C structure instance with the same name as this configuration data set shall be generated by the ECUC processor.</li> <li data-bbox="375 936 1476 992">– These C structures shall contain or reference <b>all</b> configuration information belonging to the corresponding post-build data set.</li> <li data-bbox="375 1014 1476 1070">– All these C structure instances shall be of the same type which shall have the name <i>&lt;Comp&gt;_ConfigType</i> (e.g. "Hugo_ConfigType").</li> <li data-bbox="375 1093 1476 1272">– All post-build dependent C data shall be instantiated in <i>&lt;Comp&gt;_PBcfg.c</i>. This file shall contain <b>only</b> post-build configuration data, and <b>no</b> other file belonging to this SW component shall contain information which depends on the post-build selectable contents of the post-build configuration data sets. The number of post-build data sets may also influence the contents of other files, though (e.g. if the number of post-build data sets is stored in a #define in <i>&lt;Comp&gt;_Cfg.h</i>).</li> <li data-bbox="375 1294 1476 1384">– All post-build dependent C data shall be located in a separate MemMap section <i>&lt;COMP&gt;_START/STOP_PBCFG_&lt;SIZE&gt;</i>. No pointer inside this section should point to something outside of this section.</li> <li data-bbox="375 1406 1396 1435">– The interface to this post-build configuration data shall be located in <i>&lt;Comp&gt;_PBcfg.h</i>.</li> </ul> </li> </ul>

Table 104 ECUC Processors: oAW-specific Procedures

Id	Rule <i>Chapter 4.4.6 [OP]</i>
<i>[ECUC_OP_001]</i>	<p data-bbox="343 1559 638 1588"><b>File naming conventions</b></p> <p data-bbox="343 1603 1476 1659">A oAW processor for a component <i>&lt;Comp&gt;</i> shall be a set of oAW scripts with the following naming conventions:</p> <p data-bbox="343 1682 973 1711">Workflow Control (Model Workflow Environment) files:</p> <ul style="list-style-type: none"> <li data-bbox="343 1715 1037 1744">– for configuration forwarding named <i>&lt;Comp&gt;_Forward.mwe</i></li> <li data-bbox="343 1749 1021 1778">– for configuration validation named <i>&lt;Comp&gt;_Validate.mwe</i></li> <li data-bbox="343 1783 941 1812">– for code generator named <i>&lt;Comp&gt;_Generate.mwe</i></li> <li data-bbox="343 1816 933 1845">– for ID generator named <i>&lt;Comp&gt;_Generateld.mwe</i></li> </ul> <p data-bbox="343 1868 550 1897">Forwarder scripts:</p> <ul style="list-style-type: none"> <li data-bbox="343 1901 1021 1930">– configuration forwarder files: <i>&lt;Comp&gt;_Forward_&lt;Sub&gt;.ext</i></li> </ul> <p data-bbox="343 1953 534 1982">Validator scripts:</p> <p data-bbox="901 1995 917 2024">▽</p>

Id	Rule <i>Chapter 4.4.6 [OP]</i>
	<p style="text-align: center;">△</p> <ul style="list-style-type: none"> <li>– configuration validation files: <code>&lt;Comp&gt;_&lt;Sub&gt;.chk</code></li> <li>– configuration validation extension scripts: <code>&lt;Comp&gt;_&lt;Sub&gt;.ext</code></li> </ul> <p>Generator scripts:</p> <ul style="list-style-type: none"> <li>– configuration generator template (Xpand) files: <code>&lt;Comp&gt;_Generate&lt;Sub&gt;.xpt</code></li> <li>– configuration generator extension (Xtend) files: <code>&lt;Comp&gt;_Generate&lt;Sub&gt;.ext</code></li> <li>– C code file generator template (Xpand) files: <code>&lt;Comp&gt;_&lt;Sub&gt;_Cfg_c.xpt</code></li> <li>– C header file generator template (Xpand) files: <code>&lt;Comp&gt;_&lt;Sub&gt;_Cfg_h.xpt</code></li> <li>– RTE configuration generator template (Xpand) files: <code>&lt;Comp&gt;_&lt;Sub&gt;_Cfg_Bswmd_&lt;Suffix&gt;.xpt</code> and <code>&lt;Comp&gt;_&lt;Sub&gt;_Cfg_Swcd_&lt;Suffix&gt;.xpt</code></li> </ul> <p>Generate ID scripts:</p> <ul style="list-style-type: none"> <li>– ID generator files: <code>&lt;Comp&gt;_Generateld_&lt;Sub&gt;.chk</code></li> <li>– ID generator extension files: <code>&lt;Comp&gt;_Generateld_&lt;Sub&gt;.ext</code></li> </ul> <p>The file name parts <code>_&lt;Sub&gt;</code> and <code>_&lt;Suffix&gt;</code> are optional.</p>
<i>[ECUC_OP_002]</i>	<p><b>Subdirectories</b></p> <p>oAW scripts shall be placed in the <i>scripts</i> folder of the respective component.</p> <p>If Subdirectories cannot be avoided, e.g. if the number of files is too large to have a good overview, the following folder structures shall be used:</p> <ul style="list-style-type: none"> <li>– for generate ID and configuration forwarders: <i>scripts\forwarder</i></li> <li>– for code and other file generators: <i>scripts\generator</i></li> <li>– for configuration validations: <i>scripts\validator</i></li> <li>– for utility extensions: <i>scripts\util</i></li> </ul>
<i>[ECUC_OP_003]</i>	<p><b>Forward scripts need separate action</b></p> <p>Forward scripts shall have a separate action defined with complete input and output data declaration in BAMF.</p>
<i>[ECUC_OP_004]</i>	<p><b>Id Generator scripts need separate action</b></p> <p>ID Generators shall have separate actions defined with complete input and output data declaration in BAMF.</p>
<i>[ECUC_OP_005]</i>	<p><b>Generator scripts need separate action</b></p> <p>Generator scripts shall have a separate actions defined with complete input data and output artifact declaration in BAMF.</p>

Table 105 ECUC Processors: Perl-specific Procedures

Id	Rule <i>Chapter 4.4.7</i>
<i>[ECUC_PP001]</i>	<p><b>File naming conventions</b></p> <p>A Perl processor shall be a Perl module named <code>&lt;Comp&gt;_Process.pm</code>. Additional helper Perl modules (if required) shall be named <code>&lt;Comp&gt;[_&lt;Add&gt;]_Ext.pm</code>.</p>
<i>[ECUC_PP002]</i>	<p><b>Usage of the <code>conf_process</code> API</b></p> <p>Wherever there is an API method of the configuration framework "conf_process" available for a particular task, this method shall be used by all Perl processors instead of low-level Perl operations. Only the APIs documented below from the Perl modules documented below are allowed to be used.</p>
<i>[ECUC_PP003]</i>	<p><b>Usage of global variables</b></p> <p>Global variables in Perl (i.e. variables in file scope) are evil. Try to avoid them whenever somehow possible.</p>

Id	Rule <i>Chapter 4.4.7</i>
[ECUC_PP004]	<b>Conventions for injection markers</b> Injection markers shall always have the form <code>&lt;/Identifier/&gt;</code> . To be consistent with existing Perl processors, the identifiers used for these injection markers shall be in all uppercase using "_" as word separators. The first word in these injection markers shall be the name of the SW component which owns this Perl processor.
[ECUC_PP005]	<b>Action naming conventions</b> Perl subroutines corresponding to processor actions shall be named according to the following scheme: <ul style="list-style-type: none"> <li>▶ subroutines preparing the configuration input for further processing<sup>11</sup> shall be called Prepare[&lt;Suffix&gt;]</li> <li>▶ subroutines generating files (model to text transformations) shall be called Generate[&lt;Suffix&gt;]</li> <li>▶ subroutines generating configuration data (model to model transformations) shall be called Forward[&lt;Suffix&gt;].</li> </ul> The <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders in a Perl processor.
[ECUC_PP006]	<b>Restriction of output file locations and names</b> Perl subroutines corresponding to processor actions shall generate files only within the paths provided by the configuration framework. Subfolders of these paths shall neither be created nor used by Perl processor actions. The names of the generated files shall strictly adhere to the names provided in the corresponding build action manifest file.
[ECUC_PP007]	<b>Dynamic registration of generated files</b> Whenever it is not possible to specify the name of a generated file statically in the module's BAMF file (and hence, a wildcard operator is used in the related BAMF section), each generated file with file name determined at Perl processor runtime <ul style="list-style-type: none"> <li>▶ shall be registered using the RegisterDynamicArtifact API of conf_process and</li> <li>▶ shall match exactly one BAMF specification of created artifacts which uses wildcard operators.</li> </ul> Whenever the <i>name</i> of a generated file can already be known at BAMF specification time, dynamic registration of generated files is not permitted.
[ECUC_PP008]	<b>No WhoAml or Register in AUTOSAR BSW</b> The legacy Perl subroutines WhoAml() and Register() shall not be used in AUTOSAR BSW.

Table 106 BuildAction Declarations

Id	Rule <i>Chapter 4.5.1</i>
[ECUC_B001]	<b>BuildActions names shall be unique</b> All BuildActions to be executed within a given context (that is in a full program stand at max) shall have unique names.
[ECUC_B002]	<b>Naming conventions for BuildActions</b> All BuildActions should comply with the following naming convention: <ModuleName>_<Function> where the <ModuleName> corresponds to the BSW module shortname the script was developed for and the <Function> is either Generate or Forward. For legacy MSR modules, the possible values for <Function> are Register, SetGlobals, Forward and Generate.
[ECUC_B003]	<b>Declaring Startup-/Teardown Action References</b> Startup-/Teardown Action References should be declared within just one BAMF file.

<sup>11</sup> Typical use cases for such prepare actions are the automatic calculation of ID values or centralized validations. This involves the reading and writing of these actions to the same configuration subtree.



Id	Rule <i>Chapter 4.5.1</i>
<a href="#">[ECUC_B004]</a>	<b>Usage of Startup-/Teardown-/Followup and Predecessor Action References</b> Before introducing any declaration such as Startup-/Teardown-/Followup- or Predecessor Action References it shall always be made sure that the corresponding relationship to other declarations can't be achieved with regular IO-Element declarations with justifiable effort.
<a href="#">[ECUC_B005]</a>	<b>Invocation Parameter keys</b> A BuildAction instance shall not have two or more invocation parameters with the same key.

Table 107 Artefact Declarations

Id	Rule <i>Chapter 4.5.2.1</i>
<a href="#">[ECUC_BIOA001]</a>	<b>Artefact Declaration Composition</b> The AUTOSAR standard defines an artefact declared in a BAMF file by the following four attributes: Artefact name, Extension, Class and Domain. Every artefact declaration shall contain a valid value for all of them.
<a href="#">[ECUC_BIOA002]</a>	<b>Artefact Declaration Characteristics</b> As defined by the rule above, there are four attributes which clearly define an artefact. The characters which are allowed to be used for each of them can be found below and no declaration shall contain any other character but the ones mentioned in this rule. Artefact Name: [A-Z][a-z][0-9]_ Filetype: [A-Z][a-z][0-9]_ Class: [A-Z][a-z][0-9]_ Domain: [A-Z][a-z][0-9]_
<a href="#">[ECUC_BIOA003]</a>	<b>Uniqueness of Output Declarations</b> One BuildAction instance shall never contain duplicate output element declarations which will be applied to one and the same artefact.

Table 108 Model Reference Declarations

Id	Rule <i>Chapter 4.5.2.2</i>
<a href="#">[ECUC_BIOM001]</a>	<b>Required declaration of accessed Model Data</b> Build scripts of any technical nature (Perl or oAW) shall only access model elements they also explicitly declared in their corresponding Build Action. That means, a script is only allowed to read data from /AUTOSAR_CAN/EcucModuleDefs/CAN if it also declared this AR_OBJECT to be an input IO-element under the same BuildAction it is encapsulated by.
<a href="#">[ECUC_BIOM002]</a>	<b>Model Reference Identifiers</b> For any reference to a BSW module, the following reference pattern shall be used: For an AUTOSAR standardized module named <BSWM>: /AUTOSAR_<BSWM>/EcucModuleDefs/-<BSWM> For a Non-Standardized module in AUTOSAR format named <BSWM>: /RB/RBA/<BSWM>/EcucModuleDefs/<BSWM> For a legacy MSR module named <BSWM>: /MEDC17/<BSWM>
<a href="#">[ECUC_BIOM003]</a>	<b>Model Reference Category</b> Whenever a model object is referenced by an IO-Element, the corresponding category needs to be used. That is in case of an MSR reference, the category shall be MSR_OBJECT, in case of an AUTOSAR reference, an AR_OBJECT shall be used. The model reference text needs to correspond to the rule mentioned above.



Table 109 Action Specific Declarations

Id	Rule <i>Chapter 4.5.3</i>
<i>[ECUC_BAS001]</i>	<b>Artefact marked as processor</b> Every Perl and oAW Action shall declare exactly one input artefact which is well defined (that is: no usage of wildcards in the defining attributes allowed) and marked with the Port-ID PROCESSOR.
<i>[ECUC_BAS002]</i>	<b>Generated Artefacts for Perl Actions</b> Output and log directory can be centrally configured for all Perl Actions in the single Setup Action which must exist in any Perl processing project. Any perl script creating output artefacts which are also declared in their corresponding BAMF files shall to stick to these centrally configured directories.
<i>[ECUC_BAS003]</i>	<b>Generated Artefacts for oAW Actions</b> Output artefacts which should be processed in the build toolchain need to be configured via the appropriate means for oAW (outlets defined in the workflow file or referenced centrally). Due to limitations in the toolchain, the output folder should be _out directly underneath the project root at the moment. At a later release it will be possible to configure the output folder for each oAW Action in its corresponding BAMF file. Once this is the case, this rule will have to be altered.
<i>[ECUC_BAS004]</i>	<b>Perl Actions Required Invocation Parameters</b> The following parameters shall be present for any Perl Action and be filled with a valid value: MODULE_NAME STEP_NAME The value for the parameter MODULE_NAME shall be the same as the first part of the BuildAction name The value for the parameter STEP_NAME shall be the same as the last part of the BuildAction name
<i>[ECUC_BAS005]</i>	<b>oAW Actions Required Invocation Parameters</b> There are no mandatory invocation parameters for an oAW Action as of today. As this will change in the future, they'll be added to this rule.
<i>[ECUC_BAS006]</i>	<b>Setup Action Required Invocation Parameters</b> There are no mandatory invocation parameters for the Setup Action today. As this will change in the future, they'll be added to this rule.

Table 110 Code Layout and Comments

Id	Rule <i>Chapter 5.1 [PerlCoding_CodeLayout]</i>
<i>[Perl_001]</i>	<b>Sections of Perl files</b> Every source file contains the following sections <ul style="list-style-type: none"> <li>▶ a file header (with copyright information and a short description)</li> <li>▶ the actual source code</li> </ul>
<i>[Perl_002]</i>	<b>Comments</b> Comments in the code shall be written in English
<i>[Perl_003]</i>	<b>Comment of a subroutine</b> Every function in the Perl source is preceded by a comment block, specifying a synopsis of the defined function

Id	Rule <i>Chapter 5.1 [PerlCoding_CodeLayout]</i>
<i>[Perl_004]</i>	<b>Indentation and spacing style</b> Use the following indentation style: <ul style="list-style-type: none"> <li>▶ use 2-column indentation; don't use hard tabs</li> <li>▶ opening curly bracket below to the keyword in the next line at the column as the keyword ("open braces left")</li> <li>▶ closing curly bracket lines up with the keyword that started the block</li> <li>▶ blank lines between groups of code that do different things</li> <li>▶ no space between function name and its opening parenthesis</li> <li>▶ space after each comma</li> <li>▶ space between a keyword and opening parenthesis</li> <li>▶ space around operators</li> <li>▶ line up corresponding items vertically</li> <li>▶ use parenthesis to indicate evaluation order</li> <li>▶ use spaces where it improves readability</li> </ul>
<i>[Perl_005]</i>	<b>Line length</b> The line length of 120 characters shall not be exceeded.

Table 111 Naming Conventions

Id	Rule <i>Chapter 5.2 [PerlCoding_NamingConv]</i>
<i>[Perl_006]</i>	<b>Packages and filenames</b> Perl script associated files shall start with the module name and are written in camel case (e.g. mixed upper/lower case).
<i>[Perl_007]</i>	<b>Constants</b> Constants are written with uppercase characters. They begin with letters, followed by word characters and underscores used to separate the components in a long name. Constants are preferably defined with the "use constant" pragma.
<i>[Perl_008]</i>	<b>Naming of global and local variables</b> <ul style="list-style-type: none"> <li>▶ Variable names are written in mixed upper/lower case, and start with a lower-case letter</li> <li>▶ All variables should be lexical (defined with my) and global variables should be avoided.</li> <li>▶ Prefixes may specify the kind of usage of a local variable.</li> <li>▶ The names of variables are suffixed with:               <ul style="list-style-type: none"> <li>– "_h" for hash variables</li> <li>– "_a" for array variables</li> <li>– "_s" for scalar variables</li> </ul> </li> <li>▶ Suffixes are not used for variables with a very limited scope (e. g. loop indices).</li> </ul>

Id	Rule <i>Chapter 5.2 [PerlCoding_NamingConv]</i>
<i>[Perl_009]</i>	<b>References</b> The names of reference variables are suffixed with <ul style="list-style-type: none"> <li>▶ "_ph" for references to hash variables</li> <li>▶ "_pa" for references to array variables</li> <li>▶ "_ps" for references to scalar variables</li> <li>▶ "_pg" for references to typeglobs</li> <li>▶ "_pf" for references to functions</li> </ul>
<i>[Perl_010]</i>	<b>Names of subroutines</b> Subroutines names are written in camel case, and start with an upper-case letter.
<i>[Perl_0101]</i>	<b>File handles</b> File handles (typeglobs) are written in upper case letters.
<i>[Perl_011]</i>	<b>Names for hashes and arrays</b> Name of arrays may be in plural and hashes in singular.

Table 112 Coding Conventions

Id	Rule <i>Chapter 5.3 [PerlCoding_CodingConventions]</i>
<i>[Perl_012]</i>	<b>Function Calls</b> The first thing to do in a function is to fetch its parameters
<i>[Perl_020]</i>	<b>Reading Configuration Data of Other Modules</b> The function <code>conf_process::IsModuleExistent()</code> shall be used to determine the existence of an optional Module (i.e a Module which is not necessarily part of a project environment). To actually get access to its data, an appropriate *bamf file entry is required, see rule set "ECU Configuration".

Table 113 Programming Tips

Id	Rule <i>Chapter 5.4</i>
<i>[Perl_014]</i>	<b>Defensive Programming</b> Make a habit of defensive programming <ul style="list-style-type: none"> <li>▶ always use the <code>-w</code> option when you call the perl interpreter</li> <li>▶ the "use strict" and the "use warning" should always be used in modules</li> <li>▶ always check function return values</li> <li>▶ watch for external program failures in <code>\$?</code></li> <li>▶ always check your input (including command line arguments)</li> <li>▶ always have an else after a chain of elsif's (even when the else case is empty)</li> <li>▶ always have a default after a chain of given's (even when the default case is empty)</li> <li>▶ put commas at the end of lists so your program won't break if someone inserts another item at the end of the list.</li> </ul>
<i>[Perl_015]</i>	<b>Make regular expressions readable</b> <ul style="list-style-type: none"> <li>▶ You can use comments and spaces in regular expressions to make them more readable, if you use the pattern modifier <code>/x</code></li> <li>▶ You can split a complex regular expression in parts and store them in variables packaged by <code>qr</code> (since Perl 5.6).</li> </ul>

Id	Rule <i>Chapter 5.4</i>
[Perl_016]	<b>Make dereferenciation to references readable</b> <ul style="list-style-type: none"> <li>▶ The usage of the arrow operator allows a more compact notation of dereferenciation.</li> <li>▶ The storage of intermediate results in blocks makes complicate pointer constructs more readable.</li> </ul>
[Perl_017]	<b>Handling of multi-line strings</b> For the definition of multi-line strings should preferred here documents.
[Perl_018]	<b>Local overwriting of global variables</b> Avoid the overwriting of global variables and furthermore of Perl system variables.

Table 114 Templates

Id	Rule <i>Chapter 5.5</i>
[Perl_019]	<b>Removed, template file is available</b>

Table 115 Rules

Id	Rule <i>Chapter 7.1.1.1.1.1 [AppDataType_Val_Rule]</i>
[Data_003]	<b>Removed.</b>
[Data_004a]	<b>Defining ApplicationDataType</b> <b>Scope:</b> AUTOSAR 4.x <ol style="list-style-type: none"> <li>1. Some <i>ApplicationDataType</i> are standardized by AUTOSAR and are provided as central elements.</li> <li>2. All other <i>ApplicationDataType</i> are Module/Component specific.</li> </ol> Component Developer should reuse the <i>ApplicationDataType</i> defined for the component as far as possible. Referring to <i>ApplicationDataType</i> defined by other Software Component shall not be used.
[Data_114]	<b>Data type for Application Primitive Data Type</b> <b>Scope:</b> AUTOSAR 4.x Primitive <i>ApplicationDataTypes</i> shall be mapped to Autosar specified <i>ImplementationDataTypes</i> (uint 8, sint 16 etc.) which are available as central elements.
[Data_119]	<b>DataConstr are mandatory for ApplicationDataType that are mapped to Float</b> <b>Scope:</b> AUTOSAR 4.X <i>ApplicationDataType</i> shall have a <i>DataConstr</i> , if it is mapped to <i>ImplementationDataType</i> of type float.
[Data_142]	<b>Definition of SwDataDefProps for ApplicationPrimitiveDataType</b> <i>SwDataDefProps</i> shall be a mandatory element for <i>ApplicationPrimitiveDataType</i> as mandatory element similar to <i>Short-Name</i> and <i>Category</i> .

Table 116 Rules

Id	Rule <i>Chapter 7.1.1.1.3.1 [AppDataType_String_Rule]</i>
[Data_130]	<b>Support for ApplicationDataType of Category STRING</b> <b>Scope:</b> AUTOSAR 4.X Support for <i>ApplicationDataType</i> Category STRING is not yet planned in AR_A2L Gen. As such developer should not use these categories in their development.

Table 117 Rules

Id	Rule <i>Chapter 7.1.1.2.1.1 [ApplDataType_Array_Rule]</i>
<i>[Data_140]</i>	<b>Defining Array Size</b> <b>Scope:</b> AUTOSAR 4.X <i>MaxNumberOfElements</i> describes the size of ARRAY, as such the attribute <i>MaxNumberOfElements</i> shall be defined for <i>Application-Array-Data-Type</i> <i>ArraySizeSemantics</i> should not be defined in the <i>ApplicationArrayDataType</i> (Could be defined in the <i>ImplementationDataType</i> ) <i>MaxNumberOfElements</i> and <i>ArraySizeSemantics</i> are defined within the <i>Element</i> attribute and is applicable to all the children of an array.

Table 118 Rules

Id	Rule <i>Chapter 7.1.1.2.3 [ApplDataType_Struct_Rule]</i>
<i>[Data_123]</i>	<b>SubElements in Structures shall have the same names in ApplicationDataType and ImplementationDataType</b> <b>Scope:</b> AUTOSAR 4.X SubElements in Structures shall have the same names in ApplicationDataType and Implementation-DataType
<i>[Data_141]</i>	<b>Definition of SwDataDefProps at SubElement Level</b> The <i>SwDataDefProps</i> of the <i>Elements</i> of an <i>ApplicationArrayDataType</i> or <i>ApplicationRecordDataType</i> shall be defined by the <i>ApplicationDataType</i> referenced by the <i>Element</i> and not by defining <i>SwDataDefProps</i> on <i>Elements</i> level itself. The definition of <i>SwDataDefProps</i> at parent element shall contain <i>SwCalibrationAccess</i> .

Table 119 Rules

Id	Rule <i>Chapter 7.1.2.1.1 [ImplDataType_Val_Rule]</i>
<i>[Data_004b]</i>	<b>ImplementationDataType</b> <b>Scope:</b> AUTOSAR 4.x <ol style="list-style-type: none"> <li>Primitive Implementation data types (e.g. sint16) shall not be defined locally as they are always central and Standardized and delivered by AUTOSAR.</li> <li>Complex Implementation data types for Maps, Curves etc, are always Product Line specific and are delivered as Central Elements. They are delivered as Central Elements of this Product Line. It is planned to have an AUTOSAR Standardized implementation data types for these objects in a future version. For further details refer to <a href="#">Document "Central Elements" [A_CEL]</a></li> <li>Other Complex Implementation data types (e.g. Structures) are defined as Module/Component specific in BSWMD/SWCD.</li> <li>ImplementationDataTypes referencing TEXTTABLE CompuMethods shall be module specific.</li> <li>---This point was removed based on the finding of Approval process.----</li> </ol>
<i>[Data_050]</i>	<b>Reuse of ImplementationDataTypes</b> <b>Scope:</b> AR4.x The reuse of ImplementationDataTypes is encouraged. Developer shall reuse the <i>Implementation-DataType</i> that are provided centrally or create module specific <i>ImplementationDataType</i> . <i>ImplementationDataType</i> defined for one specific module shall not be reused.
<i>[Data_051]</i>	<b>Reference to an ImplementationDataType</b> <b>Scope:</b> AR4.x <i>ImplementationDataType</i> shall not to be referenced directly by a <i>DataProtoType</i> , but a <i>DataProtoType</i> shall refer to an <i>ApplicationDataType</i> which in turn is mapped to <i>ImplementationDataType</i>

Table 120 Rules

Id	Rule <i>Chapter 7.1.2.2.1 [ImplDataType_Array]</i>
<a href="#">[Data_139]</a>	<b>Usage of ArraySize and ArraySizeSemantics</b>  <b>Scope:</b> AR4.x ImplementationDataTypes shall have <i>ArraySize</i> defined in <i>ImplementationDataTypeElement</i> which defines the size of Array. <i>ArraySizeSemantics</i> shall be FIXED-SIZE.

Table 121 Rules

Id	Rule <i>Chapter 7.1.3.1 [DataTypeMapping_Rule]</i>
<a href="#">[Data_127]</a>	<b>Definition of DataTypeMappingSet</b>  <b>Scope:</b> AUTOSAR 4.x DataTypeMappingSet may be defined centrally or decentral.  <ul style="list-style-type: none"><li>▶ If the referenced ApplicationDataType and the referenced ImplementationDataType are defined centrally, then DataTypeMappingSet is defined in the same AUTOSAR package as the ApplicationDataType</li><li>▶ Mapping between ApplicationDataType and ImplementationDataType shall be defined locally (Component/ Module Specific Mapping) if the ApplicationDataType and ImplementationDataType are also defined locally.</li></ul> This rule satisfies CEL_064 and can be referenced <a href="#">Document "Central Elements" [A_CEL]</a>
<a href="#">[Data_129]</a>	<b>Mapping Between ApplicationDataTypes and Implementation Data Types</b>  <b>Scope:</b> AUTOSAR 4.x For all ApplicationDataTypes defined by a component corresponding mapping to ImplementationDataType shall also be specified in the same component.

Table 122 Rules

Id	Rule <i>Chapter 7.2.1.1.1 [ParmDataPrototype_Value]</i>
<a href="#">[Data_009]</a>	<b>Usage of Shared Calibration Parameters and Per Instance Calibration Parameters</b>  <b>Scope:</b> . Applicable for AUTOSAR 4.x ApplicationSoftwareComponent developer shall decide upon when to use Shared Calibration Parameters and when to use PerInstance Calibration based on the usage. However in Basic Software, PerInstance Parameter shall be used.
<a href="#">[Data_010]</a>	<b>Removed</b>
<a href="#">[Data_011]</a>	<b>Sharing Calibration Parameters between instances of different "SwComponentType"</b>  <b>Scope:</b> Applicable for AUTOSAR 4.x  <ul style="list-style-type: none"><li>▶ For the <a href="#">Calibration parameters</a> to be visible (shared) in other SwComponentTypes, a dedicated ParameterSwComponentType that inherits from SwComponentType has to be used as a SwComponentPrototype within a CompositionSwComponentType.</li><li>▶ The ParameterSwComponentType has no InternalBehavior and employs exclusively PPortPrototypes of type ParameterInterface.</li><li>▶ Every SwComponentType requiring access to shared Calibration Parameters will have an RPort-Prototype typed by a ParameterInterface.</li></ul>
<a href="#">[Data_031]</a>	<b>Initial Value for ParameterDataPrototypes</b>  <b>Scope:</b> AUTOSAR 4.x Every ParameterDataPrototype shall have an initial value specified . This value shall be an implementation based one.

Table 123 Rules

Id	Rule <i>Chapter 7.2.1.5.1 [SwCalibAccess_Rule]</i>
<i>[Data_040]</i>	<b>Instantiating a ParameterDataPrototype with SwCalibrationAccess and SwImplPolicy</b> <b>Scope:</b> AUTOSAR 4.x A calibration parameter is instantiated with a ParameterDataPrototype class that aggregates a SwDataDefProps with properties <ul style="list-style-type: none"><li>▶ <i>swCalibrationAccess = readWrite and swImplPolicy = standard when the ParameterDataPrototype shall be visible in the A2L file and calibratable (changeable).</i></li><li>▶ <i>swCalibrationAccess = readOnly and swImplPolicy = standard when the ParameterDataPrototype shall be visible in the A2L file and not calibratable (changeable).</i></li><li>▶ <i>swCalibrationAccess = notAccessible and swImplPolicy = standard when the ParameterDataPrototype shall not be written to A2L file at all.</i></li></ul>
<i>[Data_143]</i>	<b>Calibration Access for Measurements</b> Measurement object shall only have READ-ONLY as Calibration Access

Table 124 Rules

Id	Rule <i>Chapter 7.2.2.6</i>
<i>[Data_120]</i>	<b>Storage for Logical State of Boolean Category</b> <b>Scope:</b> AR4.x Depending on the CPU direct addressing of single bits may not be available. So a byte or a word can be used to store only one logical state. (Referenced from <i>Document "Software Component Template" [TPS_SWCT]</i> .

Table 125 Rules

Id	Rule <i>Chapter 7.3.1</i>
<i>[Data_135]</i>	<b>Usage of Attributes of SwDataDefProps</b> Properties of SwDataDefProps shall be defined according to table <i>"Usage of Attributes of SwDataDefProps"</i> .
<i>[Data_136]</i>	<b>Not Used Attributes of SwDataDefProps</b> Properties of SwDataDefProps listed in table <i>"Not Used Attributes of SwDataDefProps"</i> shall not be used.
<i>[Data_137]</i>	<b>Usage of InstantiationDataDefProps</b> <i>InstantiationDataDefProps</i> shall not be used.

Table 126 Rules

Id	Rule <i>Chapter 7.5.2.5 [CompuMethod_Rule]</i>
<i>[Data_090]</i>	<b>Usage of IDENTICAL, LINEAR and RAT_FUNC in CompuMethod</b> <b>Scope:</b> AUTOSAR 4.x Developer shall use IDENTICAL CompuMethod for identical mapping and LINEAR CompuMethod shall be used where ever IDENTICAL CompuMethod is not possible to use. RAT_FUNC CompuMethod shall be used only for Rational Functions.
<i>[Data_091]</i>	<b>Referencing a CompuMethod from an ApplicationDataType</b> <b>Scope:</b> AUTOSAR 4.x CompuMethod can be referenced by an ApplicationDataType through SwDataDefProps.

Id	Rule <i>Chapter 7.5.2.5 [CompuMethod_Rule]</i>
<a href="#">[Data_092]</a>	<b>Floating numbers to be used for Computation Formulas</b> <b>Scope:</b> AUTOSAR 4.x The parameters of computation formulas have to be entered as float with a suffix "f". e.g. "5.0f" instead of "5"
<a href="#">[Data_110]</a>	<b>Concerns to Computation Method</b> <b>Scope:</b> AUTOSAR 4.x <ol style="list-style-type: none"><li>1. Every CompuMethod shall have a reference to UNIT.</li><li>2. If the <i>ApplicationDataType</i> has no physical unit, e.g. a Counter, then it shall reference to a unit by name "NoUnit".</li></ol>
<a href="#">[Data_111]</a>	<b>Removed</b>
<a href="#">[Data_113]</a>	<b>Removed</b>
<a href="#">[Data_115]</a>	<b>Conversion Direction for Category Linear</b> <b>Scope:</b> AUTOSAR 4.X "COMPU-PHYS-TO-INTERNAL" is the allowed direction of conversion for CompuMethod of category LINEAR.
<a href="#">[Data_116]</a>	<b>Removed</b>
<a href="#">[Data_117]</a>	<b>Conversion Direction for Category Rat_Func</b> <b>Scope:</b> AUTOSAR 4.X "COMPU-PHYS-TO-INTERNAL" is the allowed direction of conversion for CompuMethod of category RAT_FUNC.
<a href="#">[Data_118]</a>	<b>Conversion Direction for Category TextTable</b> <b>Scope:</b> AUTOSAR 4.X "COMPU-INTERNAL-TO-PHYS" is the allowed direction of conversion for CompuMethod of category TEXTTABLE.
<a href="#">[Data_124]</a>	<b>Definition of CompuMethod</b> <b>Scope:</b> AUTOSAR 4.X Arithmetic CompuMethods are mainly defined centrally. These includes following Categories <ul style="list-style-type: none"><li>▶ LINEAR</li><li>▶ RAT_FUNC</li><li>▶ IDENTICAL</li></ul> Verbal CompuMethods shall be defined centrally or decentral. These includes following Categories <ul style="list-style-type: none"><li>▶ TEXTTABLE</li></ul> This rule satisfies CEL_051 and CEL_052 which can be referenced in <a href="#">Document "Central Elements" [A_CEL]</a>
<a href="#">[Data_132]</a>	<b>Allowed Categories for CompuMethod</b> <b>Scope:</b> AUTOSAR 4.X All the categories defined by Autosar are not supported in BOSCH yet. Developer shall use the categories IDENTICAL, LINEAR, RAT_FUNC for their developmental activities.
<a href="#">[Data_133]</a>	<b>Removed</b>
<a href="#">[Data_145]</a>	<b>Values for Category TextTable</b> <b>Scope:</b> BOSCH The text values used in CompuMethod of category TextTable shall be unique.



Table 127 Rules

Id	Rule <i>Chapter 7.5.3.3 [Units_Rule]</i>
<i>[Data_112]</i>	<b>Every Unit shall contain FactorSiToUnit and OffsetSiToUnit</b> <b>Scope:</b> AUTOSAR 4.x <b>1.</b> UNIT shall contain FactorSiToUnit. <b>2.</b> UNIT shall contain OffsetSiToUnit. This rule satisfies CEL_028 <i>Document "Central Elements" [A_CEL]</i>
<i>[Data_121]</i>	<b>Removed</b>
<i>[Data_125]</i>	<b>Definition of Units and Unit Groups</b> <b>Scope:</b> AUTOSAR 4.X Units and UnitGroups shall be defined centrally. Developers shall not make a reference to unit directly from their data element as they are referenced by CompuMethod. This rule satisfies CEL_001 and CEL_031 which can be referenced in <i>Document "Central Elements" [A_CEL]</i>
<i>[Data_126]</i>	<b>Removed</b>

Table 128 Common Aspects of BSWMD

Id	Rule <i>Chapter 8.1</i>
<i>[BSWMD_Common_001]</i>	<b>BSWMD Standard File Header.</b> Every non-generated BSWMD ARXML file shall be headed by a standard file header. This file header shall be located directly behind the AUTOSAR schema definition.
<i>[BSWMD_Common_002]</i>	<b>Headline of a BSWMD ARXML File</b> Each BSWMD ARXML file shall start with a headline containing the definition of the used AUTOSAR schema.
<i>[BSWMD_Common_003]</i>	<b>ArPackage Hierarchy Within BSWMD Files</b> BSWMD files shall be conform to the <i>ArPackage</i> structure. That means that <ul style="list-style-type: none"><li>▶ BSW modules specified by AUTOSAR shall use /AUTOSAR_&lt;ModulePrefix&gt;/ as top level hierarchy.</li><li>▶ BSW modules not specified by AUTOSAR shall use /RB/RBA/&lt;ModulePrefix&gt;/ as top level hierarchy.</li></ul> The <ModulePrefix> represents the module prefix of the BSW module (but without a VendorId and a VendorApiInfix). In the BSWMD file the ARPackage structure shall be derived from the top level hierarchy.
<i>[BSWMD_Common_004]</i>	<b>Comments Within BSWMD Files</b> Within a BSWMD file comments may be set using the comment marker <!-- ... -->.
<i>[BSWMD_Common_005]</i>	<b>Order of Tag Sequences Within BSWMD Files</b> The tag structure and tag sequence of BSWMD syntax are shown in figures and examples. Unless otherwise specified the illustrated structure and tag sequences shall be followed.

Table 129 BswModuleDescription

Id	Rule <i>Chapter 8.2.1.1</i>
<i>[BSWMD_Module-Desc_001]</i>	<b>ShortName of ArPackage Child Element BswModuleDescription</b> The ShortName of the <i>ArPackage</i> for the <i>BswModuleDescription</i> shall be set to "BswModuleDescriptions".
<i>[BSWMD_Module-Desc_002]</i>	<b>ShortName of the BswModuleDescription</b> The ShortName of the <i>BswModuleDescription</i> shall be identical to the module prefix of the BSW module (but without a VendorId and a VendorApiInfix).
<i>[BSWMD_Module-Desc_003]</i>	<b>LongName of the BswModuleDescription</b> The LongName of the <i>BswModuleDescription</i> shall contain a meaningful description.

Id	Rule <i>Chapter 8.2.1.1</i>
<i>[BSWMD_Module-Desc_004]</i>	<b>Category of BswModuleDescription</b> Within the <i>BswModuleDescription</i> a Category has to be set using the tag <CATEGORY>. The following three settings are possible: <i>BSW_MODULE</i> , <i>BSW_CLUSTER</i> or <i>LIBRARY</i> .
<i>[BSWMD_Module-Desc_005]</i>	<b>BSW Module Identifier</b> <b>Scope:</b> Software based on an AUTOSAR SWS Within the <i>BswModuleDescription</i> a module identifier shall be set within the tag <MODULE-ID>.

Table 130 BswInternalBehavior

Id	Rule <i>Chapter 8.2.1.2</i>
<i>[BSWMD_IntBehav_001]</i>	<b>ShortName of the BswInternalBehavior</b> The ShortName of the <i>BswInternalBehavior</i> shall be set to "BswInternalBehavior".
<i>[BSWMD_IntBehav_002]</i>	<b>Reference to DataTypeMapping</b> If a BSW module specifies <i>ApplicationDataTypes</i> then a reference(s) to the <i>DataTypeMappingSet</i> (s) shall be set within the <i>BswInternalBehavior</i> . The following settings shall be made: <ul style="list-style-type: none"> <li>▶ The <i>DataTypeMapping</i> shall be set using the tag &lt;DATA-TYPE-MAPPING-REF&gt; containing the destination type "DATA-TYPE-MAPPING-SET"</li> <li>▶ The reference shall contain a complete ARPackage path to the <i>DataTypeMapping</i></li> <li>▶ The <i>DataTypeMapping</i> shall be enclosed from the tags &lt;DATA-TYPE-MAPPING-REFS&gt;</li> </ul>

Table 131 BswImplementation

Id	Rule <i>Chapter 8.2.1.3</i>
<i>[BSWMD_Impl_001]</i>	<b>ShortName of the ARPackage Child Element BswImplementation</b> The ShortName of the <i>ARPackage</i> for the <i>BswImplementation</i> shall be set to "BswImplementations".
<i>[BSWMD_Impl_002]</i>	<b>ShortName of the BswImplementation</b> The ShortName of the <i>BswImplementation</i> shall be identical to the module prefix of the BSW module.
<i>[BSWMD_Impl_003]</i>	<b>Specification of a VendorId and a VendorApiInfix</b> <b>Scope:</b> If there are multiple instances of BSW driver modules available (>1) based on AUTOSAR SWS A vendor identification shall be specified with tag <VENDOR-ID>. Additionally a specific name shall be specified as <i>VendorApiInfix</i> using the tag <VENDOR-API-IN-FIX>.
<i>[BSWMD_Impl_004]</i>	<b>Provision of a Code Descriptor</b> For each <i>BswImplementation</i> a code descriptor shall be provided. The following ARXML snippet shall be used: <pre>&lt;CODE-DESCRIPTORS&gt;   &lt;CODE&gt;     &lt;SHORT-NAME&gt;CodeDescriptor&lt;/SHORT-NAME&gt;     &lt;ARTIFACT-DESCRIPTORS&gt;       &lt;AUTOSAR-ENGINEERING-OBJECT&gt;         &lt;SHORT-LABEL&gt;ArEngObj&lt;/SHORT-LABEL&gt;         &lt;CATEGORY&gt;SWSRC&lt;/CATEGORY&gt;       &lt;/AUTOSAR-ENGINEERING-OBJECT&gt;     &lt;/ARTIFACT-DESCRIPTORS&gt;   &lt;/CODE&gt; &lt;/CODE-DESCRIPTORS&gt;</pre>
<i>[BSWMD_Impl_005]</i>	<b>Documentation of Programming Language</b> Within the <i>BswImplementation</i> the programming language shall be specified. The related tag <PROGRAMMING-LANGUAGE> shall be set to "C".

Id	Rule <i>Chapter 8.2.1.3</i>
<i>[BSWMD_Impl - 006]</i>	<b>Provision of Software Version Number</b> A software version number shall be provided within the <i>BswImplementation</i> using the tag <SW-VERSION>. The value shall be set conform to the schema "<Major>.<Minor>.<Patch>" where Major, Minor and Patch represent an integer number. The software version number shall be updated even before a new version of the software will be released.
<i>[BSWMD_Impl - 007]</i>	<b>Provision of AUTOSAR Release Version Number</b> <b>Scope:</b> Software based on an AUTOSAR SWS The inclusion of the AUTOSAR version information within the <i>BswImplementation</i> shall be given using the tag <AR-RELEASE-VERSION>. The value shall be set conform to the schema "<Major>.<Minor>.<Patch>" where Major, Minor and Patch represent an integer number. The AUTOSAR version information shall be updated even if the module supports a new version of an AUTOSAR specification.
<i>[BSWMD_Impl - 008]</i>	<b>Reference to BswInternalBehavior</b> A reference to the <i>BswInternalBehavior</i> (with a complete ARPackage path) shall be specified within the <i>BswImplementation</i> using the tag <BEHAVIOR-REF> containing the destination type "BSW-INTERNAL-BEHAVIOR".

Table 132 Splitting of BSWMD Files

Id	Rule <i>Chapter 8.2.2</i>
<i>[BSWMD_Split - 001]</i>	<b>Main Rule of Splitting of BSWMD Files</b> It is possible to create multiple BSWMD files. But there shall exist as few BSWMD files as possible for a single BSW module. Splitting of BSWMD files is optional and shall only be done if it cannot be prevented (e.g. if there is a technical reason or use case to do that).
<i>[BSWMD_Split - 002]</i>	<b>Possibility to Split BswInternalBehavior from BswModuleDescription</b> It is possible to split the <i>BswInternalBehavior</i> from the <i>BswModuleDescription</i> to a separate BSWMD file. Splitting of BSWMD files is optional and shall only be done if there is a technical reason or use case to do that. If a split is done it shall be ensured that the structure of the corresponding BSWMD files is correct: <ul style="list-style-type: none"> <li>▶ In the BSWMD file containing the <i>BswModuleDescription</i> only the relevant attributes and elements of the <i>BswModuleDescription</i> shall be specified, but without parts of the <i>BswInternalBehavior</i>.</li> <li>▶ In the BSWMD file containing the relevant attributes and elements of the <i>BswInternalBehavior</i> the <i>BswInternalBehavior</i> shall be embedded inside the <i>BswModuleDescription</i> and only the ShortName of the <i>BswModuleDescription</i> shall be repeated.</li> </ul>
<i>[BSWMD_Split - 003]</i>	<b>BswInternalBehavior is not Splitable</b> It is not allowed to split the <i>BswInternalBehavior</i> into different files. All attributes and subelements of the <i>BswInternalBehavior</i> shall be only specified in a single BSWMD file.
<i>[BSWMD_Split - 004]</i>	<b>Splitting of BswImplementation to a Single BSWMD File</b> The <i>BswImplementation</i> may be split to a separate BSWMD file and can exist in parallel to a BSWMD file containing the <i>BswInternalBehavior</i> (and/or the <i>BswModuleDescription</i> ). The split is optional and shall only be done if there is a technical reason to do that.
<i>[BSWMD_Split - 005]</i>	<b>Contents of BswImplementation are not Splitable</b> Contents of the <i>BswImplementation</i> shall not be split into different files. All attributes and elements of the <i>BswImplementation</i> shall be specified in the same BSWMD file.
<i>[BSWMD_Split - 006]</i>	<b>Splitting of Other BSWMD Specific ArPackages do Single BSWMD Files</b> <i>ArPackages</i> containing <i>ArElements</i> may be split to separate BSWMD files. The split is optional and shall only be done if there is a technical reason to do that.

Table 133 Definition of Measurement Variables

Id	Rule <i>Chapter 8.3.1.1</i>
<i>[BSWMD_MCSupport_001]</i>	<b>Definition of Measurement Variables in BSWMD</b> Measurement variables which shall exist in the A2L file shall be defined using <i>StaticMemorys</i> as part of the <i>BswInternalBehavior</i> in a BSWMD file.
<i>[BSWMD_MCSupport_002]</i>	<b>Definition of Measurement Variables in C Files</b> The representation of measurement variables shall be done in a module C file using the memory mapping concept method. It shall be ensured that the definitions in the module C file match to the definitions in the module BSWMD file (regarding data type and name of the measurement variable).
<i>[BSWMD_MCSupport_004]</i>	<b>Naming Convention for Measurement Variables</b> The ShortName of a measurement variable shall be conform to following convention: <Component-Prefix>_<pp><DescriptiveText>_MP

Table 134 Definition of Calibration Parameters

Id	Rule <i>Chapter 8.3.1.2</i>
<i>[BSWMD_MCSupport_003]</i>	<b>Definition of Calibration Parameters in BSWMD</b> Calibration parameters which shall exist in the A2L file shall be defined using <i>PerInstanceParameters</i> as part of the <i>BswInternalBehavior</i> in a BSWMD file. The definition using <i>ConstantMemorys</i> , which is still available in AUTOSAR 4.0.3, shall NOT be used. The representation of calibration data in a C file shall not be created by the SW developer because it is provided by the <i>RTE</i> generator.
<i>[BSWMD_MCSupport_005]</i>	<b>Naming Convention for Calibration Parameters</b> The ShortName of a calibration parameter shall be conform to following naming convention: <ComponentPrefix>_<pp><DescriptiveText>_EX
<i>[BSWMD_MCSupport_006]</i>	<b>Accessing a Calibration Parameter</b> To access a calibration parameter the following interface shall be used: SchM_CData_<Component-Prefix>[_<vi>_<ai>]_<Name>().
<i>[BSWMD_MCSupport_007]</i>	<b>Consideration of a Neutral Behavior of Calibration Related BSW Modules in Case of Development</b> <b>Scope:</b> BSW modules which are relevant for calibration Regarding the <i>calibration</i> a BSW module shall have a neutral behavior. This means that an update of a BSW module shall have no effect during a calibration. To ensure that the following points shall be considered: <ul style="list-style-type: none"> <li>▶ New functionalities should be encapsulated with a new calibration parameter to be able to switch of the new functionality.</li> <li>▶ New or extended calibration parameters shall be provided in such a way that their initial effect for the calibration is neutral. This shall be done by setting neutral initialization values whenever it is possible.</li> <li>▶ An explanation shall be given to the delivery note to document the necessary calibration changes.</li> </ul>

Table 135 BswCalledEntity

Id	Rule <i>Chapter 8.3.2.1.1</i>
<i>[BSWMD_Sched_001]</i>	<b>Definition of BswCalledEntity</b> For each BSW Called Entity, a <b>BswCalledEntity</b> element shall be defined in a <b>BswInternalBehavior</b> and a <b>BswModuleEntry</b> shall be defined. The two model elements shall exist.

Table 136 BswInterruptEntity

Id	Rule <i>Chapter 8.3.2.1.2</i>
[BSWMD_Sched_002]	<b>Definition of BswInterruptEntity</b> For each BSW Interrupt Entity, a <b>BswInterruptEntity</b> element shall be defined in a <b>BswInternalBehavior</b> and a <b>BswModuleEntry</b> shall be defined. The two model elements shall exist.

Table 137 BswSchedulableEntity

Id	Rule <i>Chapter 8.3.2.1.3</i>
[BSWMD_Sched_003]	<b>Definition of BswSchedulableEntity</b> For each BswSchedulableEntity, a <b>BswSchedulableEntity</b> element shall be defined in a <b>BswInternalBehavior</b> and a <b>BswModuleEntry</b> shall be defined. The two model elements shall exist.
[BSWMD_Sched_004]	<b>Definition of BswEvents</b> For each BswEvent, an <b>Events</b> element shall be defined in a <b>BswInternalBehavior</b> . The concerned BswSchedulableEntity shall be referenced in a <b>StartsOnEventRef</b> . The two model elements shall exist.

Table 138 BSW Modes

Id	Rule <i>Chapter 8.3.2.2</i>
[BSWMD_Mode_001]	<b>Definition of BSW Modes</b> Each BSW Mode shall be defined in a BSWMD ARXML using <b>ModeDeclaration</b> under <b>ModeDeclarations</b> . The <b>InitialModeRef</b> shall also be added to have the complete <b>ModeDeclarationGroup</b> . All these model elements shall exist.

Table 139 BSW Mode Management on the Mode Manager side

Id	Rule <i>Chapter 8.3.2.2.1</i>
[BSWMD_Mode_002]	<b>Definition of BSW Mode Management on the Mode Manager side</b> For BSW Modes on the Mode Manager side the <b>ProvidedModeGroups</b> shall be defined in a BSWMD ARXML file. The <b>ModeDeclarationGroupPrototype</b> shall be one element of the <b>ProvidedModeGroups</b> . The <b>ManagedModeGroups</b> shall also be defined in the <b>BswModuleDescription</b> with a reference to the <b>ModeDeclarationGroupPrototype</b> . All these model elements shall exist.

Table 140 BSW Mode Management on the Mode User side

Id	Rule <i>Chapter 8.3.2.2.2</i>
[BSWMD_Mode_003]	<b>Definition of BSW Mode Management on the Mode User side</b> For BSW Modes on the Mode User side the <b>RequiredModeGroups</b> shall be defined in the <b>BswModuleDescription</b> . The <b>ModeDeclarationGroupPrototype</b> shall be one element of the <b>RequiredModeGroups</b> . A reference to the <b>ModeDeclarationGroup</b> shall also be defined in the <b>TypeTref</b> of the <b>ModeDeclarationGroupPrototype</b> . All these model elements shall exist.
[BSWMD_Mode_004]	<b>Definition of BswModeSwitchEvent</b> For each BSW Mode Switch Event, a <b>BswModeSwitchEvent</b> element shall be defined in a <b>BswInternalBehavior</b> . The type of activation shall be defined in <b>Activation</b> , a reference to the <b>ModeDeclarationGroupPrototype</b> shall be defined in <b>Modelref</b> . The concerned Mode(s) shall be referenced in a <b>TargetModeRef</b> . All these model elements shall exist.

Table 141 BSW Mode Access

Id	Rule <i>Chapter 8.3.2.2.3</i>
<i>[BSWMD_Mode_-005]</i>	<b>Definition of BSW Mode Access</b> If a BswSchedulableEntity needs to access to the current Mode the <b>AccessedModeGroups</b> shall be defined in the <b>BswSchedulableEntity</b> with a reference to the <b>ModeDeclarationGroupPrototype</b> .

Table 142 BswExternalTriggers on the Sender side

Id	Rule <i>Chapter 8.3.2.3.1</i>
<i>[BSWMD_Trigger_-001]</i>	<b>Definition of BswExternalTriggers on the Sender side</b> For each External Trigger, a <b>Trigger</b> element shall be defined in a <b>ReleasedTrigger</b> element of a <b>BswModuleDescription</b> . For the BswSchedulableEntity, a reference to the ExternalTrigger shall be defined in <b>TriggerRef-Conditional</b> element in the <b>IssuedTrigger</b> element of the <b>BswSchedulableEntity</b> . All these model elements shall exist.

Table 143 BswExternalTriggers on the Receiver side

Id	Rule <i>Chapter 8.3.2.3.2</i>
<i>[BSWMD_Trigger_-002]</i>	<b>Definition of BswExternalTriggers on the Receiver side</b> For each External Trigger, a <b>Trigger</b> element shall be defined in a <b>RequiredTrigger</b> element of a <b>BswModuleDescription</b> .
<i>[BSWMD_Trigger_-003]</i>	<b>Definition of BSWExternalTriggerOccuredEvent</b> For each BSW External Trigger Occured Event, a <b>BswExternalTriggerOccuredEvent</b> element shall be defined in an <b>Events</b> element of a <b>BswInternalBehavior</b> . The concerned BswSchedulableEntity shall be referenced in a <b>StartsOnEventRef</b> . All these model elements shall exist.

Table 144 BswInternalTriggers

Id	Rule <i>Chapter 8.3.2.3.3</i>
<i>[BSWMD_Trigger_-004]</i>	<b>Definition of BswInternalTriggers</b> For each Internal Trigger, a <b>BswInternalTriggeringPoint</b> element shall be defined in a <b>InternalTriggeringPoints</b> element of a <b>BswInternalBehavior</b> .
<i>[BSWMD_Trigger_-005]</i>	<b>Definition of BswInternalTriggerOccuredEvent</b> For each BSW Internal Trigger Occured Event, a <b>BswInternalTriggerOccuredEvent</b> element shall be defined in an <b>Events</b> element of a <b>BswInternalBehavior</b> . For the BswSchedulableEntity to be activated by the BSW Internal Trigger Occured Event, a reference to Bsw Internal Triggering Point shall be defined in the <b>BswInternalTriggeringPointRefConditional</b> of a <b>ActivationPoint</b> element of a <b>BswSchedulableEntity</b> . The concerned BswSchedulableEntity shall be referenced in a <b>StartsOnEventRef</b> . All these model elements shall exist.

Table 145 Mapping between BSW and a corresponding SWC

Id	Rule <i>Chapter 8.3.2.4</i>
<i>[BSWMD_SWC_-001]</i>	<b>Definition of SwcBswRunnableMapping</b> For each SwcBswRunnable Mapping, a <b>SwcBswRunnableMappings</b> element shall be defined, a reference to a BswSchedulableEntity shall be defined in <b>BswEntityRef</b> and a reference to the Runnable Entity shall be defined in a <b>SwcRunnableRef</b> . A reference to a Bsw Internal Behavior shall be defined in <b>BswBehaviorRef</b> . And a reference to the Swc Internal Behavior shall be defined in a <b>SwcRunnableRef</b> . All these element shall be defined in a <b>SwcBswMapping</b> . <b>BswModuleEntry</b> shall be defined. Mapping between BSW and SWC shall be defined with the <b>Swc-BswMapping</b> element in an ARXML file according to AUTOSAR specifications. The Runnable mappings shall be defined in a <b>SwcBswRunnableMappings</b> .



Id	Rule <i>Chapter 8.3.2.4</i>
<i>[BSWMD_SWC_-002]</i>	<b>Referencing a SwcBswMapping in a BSWMD ARXML file</b> If a Runnable Mapping has been defined in a <b>SwcBswRunnableMappings</b> element of a <b>SwcBswMapping</b> , a reference to the Swc Bsw Mapping shall be added in a <b>SwcBswMappingRef</b> element of the <b>BswImplementations</b> of the concerned BSWMD. The same kind of reference shall be added for the concerned SWC module.
<i>[BSWMD_SWC_-003]</i>	<b>Definition of SynchronizedModeGroups between SWC and BSW</b> For each Synchronized Mode Groups a <b>SwcBswSynchronizedModeGroupPrototype</b> element shall be defined, For BSW, a reference to a Mode Declaration Group Prototype shall be defined in <b>BswModeGroupIref</b> . For SWC, a reference to the provided port used by the SWC in a <b>ContextPPortRef</b> and a reference to the Mode Declaration Group Prototype shall be defined in <b>SwcModeGroupIref</b> . All these model elements shall be defined in a <b>SynchronizedModeGroups</b> element of a <b>SwcBswMapping</b> .
<i>[BSWMD_SWC_-004]</i>	<b>Definition of SynchronizedTriggers between SWC and BSW</b> For each Synchronized Triggers a <b>SwcBswSynchronizedTrigger</b> element shall be defined, For BSW, a reference to a Bsw Trigger shall be defined in <b>BswTriggerRef</b> For SWC, a reference to the provided port used by the Swc in a <b>ContextPPortRef</b> and a reference to the trigger used by the SWC in a <b>TargetTriggerRef</b> . All these model elements shall be defined in a <b>SynchronizedTriggers</b> element of a <b>SwcBswMapping</b> .

Table 146 Common Attributes of a BswModuleEntry

Id	Rule <i>Chapter 8.3.4.1</i>
<i>[BSWMD_Entry_-001]</i>	<b>ShortName of the ArPackage Kind Element BswModuleEntry</b> The ShortName of the <i>ArPackage</i> for the <i>BswModuleEntry</i> shall be set to "BswModuleEntry".
<i>[BSWMD_Entry_-002]</i>	<b>ShortName of the BswModuleEntry</b> The ShortName of the <i>BswModuleEntry</i> shall be set identical to the name of the C-function (but without a VendorId and a VendorApiInfix in the BSW module prefix).
<i>[BSWMD_Entry_-003]</i>	<b>Serviceld of the BswModuleEntry</b> <b>Scope:</b> Standardized Interfaces specified in an AUTOSAR SWS In the tag <SERVICE-ID> the service identifier of the Standardized Interfaces specified in an AUTOSAR SWS shall be entered.
<i>[BSWMD_Entry_-004]</i>	<b>Reentrancy of a BswModuleEntry</b> The tag <IS-REENTRANT> shall be used to describe the reentrancy of the specified service. Following two settings are possible: <i>true</i> or <i>false</i>
<i>[BSWMD_Entry_-005]</i>	<b>Synchronity of a BswModuleEntry</b> The tag <IS-SYNCHRONOUS> shall be used to describe the synchronity of the specified service. Following two settings are possible: <i>true</i> or <i>false</i>
<i>[BSWMD_Entry_-006]</i>	<b>Call Type of the BswModuleEntry</b> With the tag <CALL-TYPE> the type of call associated with this specified service shall be set. Following settings are possible: <i>CALLBACK</i> , <i>INTERRUPT</i> , <i>REGULAR</i> or <i>SCHEDULED</i> .
<i>[BSWMD_Entry_-007]</i>	<b>Execution Context of the BswModuleEntry</b> With the tag <EXECUTION-CONTEXT> the execution context required or guaranteed for the call associated with this specified service shall be set. Following settings are possible: <i>HOOK</i> , <i>INTERRUPT-CAT-1</i> , <i>INTERRUPT-CAT-2</i> , <i>TASK</i> or <i>UNSPECIFIED</i> .

Id	Rule <i>Chapter 8.3.4.1</i>
<i>[BSWMD_Entry-008]</i>	<b>SwServiceImplPolicy of the BswModuleEntry</b> With the tag <SW-SERVICE-IMPL-POLICY> the implementation policy of the specified service shall be documented. Following settings are possible: <i>INLINE</i> , <i>INLINE-CONDITIONAL</i> , <i>MACRO</i> or <i>STANDARD</i> .

Table 147 Definition of a Return Value and Arguments of a BswModuleEntry

Id	Rule <i>Chapter 8.3.4.2</i>
<i>[BSWMD_Entry-RetArg_001]</i>	<b>ShortName of a Return Value of a BswModuleEntry</b> The ShortName of a return value of a <i>BswModuleEntry</i> shall be set to 'ReturnValue'.
<i>[BSWMD_Entry-RetArg_002]</i>	<b>ShortName of an Argument of a BswModuleEntry</b> The ShortName of an argument of a <i>BswModuleEntry</i> shall be set identical to the name of the argument which is used in the C code or specified in the AUTOSAR specification.
<i>[BSWMD_Entry-RetArg_003]</i>	<b>Category of a Return Value or an Argument of a BswModuleEntry</b> With the tag <CATEGORY> the kind of the return value or the argument shall be specified (value, data pointer, function pointer). Following categories are possible: <i>TYPE_REFERENCE</i> , <i>DATA_REFERENCE</i> or <i>FUNCTION_REFERENCE</i> . Dependent on a specific category appropriate <i>SwDataDefProps</i> shall be specified.
<i>[BSWMD_Entry-RetArg_004]</i>	<b>Direction of a Return Type or an Argument of a BswModuleEntry</b> With the tag <DIRECTION> the kind of the data flow of the return value or the argument of a <i>BswModuleEntry</i> may be specified. The definition of the direction is optional. Following values are possible: <i>IN</i> , <i>INOUT</i> or <i>OUT</i> . A return value can only have the direction 'OUT'. The direction of an argument can differ corresponding to the kind of the argument. A value argument has always the direction 'IN' while a pointer can have the direction 'IN', 'INOUT' or 'OUT' depending on the usage of the pointer argument.
<i>[BSWMD_Entry-RetArg_005]</i>	<b>Definition of a Value as Return Value or Argument of a BswModuleEntry</b> To specify a value as return value or argument of a <i>BswModuleEntry</i> following settings shall be made: <ul style="list-style-type: none"> <li>▶ A reference to an <i>ImplementationDataType</i> using the tag &lt;IMPLEMENTATION-DATA-TYPE-REF&gt; shall be set within the <i>SwDataDefProps</i> tag structure</li> <li>▶ The destination type of the <i>ImplementationDataTypeRef</i> shall be set to 'IMPLEMENTATION-DATA-TYPE'</li> <li>▶ A complete path to a centrally or locally defined <i>ImplementationDataType</i> shall be set</li> <li>▶ The category of the return value or argument shall be set to 'TYPE-REFERENCE' (conform to rule <i>[BSWMD_EntryRetArg_003]</i> )</li> </ul>
<i>[BSWMD_Entry-RetArg_006]</i>	<b>Definition of a Data Pointer as Return Value or Argument of a BswModuleEntry</b> To specify a data pointer as return value or argument of a <i>BswModuleEntry</i> following settings shall be made: <ul style="list-style-type: none"> <li>▶ A reference to an <i>ImplementationDataType</i> using the tag &lt;IMPLEMENTATION-DATA-TYPE-REF&gt; shall be set within an inner <i>SwDataDefProps</i> tag structure inside of <i>SwPointerTargetProps</i> which are part of an outer <i>SwDataDefProps</i> tag structure of the return value or argument definition</li> <li>▶ The destination type of the <i>ImplementationDataTypeRef</i> shall be set to 'IMPLEMENTATION-DATA-TYPE'</li> <li>▶ A complete path to a centrally or locally defined <i>ImplementationDataType</i> shall be set</li> <li>▶ The category of the return value or argument shall be set to 'DATA-REFERENCE' (conform to rule <i>[BSWMD_EntryRetArg_003]</i> )</li> </ul>



Id	Rule <i>Chapter 8.3.4.2</i>
[BSWMD_Entry- RetArg_007]	<p><b>Definition of a Void Pointer as Return Value or Argument of a BswModuleEntry</b></p> <p>To specify a void pointer as return value or argument of a <i>BswModuleEntry</i> following settings shall be made:</p> <ul style="list-style-type: none"> <li>▶ A reference to a <i>SwBaseType</i> using the tag &lt;BASE-TYPE-REF&gt; shall be set within an inner <i>SwDataDefProps</i> tag structure inside of <i>SwPointerTargetProps</i> which are part of an outer <i>SwDataDefProps</i> tag structure of the return value or argument definition</li> <li>▶ The destination type of the <i>ImplementationDataTypeRef</i> shall be set to 'SW-BASE-TYPE'</li> <li>▶ The path '/AUTOSAR_Platform/SwBaseTypes/void' shall be set</li> <li>▶ The category of the return value or argument shall be set to 'DATA-REFERENCE' (conform to rule [BSWMD_EntryRetArg_003] )</li> </ul>
[BSWMD_Entry- RetArg_008]	<p><b>Definition of a Function Pointer as Return Value or Argument of a BswModuleEntry</b></p> <p>To specify a function pointer as return value or argument of a <i>BswModuleEntry</i> following settings shall be made:</p> <ul style="list-style-type: none"> <li>▶ A reference to a <i>BswModuleEntry</i> using the tag &lt;FUNCTION-POINTER-SIGNATURE-REF&gt; shall be set within <i>SwPointerTargetProps</i> which are part of the <i>SwDataDefProps</i> tag structure of the return value or argument definition</li> <li>▶ The destination type of the <i>FunctionPointerSignatureRef</i> shall be set to 'BSW-MODULE-ENTRY'</li> <li>▶ A complete path to a <i>BswModuleEntry</i> shall be set</li> <li>▶ The category of the return value or argument shall be set to 'FUNCTION-REFERENCE' (conform to rule [BSWMD_EntryRetArg_003] )</li> </ul>
[BSWMD_Entry- RetArg_009]	<p><b>Definition of an Implementation Policy for Return Values and Arguments of a BswModuleEntry</b></p> <p>Within the <i>SwDataDefProps</i> an implementation policy shall be specified using the tag &lt;SW-IMPL--POLICY&gt;.</p> <p>Following values are possible: <i>STANDARD</i> or <i>CONST</i>.</p> <p>The tag &lt;SW-IMPL-POLICY&gt; is mandatory on each level of <i>SwDataDefProps</i>.</p> <p>The implementation policy 'CONST' is only relevant in context of data pointers. In all other cases (values, function pointers) the implementation policy shall be set to 'STANDARD'.</p>
[BSWMD_Entry- RetArg_010]	<p><b>Definition of an Additional Native Type Qualifier for Return Values and Arguments of a BswModuleEntry</b></p> <p>Within the <i>SwDataDefProps</i> an additional type qualifier (e.g. "volatile") may be specified using the tag &lt;ADDITIONAL-NATIVE-TYPE-QUALIFIER&gt;.</p>
[BSWMD_Entry- RetArg_011]	<p><b>Definition of a Target Category for Data Pointers</b></p> <p><b>Scope:</b> Only relevant for the definition of data pointers</p> <p>Within the inner <i>SwDataDefProps</i> the category of the target shall be specified using the tag &lt;TARGET-CATEGORY&gt;. The category of the referenced data shall be set.</p> <p>Following values are possible: <i>VALUE</i>, <i>ARRAY</i> or <i>STRUCTURE</i></p>
[BSWMD_Entry- RetArg_012]	<p><b>Definition of Return Values and Arguments if the BswModuleEntry is a Macro</b></p> <p>If the <i>BswModuleEntry</i> represents a macro the return type and the arguments shall be defined without <i>SwDataDefProps</i>. Only the common arguments are relevant.</p>
[BSWMD_Entry- RetArg_013]	<p><b>BswModuleEntry with no Return Value</b></p> <p>In case of an empty return type ("void" in C) no return value shall be specified within the <i>BswModuleEntry</i>. The &lt;RETURN-TYPE&gt; tag structure shall not be set.</p>
[BSWMD_Entry- RetArg_014]	<p><b>BswModuleEntry with no Argument</b></p> <p>In case of an empty argument list ("void" in C) no arguments shall be specified within the <i>BswModuleEntry</i>. The &lt;ARGUMENT&gt; tag structure shall not be set.</p>

Table 148 Main Structure of *ARPackages*

Id	Rule <i>Chapter 9.2 [ARPackage-1]</i>
<a href="#">[ARPac_10]</a>	<p><b>Main Pattern for Package Structure</b></p> <p><b>Scope:</b> RB deliverables, except AUTOSAR specified software (e.g. BSW modules, Application Interfaces (AISpecification))</p> <p>The following structure should be followed:</p> <pre> /RB   /{domain}     [{/subdomain}] *       /{block}         /{kind} [_Blueprint _Example] </pre> <p>where</p> <ul style="list-style-type: none"> <li>▶ RB is the top level <i>ARPackage</i> for RB, see <a href="#">[ARPac_11]</a></li> <li>▶ domain is an ECU domain, such as PT (power train), RBA (RB's first implementation of AUTOSAR BSW: CUBAS), see <a href="#">[ARPac_12]</a></li> <li>▶ subdomain gives the possibility to create a structure of sub domains</li> <li>▶ block is the name of a component type or a BSW module or "CentralElements" or similar</li> <li>▶ kind denotes the kind of the <i>ARElement</i>(s) which are contained, see <a href="#">[ARPac_14]</a></li> </ul> <p>For an AUTOSAR specified BSW module there is another package structure to be used. See rule <a href="#">[ARPac_41]</a>.</p> <p>For ECU Configuration values a special <i>ARPackage</i>-Structure is to be used ("/RB/UBK/Project/..."). See rule <a href="#">[ARPac_46]</a> and rule <a href="#">[ARPac_52]</a>.</p>
<a href="#">[ARPac_11]</a>	<p><b>Top Level ARPackage</b></p> <p><b>Scope:</b> RB deliverables, except AUTOSAR specified software (e.g. BSW modules)</p> <p>There shall be exactly one top level <i>ARPackage</i> in an .arxml file.</p> <p>The top level package shall be named "RB".</p>
<a href="#">[ARPac_12]</a>	<p><b>Domain oriented ARPackages</b></p> <p><b>Scope:</b> RB deliverables, except AUTOSAR specified BSW modules</p> <p>The package name on second level should be the name of the ECU-domain, as defined in <a href="#">Table 45</a>.</p> <p>No names derived from organisations should be used (e.g. DGS, CC-DA).</p> <p>The name "AUTOSAR" and all names beginning with "AUTOSAR" are reserved.</p> <p>If required subdomains may be defined.</p>
<a href="#">[ARPac_13]</a>	<p><b>Removed</b></p>
<a href="#">[ARPac_14]</a>	<p><b>Names for ARElement-kind based ARPackages</b></p> <p><b>Scope:</b> AR4.x</p> <p>Whenever (sub-) <i>ARPackages</i> are created for the different kinds of <i>ARElements</i>, <i>ARPackage</i>'s <i>Short-Name</i> shall be derived from the kind, concatenated by a plural "s". In most cases this is the name of the direct subclass of <i>ARElement</i> or <i>FibexElement</i>. Complete List is given in <a href="#">Table 46 [Recommended Packages]</a></p> <p>If the <i>ARPackage</i> contains <i>blueprint elements</i> or <i>example elements</i> then "_Blueprint" resp. "_Example" has to be added to this name (e.g. "ApplicationDataTypes_Blueprint").</p> <p>Note: the usage of <i>examples</i> is currently not defined in our methods and shall not be used. Details will be defined in <a href="#">[A_Data]</a> or other guidelines.</p>
<a href="#">[ARPac_14A]</a>	<p><b>Sort kind-based ARPackages alphabetically</b></p> <p>For better readability the kind-based <i>ARPackages</i> should be sorted alphabetically.</p>
<a href="#">[ARPac_15]</a>	<p><b>Omit empty ARPackages</b></p> <p>If an <i>ARPackage</i> contains no <i>ARElements</i> you may omit it.</p>

Id	Rule <i>Chapter 9.2 [ARPackage-1]</i>
<i>[ARPac_16]</i>	<b>ARPackage for Common ARElements</b> Scope: AR4.x There are different kinds of common <i>ARElements</i> . They shall be covered by a <i>ARPackage</i> named "-Common". The actually known common element types are: <i>CentralElements</i> , <i>DesignPatterns</i> , <i>DocumentationPatterns</i> , <i>NamePatterns</i> , <i>PackagePatterns</i> , <i>SwArchitecture</i> , <i>NamingConventions</i> . The Common- <i>ARPackages</i> shall be used on that <i>ARPackage</i> -level for which the elements are common. This can be the level "RB" or the level of a domain or of a sub domain.
<i>[ARPac_17]</i>	<b>Category of ARPackages</b> The following <i>Category</i> s for <i>ARPackage</i> are predefined by AUTOSAR: <ul style="list-style-type: none"> <li>▶ <i>BLUEPRINT</i> (blueprint elements)</li> <li>▶ <i>STANDARD</i> (standardized objects)</li> <li>▶ <i>ICS</i> (Implementation Conformance Statement)</li> <li>▶ <i>EXAMPLE</i> (examples how to apply Elements in STANDARD or BLUEPRINT packages)</li> </ul> You shall NOT define any other category for <i>ARPackages</i> .
<i>[ARPac_18]</i>	<b>Empty Category of ARPackages</b> For <i>ARPackages</i> which contain model elements of none of the categories <i>BLUEPRINT</i> , <i>STANDARD</i> , <i>ICS</i> , <i>EXAMPLE</i> you shall NOT set any category. <i>Category</i> attribute shall be omitted. This is the standard case for most <i>ARPackages</i> .
<i>[ARPac_191]</i>	<b>Category BLUEPRINT for ARPackages</b> The <i>Category</i> of <i>ARPackages</i> containing blueprint elements shall be set to <i>BLUEPRINT</i> .
<i>[ARPac_192]</i>	<b>Category STANDARD for ARPackages</b> The <i>Category</i> of <i>ARPackages</i> containing standardized elements shall be set to <i>STANDARD</i> . This <i>Category</i> is typically created only by architects.
<i>[ARPac_20]</i>	<b>Category ICS, EXAMPLE for ARPackages</b> The <i>Category</i> s <i>ICS</i> , <i>EXAMPLE</i> shall NOT be used since the usage is not defined in our methods.

Table 149 ARPackage for Basic Software

Id	Rule <i>Chapter 9.3 [ARPackage-BSW]</i>
<i>[ARPac_41]</i>	<b>Top level ARPackages for AUTOSAR Specified BSW Modules</b> <b>Scope:</b> BSW modules specified by AUTOSAR Each basic software module, if contained in AUTOSAR's BSW module list ( <i>[TR_BSWModuleList]</i> ) or in AUTOSAR's list of "virtual modules" ( <i>TR_PDN_00001</i> in <i>[TR_PDN]</i> ), shall define its own <i>ARPackage</i> as top level <i>ARPackage</i> . The <i>ShortName</i> shall be "AUTOSAR_" + {moduleName}. List of BSW-Modules is given in table <i>Table 47 [BSWModuleNames]</i> , taken from AUTOSAR specification 4.1.1.
<i>[ARPac_43]</i>	<b>Top level ARPackages for non AUTOSAR Specified BSW Modules</b> <b>Scope:</b> BSW modules not specified by AUTOSAR Such modules shall be put into the top level <i>ARPackage</i> "RB". They shall be put into the second level <i>ARPackage</i> for the concerning domain. This <i>ARPackage</i> shall be either "RBA" if it is related to RBA, otherwise the ECU domain (e.g. PT for power train). The <i>ARPackage</i> on the next level shall get the name of the concerning BSW module.
<i>[ARPac_44]</i>	<b>Kind ARPackages for Basic Software Modules</b> For BSW modules the rule " <i>Rule ARPac_14: Names for ARElement-kind based ARPackages</i> " shall be followed. It shall be followed by AUTOSAR specified and non AUTOSAR specified BSW modules.

Id	Rule <i>Chapter 9.3 [ARPackage-BSW]</i>
<a href="#">[ARPac_45]</a>	<p><b>Software Module Configuration (preconfigured or recommended values) -- currently NOT used</b></p> <p><b>Scope:</b> upcoming</p> <p>Preconfigured or recommended ecuc values for BSW modules (AUTOSAR specified or not) shall be put into the <i>ARPackage</i> of the module itself and there in a sub package <i>EcucModuleConfiguration-Values</i>.</p> <p>Examples:</p> <pre>/AUTOSAR_LinSM/EcucModuleConfigurationValues /RB/RBA/rba_Bernd/EcucModuleConfigurationValues</pre> <p>For preconfigured values the container's <i>ShortName</i> shall be <i>PreconfiguredValues</i>.</p> <p>For recommended values the container's <i>ShortName</i> shall be <i>RecommendedValues</i>.</p>
<a href="#">[ARPac_46]</a>	<p><b>Software Module Configuration (for protected values)</b></p> <p>Protected configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following <i>ARPackage</i> structure:</p> <pre>/RB   /UBK     /Project       /EcucModuleConfigurationValues</pre> <p><b>Note:</b> the word "Project" is a keyword and shall not be replaced by any project's name. Reason: Tools will merge all configuration values coming from different files only if they are elements of the same <i>ARPackage</i>.</p> <p>Protected configuration values shall <b>NOT</b> be put into the BSW module's <i>ARPackage</i> (e.g. <i>/AUTOSAR_LinSM</i> or <i>/RB/RBA/RBA_Bernd</i>).</p>
<a href="#">[ARPac_47]</a>	<b>removed</b>

Table 150 *ARPackage* for Project based *ARElements*

Id	Rule <i>Chapter 9.4 [ARPackage-Prj]</i>
<a href="#">[ARPac_50]</a>	<b>removed</b>
<a href="#">[ARPac_50v1]</a>	<p><b>ARPackages for Projects</b></p> <p><i>ARPackages</i> for projects mainly shall follow <a href="#">[ARPac_10]</a>.</p> <p>The contents of {domain} shall be set to the fixed word <i>UBK</i>.</p> <p>The contents of {block} shall be set to the fixed word <i>Project</i>.</p> <p>There are some other rules in this chapter which define some exceptions.</p>
<a href="#">[ARPac_51]</a>	<p><b>ARPackage for top level component type</b></p> <p>The name for {block}, as defined in <a href="#">[ARPac_10]</a> shall be set to the name of the top level composition type. This name will be defined by ECU software architects, e.g. to <i>RootSwCompo</i>.</p> <p>A prefix may be added (e.g. <i>PCT_</i> for DGS's power train).</p> <p>Typically there will be some sub- <i>ARPackages</i>:</p> <ul style="list-style-type: none"> <li>▶ <i>SwComponentTypes</i> for the component type</li> <li>▶ <i>FlatMaps</i> for the flat map</li> <li>▶ <i>CalibrationParameterValueSets</i></li> </ul> <p>Also for the ECU flat view composition a name may be defined by ECU software architects, e.g. to <i>EcuFlatView</i>.</p>

Id	Rule <i>Chapter 9.4 [ARPackage-Prj]</i>
<a href="#">[ARPac_52]</a>	<b>ECUC Software Module Configuration</b> Configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following <i>ARPackage</i> structure: <pre>           /RB             /UBK               /Project                 /EcucModuleConfigurationValues           </pre> <b>Note:</b> the word "Project" is a keyword and shall not be replaced by the project's name. Reason: Tools will merge all configuration values coming from different files only if they are elements of the same <i>ARPackage</i> .
<a href="#">[ARPac_53]</a>	<b>ARPackage for Configuration of ECU, RTE, OS</b> The configuration values for ECU, RTE, OS shall be handled as defined in <a href="#">[ARPac_52]</a> .
<a href="#">[ARPac_54]</a>	<b>ARPackage for AliasNameSets</b> The <i>ARPackage</i> for <i>AliasNameSets</i> typically shall reside in the <i>ARPackage</i> of the project as defined in Rule <a href="#">[ARPac_50v1]</a> .
<a href="#">[ARPac_55]</a>	<b>System and ECU based Elements</b> Elements based on a system or ECU should use <i>ARPackage</i> names which contain <i>SYS</i> resp. <i>ECU</i> . A prefix <i>PCFG_</i> (for project configuration) may be used. Examples are packages for the hardware topology, system extract, RTE container tasks. Detailed rules are defined in <a href="#">[G_Integ]</a> .

Table 151 Use of *ReferenceBase*

Id	Rule <i>Chapter 9.5.2 [ARPackage-RefBase]</i>
<a href="#">[ARPac_71]</a>	<b>Avoid usage of Element ReferenceBase</b> The usage of the Element <i>ReferenceBase</i> is not yet supported by all authoring tools and should NOT be used. Exceptions are defined in subsequent rules.
<a href="#">[ARPac_72]</a>	<b>Usage of ReferenceBase</b> In tool generated arxml files <i>ReferenceBase</i> should be used for making the files more readable.
<a href="#">[ARPac_73]</a>	<b>A defined ReferenceBase shall always be used</b> If a <i>ReferenceBase</i> to an <i>ARPackage</i> is defined it shall be used in <b>all</b> references going to the elements in this <i>ARPackage</i> , including sub- <i>ARPackages</i> .
<a href="#">[ARPac_74]</a>	<b>ShortName of ReferenceBases</b> <i>ShortName</i> of a <i>ReferenceBase</i> shall follow the following pattern: { <i>Scope</i> }_{ <i>ElementKind</i> } where <i>Scope</i> is the scope wherefore the objects are defined (e.g. the delivery package) and <i>ElementKind</i> shall be taken out of the list as defined in Rule <a href="#">[ARPac_14]</a> .
<a href="#">[ARPac_75]</a>	<b>ReferenceBases and their usage in references shall reside in the same delivery unit</b> The definition of a <i>ReferenceBase</i> shall be in the same file as their usage in references. <b>Exception:</b> the <i>ReferenceBase</i> for referencing central elements may be defined in central file(s).

Table 152 Units

Id	Rule <i>Chapter 10.1.1 [CenElemChapter_1]</i>
<a href="#">[CEL_001]</a>	<b>Unit</b> The following AUTOSAR elements shall be defined centrally: ► Unit

Id	Rule <i>Chapter 10.1.1 [CenElemChapter_1]</i>
<i>[CEL_028]</i>	<b>Properties of Units</b> Every AUTOSAR Unit shall have: <ul style="list-style-type: none"> <li>▶ FactorSiToUnit</li> <li>▶ OffsetSiToUnit</li> <li>▶ Reference to PhysicalDimension</li> <li>▶ DisplayUnit</li> </ul>
<i>[CEL_031]</i>	<b>UnitGroup</b> The following AUTOSAR elements shall be defined centrally or locally: <ul style="list-style-type: none"> <li>▶ UnitGroup</li> </ul>
<i>[CEL_041]</i>	<b>PhysicalDimension</b> The following AUTOSAR elements shall be defined centrally: <ul style="list-style-type: none"> <li>▶ PhysicalDimension</li> </ul>
<i>[CEL_071]</i>	<b>Unit: Accuracy of FactorSiToUnit and OffsetSiToUnit</b> FactorSiToUnit and OffsetSiToUnit shall use the same accuracy (as specified from the responsible AUTOSAR working group). If a unit has not yet been specified by AUTOSAR the accuracy shall be set to 8 digits. Counted are the digits before and after the dot. Leading zeroes are not counted.

Table 153 CompuMethods

Id	Rule <i>Chapter 10.1.2 [CenElemChapter_2]</i>
<i>[CEL_051]</i>	<b>Arithmetic CompuMethod</b> The following AUTOSAR elements shall be defined mainly centrally: <ul style="list-style-type: none"> <li>▶ Arithmetic CompuMethod (categories: RAT_FUNC, LINEAR, IDENTICAL)                In special cases, e.g. conversions for complex drivers, the CompuMethod may be allocated locally.</li> </ul>
<i>[CEL_052]</i>	<b>Verbal CompuMethod</b> The following AUTOSAR elements shall be defined centrally or locally: <ul style="list-style-type: none"> <li>▶ CompuMethod of category TEXTTABLE</li> </ul>
<i>[CEL_055]</i>	<b>Referencing of Units by CompuMethod</b> Arithmetic and verbal CompuMethods shall refer to an Unit. If the ComputMethod has no Unit, the Unit with the shortname <i>NoUnit</i> shall be referred.

Table 154 System Constants

Id	Rule <i>Chapter 10.1.3 [CenElemChapter_3]</i>
<i>[CEL_002]</i>	<b>SwSystemconst</b> The following AUTOSAR elements may be defined centrally or locally: <ul style="list-style-type: none"> <li>▶ SwSystemconst</li> </ul>

Table 155 Address Methods

Id	Rule <i>Chapter 10.1.4 [CenElemChapter_4]</i>
<i>[CEL_003]</i>	<b>SwAddrMethod</b> The following AUTOSAR elements shall be defined centrally: <ul style="list-style-type: none"> <li>▶ SwAddrMethod</li> </ul>

Table 156 Record Layouts

Id	Rule <i>Chapter 10.1.5 [CenElemChapter_5]</i>
<i>[CEL_004]</i>	<b>SwRecordLayout</b> The following AUTOSAR elements are defined in the packages IFX and IFL, which are not part of the CEL packages but are still central elements: <ul style="list-style-type: none"><li>▶ SwRecordLayout</li></ul>

Table 157 Keywords

Id	Rule <i>Chapter 10.1.6 [CenElemChapter_6]</i>
<i>[CEL_005]</i>	<b>Keywords</b> The following AUTOSAR elements shall be defined centrally for all RB product lines: <ul style="list-style-type: none"><li>▶ KeywordSets</li></ul>

Table 158 Data types

Id	Rule <i>Chapter 10.1.7 [CenElemChapter_7]</i>
<i>[CEL_006]</i>	<b>ImplementationDataType</b> The following AUTOSAR elements may be defined centrally and locally: <ul style="list-style-type: none"><li>▶ ImplementationDataType</li></ul>
<i>[CEL_061]</i>	<b>ApplicationDataType</b> The following AUTOSAR elements may be defined centrally or locally: <ul style="list-style-type: none"><li>▶ ApplicationDataType</li></ul>

Table 159 SwBaseType

Id	Rule <i>Chapter 10.1.8 [CenElemChapter_8]</i>
<i>[CEL_062]</i>	<b>SwBaseType</b> The following AUTOSAR elements shall be defined centrally. <ul style="list-style-type: none"><li>▶ SwBaseType</li></ul>

Table 160 Data Constraints

Id	Rule <i>Chapter 10.1.9 [CenElemChapter_9]</i>
<i>[CEL_063]</i>	<b>DataConstraints</b> The following AUTOSAR elements shall be defined centrally, if the AUTOSAR element, which is referencing it, is defined centrally, too. <ul style="list-style-type: none"><li>▶ DataConstraints</li></ul>

Table 161 DataTypeMappingSet

Id	Rule <i>Chapter 10.1.10 [CenElemChapter_10]</i>
<i>[CEL_064]</i>	<b>DataTypeMappingSet</b> The following AUTOSAR elements are defined centrally and locally: <ul style="list-style-type: none"><li>▶ DataTypeMappingSet</li></ul> It may only be defined centrally if the referenced ApplicationDataType AND the referenced ImplementationDataType are defined centrally, too. In this case the DataTypeMappingSet has the same parent AR package as the ApplicationDataType (and the same delivery unit).



Table 162 ApplicationDataTypes and ImplementationDataTypes

Id	Rule <i>Chapter 10.2.1 [CenElemChapter_ApplImplDataTypes]</i>
<i>[CEL_008]</i>	<b>Constraints Referenced by ApplicationDataTypes and ImplementationDataTypes</b> For all centrally defined ApplicationDataTypes and ImplementationDataTypes the referenced data constraint shall be defined centrally too.
<i>[CEL_050]</i>	<b>Reuse of ImplementationDataTypes</b> The reuse of ImplementationDataTypes is encouraged. This implies that there should be no 1:1 mapping between application and implementation data types.
<i>[CEL_054]</i>	<b>ImplementationDataType References</b> ImplementationDataTypes shall not be referenced directly, but via a DataTypeMappingSet.

Table 163 AUTOSAR Platform Types

Id	Rule <i>Chapter 10.2.2 [CenElemChapter_ARPlatformTypes]</i>
<i>[CEL_007]</i>	<b>AUTOSAR PlatformTypes</b> PlatformTypes are defined in the following file artefacts: <ul style="list-style-type: none"> <li>▶ Platform_Types.arxml (Hardware independent)</li> <li>▶ PlatformBase_Types.arxml (Hardware dependent)</li> </ul> The ASW may only reference the PlatformTypes, but never the PlatformBaseTypes.
<i>[CEL_025]</i>	<b>Reference Base AUTOSAR Platform Types</b> Reference base concept should be used instead of full qualified package path. With the reference base concept it is possible to define references which substitutes the path of the AUTOSAR Platform Types. With a reference base "Platform_ImplementationDataTypes" (which is conform to the rules for reference bases in AR package guideline, Rule ARPac_71 to Rule ARPac_75) the definition can be written in following form: <pre>&lt;IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE" BASE= "Platform_ImplementationDataTypes"&gt;uint32&lt;/IMPLEMENTATION-DATA-TYPE-REF&gt;</pre>

Table 164 AUTOSAR Standard Types

Id	Rule <i>Chapter 10.2.3 [CenElemChapter_ARStandardTypes]</i>
<i>[CEL_026]</i>	<b>Reference Base AUTOSAR Standard Types</b> Reference base concept should be used instead of full qualified package path. With the reference base concept it is possible to define references which substitutes the path of the AUTOSAR Standard Types. With a reference base "Std_ImplementationDataTypes" (which is conform to the rules for reference bases in AR package guideline) the definition can be written in following form: <pre>&lt;IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE" BASE= "Std_ImplementationDataTypes"&gt;E_OK&lt;/IMPLEMENTATION-DATA-TYPE-REF&gt;</pre>
<i>[CEL_091]</i>	<b>AUTOSAR Standard Types</b> AUTOSAR Standard Types are elements of ARPackage /AUTOSAR_Std/ImplementationDataTypes. The Category of this ARPackage is "STANDARD".

Table 165 Other Types

Id	Rule <i>Chapter 10.2.5 [CenElemChapter_OtherTypes]</i>
<i>[CEL_009]</i>	<b>RB Standard Types</b> For RB additional standard types besides the AUTOSAR standard types can be defined. They are called RB Standard Types.



Table 166 AR Package Structure for Central Elements

Id	Rule <i>Chapter 10.3 [CenElemChapter_PackStructure]</i>
<i>[CEL_011]</i>	<b>AR Package Structure for Central Elements</b> Path declaration <ul style="list-style-type: none"> <li>▶ for UBK ASW: /RB/Common/CentralElements/{kind}s</li> <li>▶ for BSW/CUBAS + ASW: /RB/RBA/Common/CentralElements/{kind}s</li> <li>▶ for Powertrain ASW: /RB/PT/Common/CentralElements/{kind}s</li> <li>▶ for Chassis ASW: /RB/Chassis/Common/CentralElements/{kind}s</li> <li>▶ for BSW but not CUBAS: <b>under discussion</b>.</li> </ul>
<i>[CEL_012]</i>	<b>Package Path for AUTOSAR Standard Types</b> Path for standard types according to AUTOSAR: /AUTOSAR_Std/ImplementationDataTypes/{NameOfStandardType} The category of these elements is STANDARD.
<i>[CEL_013]</i>	<b>Package Path for AUTOSAR SwBaseTypes</b> Path for SwBaseTypes according to AUTOSAR: /AUTOSAR_Platform/SwBaseTypes/{NameOfSwBaseType} The category of these elements is STANDARD.
<i>[CEL_016]</i>	<b>Package Path for RB Standard Types</b> Package path for RB standard types: /RB/Common/CentralElements/ImplementationDataTypes or /RB/{domain}/Common/CentralElements/ImplementationDataTypes. The Category of the leaf ARPackage "ImplementationDataTypes" is "STANDARD".
<i>[CEL_020]</i>	<b>Referencing to Central Elements</b> An ASW product line shall not use central elements of another product line. An ASW product line may use UBK common central elements, CUBAS central elements, their own central elements and the AUTOSAR central elements. E. g. ASW PT may use /RB/Common/CentralElements, /RB/RBA/Common/CentralElements and /-AUTOSAR_*/ and /RB/PT/Common/CentralElements. E. g. ASW PT shall not use /RB/Chassis/Common/CentralElements. BSW may access to /AUTOSAR/, /AUTOSAR_* and /RB/RBA, but not to /RB/Common/CentralElements. Reason: Self-Containment of CUBAS, because CUBAS should be delivered as separate product.
<i>[CEL_053]</i>	<b>Package Path for AUTOSAR ComStackTypes</b> Path for ComStackTypes according to AUTOSAR: /AUTOSAR_Comtype/ImplementationDataTypes/{NameOfComtype} The category of these elements is STANDARD.
<i>[CEL_112]</i>	<b>Package Path for AUTOSAR Platform Types</b> Path for platform types according to AUTOSAR: /AUTOSAR_Platform/ImplementationDataTypes/{NameOfPlatformType} The category of these elements is STANDARD.
<i>[CEL_113]</i>	<b>Package Path for UBK CompuMethods</b> Path for centrally defined CompuMethods: /RB/RBA/Common/CentralElements/CompuMethods/{NameOfCompuMethod}
<i>[CEL_114]</i>	<b>Package Path for SwAddrMethods</b> Path for centrally defined SwAddrMethods: /RB/RBA/Common/CentralElements/SwAddrMethods/{NameOfSwAddrMethod}

<i>[CEL_115]</i>	<b>Package Path for AUTOSAR KeywordSets</b> ARPackagePath for AUTOSAR KeywordSets: /AUTOSAR_AISpecification/KeywordSets/{NameOfKeywordSet} derived from /AUTOSAR/AISpecification/KeywordSets_Blueprint/KeywordList ARPackagePath for UBK KeywordSets: /RB/Common/NamingConventions/KeywordSets/{NameOfKeywordSet}
------------------	---

Table 167 Delivery Aspects

Id	Rule <i>Chapter 10.4 [CenElemChapter_DeliveryAspects]</i>
<i>[CEL_018]</i>	<b>Delivery Units</b> <b>A general assumption for the delivery unit names is done to refer always on common names within this guideline. Further this names can be mapped to SCM specific implementations.</b> The delivery unit for CUBAS central elements is CUCEL. CUCEL provides central elements for BSW but also elements which are relevant for BSW and ASW (e.g. PlatformTypes and StandardTypes). The delivery unit for ASW central elements of AUTOSAR WP 10.x and defined for UBK are called UBKCEL. The delivery unit for business unit specific central elements are the business unit specific CELs.
<i>[CEL_019]</i>	<b>Additional Central Element in Business Unit</b> Business units may define additional delivery units for central elements.
<i>[CEL_027]</i>	<b>Downward Compatibility of CUCEL and UBKCEL</b> CUCEL and UBKCEL are always downward compatible.

Table 168 Structural Conventions

Id	Rule <i>Chapter 11.2.1</i>
<i>[CLF_Structure_001]</i>	<b>CLF for Packages, components and configuration</b> Each CUBAS package (PAC), structural component (STCOMP), component (COMP), as well as the corresponding configuration container (ECU_CFG), shall have its own CLF file. CLF files of components need to be included in the parent CLF package, e.g. structural component or package.
<i>[CLF_Structure_002]</i>	<b>Project</b> There shall be only one top-level CLF file that defines the project specific includes and variants.
<i>[CLF_Structure_003]</i>	<b>Package Classes</b> The classes to be used for CLF packages are the same as eASee.BASD container classes in CDG context: PAC, PAC_D, STCOMP, COMP, ECU_CFG, TEST
<i>[CLF_Structure_004]</i>	<b>File Classes</b> The classes to be used for files are defined in a standard rules set from CDG-SMT/EMT (Rules.clf). If the project needs additional rules which are not covered by the standard rules, a project specific rules file can be used. This rule set shall be included in the top level CLF file (e.g. MyProject.clf). Rule files shall reside in the top level folder of the project, aside MyProject.clf
<i>[CLF_Structure_005]</i>	<b>Exclusivity</b> In CUBAS packages, structural components, components and configuration there shall be only one CLF file in one folder.
<i>[CLF_Structure_006]</i>	<b>Variants</b> To switch between different configurations at the same project, e.g. for different micro controllers or test boards or ECU, the CLF variant handling shall be used.

Table 169 Naming Conventions

Id	Rule <i>Chapter 11.2.2</i>
<i>[CLF_Naming_001]</i>	<b>Domain</b> The domain shall be used only once in the top level CLF file, and it shall have the value CUBAS.

Id	Rule <i>Chapter 11.2.2</i>
<i>[CLF_Naming_002]</i>	<b>Name uniqueness</b> The CLF file name and the containing package name shall be identical. CLF files and package names shall be unique across the whole project
<i>[CLF_Naming_003]</i>	<b>Packages and components</b> The name of the CLF file shall have the name of the respective CUBAS package or component. The same name shall be used as package name inside the CLF file.
<i>[CLF_Naming_004]</i>	<b>Structural components</b> CLF files for structural components shall have the name of the respective structural component extended by _Stc.
<i>[CLF_Naming_005]</i>	<b>Packages and components configuration</b> CLF files for configuration shall have the name of their respective package or component, extended by _Cfg.
<i>[CLF_Naming_006]</i>	<b>Structural components configuration</b> CLF files for configuration of structural components shall have the name of the respective structural component, extended by _Stc_Cfg.
<i>[CLF_Naming_007]</i>	<b>Variant Names</b> Variant names shall use capital letters and underscore as special character as delimiter for name parts. They have to be unique and self-explanatory. Variant names for microcontrollers have to be defined like this: <MANUFACTURER>_<DEVICE>_<HWPACKAGE>

## B Rules Derivation

The following [Table 170](#) will show all derivations of the rules regarding AUTOSAR specifications and MISRA Coding Standard. Rules with no reference to AUTOSAR or MISRA are CDG specific add ons.

Column "Rule" specifies the referred rule from this guidelines.

In the column "Reference AUTOSAR" all requirements from AUTOSAR are listed.

- ▶ BSWxyz are requirements from *"General Requirements on Basic Software Modules"*
- ▶ PLATFORMxyz are requirements from *"Platform Types"*
- ▶ STDxyz are requirements from *"Standard Types"*
- ▶ COMTYPExyz are requirements from *"Communication Stack Types"*
- ▶ MEMMAPxyz are requirements from *"Specification of Memory Mapping"*
- ▶ COMPILERxyz are requirements from *"Specification of Compiler Abstraction"*
- ▶ PROGxyz are requirements from *"Specification of C Implementation Rules"*
- ▶ TPS\_BSWMD\_xyz and BSWMDT are requirements from *"Specification of BSW Module Description Template"*
- ▶ TPS\_SWCD\_xyz and SWCDT are requirements from *"Software Component Template"*

In the last column "Reference MISRA" all references to MISRA rules are listed. Additional markers are used within this column. A '+' indicates that a MISRA rule is used with add ons. A '\*' indicates that a MISRA rule is modified. If more then one MISRA rule is listed BSW coding rule represents a summary of listed MISRA rules.

Table 170 Overview of Rules Derivation (AUTOSAR and MISRA)

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule CCode_001	–	1.1 +, 3.3
Rule CCode_002	–	3.2, 4.1, 4.2
Rule CCode_003	–	7.1
Rule CCode_004	–	6.4 +, 6.5
Rule CCode_005	–	3.5 +
Rule CCode_006	–	–
Rule CCode_MisraHIS_001	BSW007	all
Rule CCode_MisraHIS_002	BSW007	–
Rule CCode_MisraHIS_003	–	–
Rule CCode_MisraHIS_004	–	–
Rule CCode_Inits_001	–	9.1 *
Rule CCode_Inits_002	–	9.3
Rule CCode_Expr_001	–	12.1
Rule CCode_Expr_002	–	12.2
Rule CCode_Expr_003	–	12.3
Rule CCode_Expr_004	–	12.4 +
Rule CCode_Expr_005	–	12.5 +
Rule CCode_Expr_006	PLATFORM034 (partial)	12.6
Rule CCode_Expr_007	–	12.7 *
Rule CCode_Expr_008	–	12.8

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule CCode_Expr_009	–	12.9 *
Rule CCode_Expr_010	–	12.10 +
Rule CCode_Expr_011	–	12.11
Rule CCode_Expr_012	–	12.13
Rule CCode_Expr_013	–	10.4
Rule CCode_Expr_014	–	10.0a
Rule CCode_Expr_015	–	10.0b
Rule CCode_Control_001	–	13.1
Rule CCode_Control_002	–	13.2
Rule CCode_Control_003	–	13.3 +
Rule CCode_Control_004	–	13.4, 13.5
Rule CCode_Control_005	–	13.6
Rule CCode_CntrFlow_001	–	14.1
Rule CCode_CntrFlow_002	BSW00328	–
Rule CCode_CntrFlow_003	–	14.2
Rule CCode_CntrFlow_004	–	14.3
Rule CCode_CntrFlow_005	–	14.4 +, 14.5 +
Rule CCode_CntrFlow_006	–	14.6
Rule CCode_CntrFlow_007	–	14.10
Rule CCode_Switch_001	–	(15.0)
Rule CCode_Switch_002	–	15.1
Rule CCode_Switch_003	–	15.2 +
Rule CCode_Switch_004	–	15.3
Rule CCode_Switch_005	–	15.4
Rule CCode_Switch_006	–	15.5
Rule CCode_Struct_001	–	(18.4), 18.2
Rule CCode_Struct_002	–	–
Rule CCode_Struct_003	–	18.1
Rule CCode_Struct_004	–	–
Rule CCode_Prepro_001	–	19.1
Rule CCode_Prepro_002	–	19.2
Rule CCode_Prepro_003	–	19.3
Rule CCode_Prepro_004	–	19.6
Rule CCode_Prepro_005	–	19.14
Rule CCode_Prepro_006	–	19.16
Rule CCode_Prepro_007	–	19.17
Rule CCode_Prepro_008	–	19.11
Rule CCode_Macro_001	–	19.4
Rule CCode_Macro_002	–	19.5
Rule CCode_Macro_003	–	19.10
Rule CCode_Macro_004	–	19.8
Rule CCode_Macro_005	–	19.9
Rule Abstr_Main_001	BSW006	1.2 +

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule Abstr_Main_002	–	20.5, 20.6, 20.7, 20.8, 20.9, 20.10, 20.11, 20.-12
Rule Abstr_Main_003	–	–
Rule Abstr_Main_004	–	20.1, 20.2
Rule Abstr_NearFar_001	BSW00306, BSW00361, COMPILER035, COMPILER036, COMPILER040, COMPILER041, COMPILER046, COMPILER052, COMPILER055, COMPILER059	–
Rule Abstr_MemMap_001	MEMMAP021	–
Rule Abstr_MemMap_002	MEMMAP023	–
Rule Abstr_MemMap_003	–	–
Rule Abstr_MemMap_004	–	–
Rule Abstr_Inline_001	COMPILER060	(8.1)
Rule Abstr_Inline_002	–	–
Rule Abstr_Asm_001	–	2.1 +
Rule Abstr_DynMem_001	–	20.4
Rule Abstr_DynMem_002	–	18.3
Rule Abstr_PtrConv_001	–	11.1
Rule Abstr_PtrConv_002	–	11.2
Rule Abstr_PtrConv_003	–	11.3
Rule Abstr_PtrConv_004	–	11.4 +
Rule Abstr_PtrConv_005	–	11.5
Rule Abstr_PtrArith_001	–	17.1
Rule Abstr_PtrArith_002	–	17.2
Rule Abstr_PtrArith_003	–	17.3
Rule Abstr_PtrArith_004	–	17.6 *
Rule Abstr_PtrArith_005	–	17.5
Rule CCode_Types_001	BSW00304, PLATFORM013, PLATFORM014, PLATFORM015, PLATFORM016, PLATFORM017, PLATFORM018, PLATFORM041, PLATFORM042, PLATFORM061	6.2 *
Rule CCode_Types_002	BSW00304	–
Rule CCode_Types_003	–	6.1
Rule CCode_Types_004	–	–
Rule CCode_Types_005	BSW00355	–
Rule CCode_Types_006	BSW00304, PLATFORM005, PLATFORM020, PLATFORM021, PLATFORM022, PLATFORM023, PLATFORM024, PLATFORM025	–
Rule CCode_Types_007	PLATFORM033	–
Rule CCode_Symbols_001	PLATFORM054, PLATFORM055, PLATFORM056, PLATFORM026 (partial), PLATFORM034 (partial)	–
Rule CCode_Symbols_002	PLATFORM038, PLATFORM039	–
Rule CCode_Symbols_003	STD006, STD007, STD013, STD010	–
Rule CCode_Symbols_004	STD015	–
Rule CCode_Symbols_005	COMPILER051	–
Rule CCode_ComStackTypes_001	COMTYPE005	–

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule CCode_ComStackTypes_002	COMTYPE007	–
Rule CCode_ComStackTypes_003	COMTYPE008, COMTYPE010, COMTYPE017	–
Rule CCode_ComStackTypes_004	COMTYPE011	–
Rule CCode_ComStackTypes_005	COMTYPE012	–
Rule CCode_ComStackTypes_006	COMTYPE018	–
Rule CCode_ComStackTypes_007	COMTYPE019	–
Rule CCode_ComStackTypes_008	COMTYPE020	–
Rule CCode_ComStackTypes_009	COMTYPE022	–
Rule CCode_ComStackTypes_010	COMTYPE027	–
Rule CCode_ComStackTypes_011	COMTYPE028	–
Rule CCode_ComStackTypes_012	COMTYPE026	–
Rule CCode_ComStackTypes_013	COMTYPE031	–
Rule CDGNaming_001	BSW00300 (together with Rule CDGNaming_011), BSW00347	–
Rule CDGNaming_002	BSW00441 (partial)	–
Rule CDGNaming_012	ECUConfiguration: ecuc_sws_2108	–
Rule CDGNaming_003	BSW00307, BSW00305 (partial), BSW00310	–
Rule CDGNaming_004	–	5.3
Rule CDGNaming_005	–	5.4
Rule CDGNaming_006	–	–
Rule CDGNaming_007	–	5.5
Rule CDGNaming_008	–	5.6
Rule CDGNaming_009	–	5.2
Rule CDGNaming_010	–	5.1 *
Rule CDGNaming_011	BSW00300 (together with Rule CDGNaming_001)	–
Rule BSW_Files_001	BSW00346, BSW00334	–
Rule BSW_Files_002	–	–
Rule BSW_Files_003	BSW00370	–
Rule BSW_Files_004	BSW00345, BSW00346, BSW158	–
Rule BSW_Files_005	–	–
Rule BSW_Files_006	BSW00415	–
Rule BSW_Files_007	–	–
Rule BSW_Files_008	–	–
Rule BSW_HeaderInc_001	BSW00346	–
Rule BSW_HeaderInc_002	BSW00346	–
Rule BSW_HeaderInc_003	BSW00346	–
Rule BSW_HeaderInc_004	BSW00346, BSW00348, BSW00353	–
Rule BSW_HeaderInc_005	BSW00346	–
Rule BSW_HeaderInc_006	–	–
Rule BSW_HeaderInc_007	BSW00302	–
Rule BSW_HeaderInc_008	BSW00301	–
Rule BSW_HeaderInc_009	PROG_044	19.15 *
Rule BSW_HeaderInc_010	BSW004	–
Rule BSW_ServiceRTE_001	–	–

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule BSW_ServiceRTE_002	–	–
Rule BSW_APIDesign_003	BSW00359, BSW00360	–
Rule BSW_APIDesign_004	BSW00371	–
Rule BSW_APIDesign_005	BSW00331	–
Rule BSW_APIDesign_006	BSW00413	–
Rule BSW_APIDesign_007	BSW00343	–
Rule BSW_APIDesign_008	BSW00357, BSW00449, STD005, STD0011, BSW00377	–
Rule BSW_APIDesign_009	–	16.10
Rule BSW_APIDesign_010	–	16.1
Rule BSW_APIDesign_011	–	16.2 +
Rule BSW_APIDesign_012	–	16.6
Rule BSW_APIDesign_013	–	16.8
Rule BSW_APIDesign_014	–	16.9
Rule BSW_APIDesign_015	–	–
Rule BSW_APIDesign_016	–	–
Rule BSW_APIDesign_017	–	–
Rule BSW_APIDesign_018	–	–
Rule BSW_ProcISR_001	BSW00358	–
Rule BSW_ProcISR_002	–	–
Rule BSW_ProcISR_003	BSW00376	–
Rule BSW_ProcISR_004	–	–
Rule BSW_ProcISR_005	–	–
Rule BSW_ProcISR_006	–	–
Rule BSW_FuncObj_001	BSW00308	8.5 +
Rule BSW_FuncObj_002	–	8.2 +
Rule BSW_FuncObj_003	–	16.5
Rule BSW_FuncObj_004	–	8.9 *
Rule BSW_FuncObj_005	–	8.6 +
Rule BSW_FuncObj_006	–	8.1 +, 8.8 +
Rule BSW_FuncObj_007	–	8.3, 16.3, 16.4
Rule BSW_FuncObj_008	–	8.10, 8.11
Rule BSW_FuncObj_009	–	16.7
Rule BSW_FuncObj_010	–	8.4
Rule BSW_FuncObj_011	BSW00309	–
Rule BSW_Reentrancy_002	BSW00312	–
Rule BSW_Reentrancy_004	–	–
Rule BSW_VersionInfo_001	BSW00402	–
Rule BSW_VersionInfo_002	BSW00374	–
Rule BSW_VersionInfo_003	BSW00379	–
Rule BSW_VersionInfo_004	BSW003, BSW00318, BSW00321	–
Rule BSW_VersionInfo_005	BSW00407	–
Rule BSW_VersionInfo_006	–	–
Rule BSW_Sched_001	BSW00435	–



Rule	Reference AUTOSAR	Reference MISRA 2004
Rule CCode_Style_001	–	–
Rule CCode_Style_002	–	–
Rule CCode_Style_003	–	–
Rule CCode_Style_004	–	–
Rule CCode_Style_005	–	–
Rule CCode_Style_006	–	–
Rule CCode_Style_007	–	9.2
Rule CCode_Style_008	–	–
Rule CCode_Style_009	–	–
Rule CCode_Style_010	–	–
Rule CCode_Style_011	–	–
Rule CCode_Style_012	–	–
Rule CCode_Style_013	–	–
Rule CCode_BlockStyle_001	–	14.8 +
Rule CCode_BlockStyle_002	–	–
Rule CCode_BlockStyle_003	–	–
Rule CCode_BlockStyle_004	–	14.8 *
Rule CCode_Comments_001	–	–
Rule CCode_Comments_002	–	–
Rule CCode_Comments_003	–	2.3 *
Rule CCode_Comments_004	–	2.4
Rule CCode_Comments_005	–	–
Rule CCode_Comments_006	–	–
Rule CCode_Comments_007	–	–
Rule CCode_Comments_008	–	–
Rule BSWMD_Common_002	–	–
Rule BSWMD_Common_001	–	–
Rule BSWMD_Common_003	–	–
Rule BSWMD_Common_004	–	–
Rule BSWMD_Common_005	–	–
Rule BSWMD_ModuleDesc_001	–	–
Rule BSWMD_ModuleDesc_002	BSWMDT: constr_4019	–
Rule BSWMD_ModuleDesc_003	–	–
Rule BSWMD_ModuleDesc_004	BSWMDT: constr_4020	–
Rule BSWMD_ModuleDesc_005	–	–
Rule BSWMD_IntBehav_001	–	–
Rule BSWMD_IntBehav_002	BSWMDT: Table 6.1	–
Rule BSWMD_Impl_001	–	–
Rule BSWMD_Impl_002	–	–
Rule BSWMD_Impl_003	TPS_BSWMDT_4031, BSW00347	–
Rule BSWMD_Impl_004	–	–
Rule BSWMD_Impl_005	–	–
Rule BSWMD_Impl_006	BSW00318, BSW00321	–

Rule	Reference AUTOSAR	Reference MISRA 2004
Rule BSWMD_Impl_007	TPS_BSWMDT_4030, BSW00318	–
Rule BSWMD_Impl_008	–	–
Rule BSWMD_Split_001	–	–
Rule BSWMD_Split_002	BSWMDT: Table 4.2	–
Rule BSWMD_Split_003	–	–
Rule BSWMD_Split_004	–	–
Rule BSWMD_Split_005	–	–
Rule BSWMD_Split_006	–	–
Rule BSWMD_MCSupport_001	BSWMDT: Table 6.1	–
Rule BSWMD_MCSupport_002	–	–
Rule BSWMD_MCSupport_004	–	–
Rule BSWMD_MCSupport_003	BSWMDT: Table 6.2	–
Rule BSWMD_MCSupport_005	–	–
Rule BSWMD_MCSupport_006	–	–
Rule BSWMD_MCSupport_007	–	–
Rule BSWMD_Entry_001	–	–
Rule BSWMD_Entry_002	BSWMDT: Table 5.1, TPS_BSWMDT_4008	–
Rule BSWMD_Entry_003	BSWMDT: Table 5.1	–
Rule BSWMD_Entry_004	BSWMDT: Table 5.1	–
Rule BSWMD_Entry_005	BSWMDT: Table 5.1	–
Rule BSWMD_Entry_006	BSWMDT: Table 5.1, Table 5.3: BswCallType	–
Rule BSWMD_Entry_007	BSWMDT: Table 5.1, Table 5.2, constr_4014	–
Rule BSWMD_Entry_008	BSWMDT: Table 5.1, Table 5.4	–
Rule BSWMD_EntryRetArg_001	–	–
Rule BSWMD_EntryRetArg_002	TPS_BSWMDT_4008	–
Rule BSWMD_EntryRetArg_003	SWCDT: constr_1006	–
Rule BSWMD_EntryRetArg_004	TPS_BSWMDT_4012, BSWMDT: Table 5.7, constr_4052, constr_4053	–
Rule BSWMD_EntryRetArg_005	TPS_BSWMDT_4009, BSWMDT: Table 5.5, TPS_BSWMDT_4010	–
Rule BSWMD_EntryRetArg_006	TPS_BSWMDT_4009, BSWMDT: Table 5.5, Table 5.6, TPS_BSWMDT_4011	–
Rule BSWMD_EntryRetArg_007	TPS_BSWMDT_4009, BSWMDT: Table 5.5, Table 5.6, TPS_BSWMDT_4011, SWCDT: constr_1177	–
Rule BSWMD_EntryRetArg_008	TPS_BSWMDT_4009, BSWMDT: Table 5.5, Table 5.6, TPS_BSWMDT_4011, BSWMDT: constr_4021	–
Rule BSWMD_EntryRetArg_009	TPS_BSWMDT_4007, BSWMDT: Figure 5.1, constr_4021, TPS_SWCT_1275, SWCDT: Table 5.45	–
Rule BSWMD_EntryRetArg_010	BSWMDT: Figure 5.1, SWCDT: Table 5.42	–
Rule BSWMD_EntryRetArg_011	BSWMDT: Figure 5.6, SWCDT: Table 5.20	–
Rule BSWMD_EntryRetArg_012	–	–
Rule BSWMD_EntryRetArg_013	BSWMDT: constr_4056	–
Rule BSWMD_EntryRetArg_014	BSWMDT: constr_4057	–

In the introduction chapters following AUTOSAR requirements are used: BSW161, BSW162, BSW00342.

## C List of Basic Software Modules

In [Table 171](#), [Table 172](#) and [Table 173](#) overview lists of all BSW and Library modules are shown based on AUTOSAR 4.1 Rev 1. All lists are taken from: [\[Document / URL: \\SI8256\autosar\\$\SVN3-COPY\22\\_Releases\41\\_Release4.1R0001\02\\_Auxiliary\AUTOSAR\\_TR\\_BSWModuleList.pdf\]](#).

Table 171 Overview of Basic Software Modules

Module Long Name	Module Prefix (API Service Prefix)	Module ID (uint16)	AUTOSAR SW Layer
ADC Driver	Adc	123	I/O Drivers
COM	Com	050	Communication Services
BSW Mode Manager	BswM	042	System Services
BSW Scheduler Module	SchM	130	System Services
CAN Driver	Can	080	Communication Drivers
CAN Interface	CanIf	060	Communication HW Abstraction
CAN Network Management	CanNm	031	Communication Services
CAN State Manager	CanSM	140	Communication Services
CAN Transceiver Driver	CanTrcv	070	Communication HW Abstraction
CAN Transport Layer	CanTp	035	Communication Services
COM Manager	ComM	012	System Services
Complex Drivers	no prefix (AUTOSAR interface)	255	Complex Drivers
Core Test	CorTst	103	Microcontroller Drivers
Crypto Service Manager	Csm	110	System Services
Debugging	Dbg	057	Communication Services
Development Error Tracer	Det	015	System Services
Diagnostic Communication Manager	Dcm	053	Communication Services
Diagnostic Event Manager	Dem	054	System Services
Diagnostic Log and Trace	Dlt	055	System Services
Diagnostic over IP	DoIP	173	Communication Services
DIO Driver	Dio	120	I/O Drivers
ECU State Manager	EcuM	010	System Services
EEPROM Abstraction	Ea	040	Memory HW Abstraction
EEPROM Driver	Eep	090	Memory Drivers
Ethernet Driver	Eth	088	Communication Drivers
Ethernet Interface	EthIf	065	Communication HW Abstraction
Ethernet Transceiver Driver	EthTrcv	073	Communication HW Abstraction
UDP Network Management	UdpNm	033	Communication Services
Ethernet State Manager	EthSM	143	Communication Services
Flash Driver	Fls	092	Memory Drivers
Flash EEPROM Emulation	Fee	021	Memory HW Abstraction
Flash Test	FlsTst	104	Memory Drivers
FlexRay Driver	Fr	081	Communication Drivers
FlexRay Interface	FrIf	061	Communication HW Abstraction
FlexRay Network Management	FrNm	032	Communication Services
FlexRay State Manager	FrSM	142	Communication Services

Module Long Name	Module Prefix (API Service Prefix)	Module ID (uint16)	AUTOSAR SW Layer
FlexRay Tranceiver Driver	FrTrcv	071	Communication HW Abstraction
FlexRay AUTOSAR Transport Layer	FrArTp	038	Communication Services
FlexRay ISO Transport Layer	FrTp	036	Communication Services
Function Inhibition Manager	FIM	011	System Services
GPT Driver	Gpt	100	Microcontroller Drivers
ICU Driver	Icu	122	I/O Drivers
IO HW Abstraction	no prefix (AUTOSAR interface)	254	I/O HW Abstraction
IPDU Multiplexer	IpduM	052	Communication Services
LIN Driver	Lin	082	Communication Drivers
LIN Interface	LinIf	062	Communication HW Abstraction
LIN Network Management	LinNm	063	Communication Services
LIN State Manager	LinSM	141	Communication Services
LIN Transceiver Driver	LinTrcv	064	Communication HW Abstraction
MCU Driver	Mcu	101	Microcontroller Drivers
Memory Abstraction Interface	MemIf	022	Memory Services
Network Management Interface	Nm	029	Communication Services
NVRAM Manager	NvM	020	Memory Services
OCU Driver	Ocu	125	I/O Drivers
OS	Os (not used as API prefix)	001	System Services – OS
PDU Router	PduR	051	Communication Services
Port Driver	Port	124	I/O Drivers
PWM Driver	Pwm	121	I/O Drivers
RAM Test	RamTst	093	Memory Drivers
RTE	Rte	002	RTE
SAE J1939 Diagnostic Communication Manager	J1939Dcm	058	Communication Services
SAE J1939 Network Management	J1939Nm	034	Communication Services
SAE J1939 Request Manager	J1939Rm	059	Communication Services
SAE J1939 Transport Layer	J1939Tp	037	Communication Services
Service Discovery	Sd	171	Communication Services
Socket Adaptor	SoAd	056	Communication Services
SPI Handler Driver	Spi	083	Communication Drivers
Synchronized Time-Base Manager	StbM	160	System Services
TCP/IP Stack	Tcplp	170	Communication Services
Time Service	Tm	014	System Services
TTCAN Driver	Ttcan	084	Communication Drivers
TTCAN Interface	TtcanIf	066	Communication HW Abstraction
Watchdog Drive	Wdg	102	Microcontroller Drivers
Watchdog Interface	WdgIf	043	Onboard Device Abstraction
Watchdog Manager	WdgM	013	System Services

Module Long Name	Module Prefix (API Service Prefix)	Module ID (uint16)	AUTOSAR SW Layer
XCP	Xcp	212	Communication Services

Table 172 Overview of Library Modules

Libraries short name	Libraries abbreviation (API service prefix)	Module ID (uint16)
CRC Library	Crc	201
BFx Library	Bfx	205
Crypto Abstraction Library	Cal	206
E2E Library	E2E	207
EFx Library	Efx	208
IFl Library	lfl	209
MFl Library	Mfl	210
MFx Library	Mfx	211
IFx Library	lfx	213

Table 173 Overview of Special Files

Module short name	Short name (API service prefix)	Module ID
Platform Types	Platform	199
Compiler Abstraction	Compiler	198
Standard Types	Std	197
Communication Stack Types	Comtype	196
Memory Mapping	MemMap	195

## D Physical and Logical Types

In [Table 174](#) and [Table 175](#) overview lists of all physical and logical types of naming convention are shown. The types are used for the <pp> part of a name. Both lists are independent and not synchronized with the physical and logical types list of the UBK naming convention.

Table 174 Physical Types for Naming Convention <pp>

pp-Name	Display Unit	Short-Name of Display Unit	Long Name (English)	Long Name (German)
a	$m/s^2$	<i>MtrPerSecSqd</i>	<i>acceleration</i>	<i>Beschleunigung</i>
ag	°	Rad	angle	Winkel
am	$Nm*s$	<i>NwtMtrSec</i>	<i>angular momentum</i>	<i>Drehimpuls</i>
amt	<i>mol</i>	<i>Mol</i>	<i>amount of substance</i>	<i>Stoffmenge</i>
aosc	$mol/m^3$	<i>MolPerMtrCubd</i>	<i>amount of substance concentration</i>	<i>Stoffmengenkonzentration</i>
ar	$m^2$	<i>MtrSqd</i>	<i>area</i>	<i>Fläche</i>
cnd	$S/m$	<i>SPerMtr</i>	<i>conductivity</i>	<i>Leitfähigkeit</i>
egy	J	Jou	<i>heat (combustion heat)</i>	<i>Wärme</i>
fac	1	No unit	factor	Faktor
frq	Hz	Hz	frequency	Frequenz
htc	$W/(K*m^2)$	<i>WattPerKelvinMtrSqd</i>	<i>heat transfer coefficient</i>	<i>Wärmeübergangskoeffizient</i>
i	A	Ampr	electric current	Elektrischer Strom
jerk	$m/s^3$	<i>MtrPerSecCubd</i>	<i>jerk</i>	<i>Ruck</i>
len	$m$ (mile)*	<i>Mtr</i>	<i>length, distance</i>	<i>Länge, Strecke</i>
m	g	Gr	mass	Masse
moi	$kg*m^2$	<i>KiloGrMtrSqd</i>	<i>moment of inertia</i>	<i>Trägheitsmoment</i>
n	rpm	Rpm	rotational speed	Drehzahl
p	Pa (bar, mm-Hg)*	Pa	pressure	Druck
psi	–	–	<i>flux</i>	<i>Fluss</i>
pwr	W	Watt	power	Leistung
q	$A*s$ (C)*	<i>AmprSec, Coulmb</i>	<i>electric charge</i>	<i>elektrische Ladung</i>
r	Ohm	Ohm	resistance	Widerstand
rad	rad	NoUnit	measurement of plain angle	Radiant
rat	%, 1	Perc	ratio, duty cycle	Verhältnis, Tastverhältnis
rd	$m$	<i>Mtr</i>	<i>radius</i>	<i>Radius</i>
re	–	<i>NoUnit</i>	<i>reynolds number</i>	<i>Reynoldszahl</i>
rho	$kg/m^3$	<i>KiloGrPerMtrCubd</i>	<i>density</i>	<i>Dichte</i>
spl	dB	<i>DeciBel</i>	<i>sound pressure level</i>	<i>Schalldruckpegel</i>
t	degC	DegCgrd	temperature	Temperatur
ti	s (min, h)*	Sec (Mins, Hr)	time, duration	Zeitpunkt, zeitliche Dauer
tq	$Nm$	<i>NwtMtr</i>	<i>torque</i>	<i>Drehmoment</i>
u	V	Volt	voltage	Spannung
v	$m/s$	<i>MtrPerSec</i>	<i>velocity</i>	<i>Geschwindigkeit</i>
vcos	$m^2/s, N*s/m^2$	<i>MtrSqdPerSec, NwtSecPerMtrSqd</i>	<i>viscosity (kinematical), viscosity (dynamical)</i>	<i>Viskosität (kinematisch), Viskosität (dynamisch)</i>
vol	$m^3$ (L)*	<i>MtrCubd, Liter</i>	<i>volume</i>	<i>Volumen</i>

pp-Name	Display Unit	Short-Name of Display Unit	Long Name (English)	Long Name (German)
<i>w</i>	<i>Ws</i>	<i>WattSec</i>	<i>work, energy</i>	<i>Arbeit, Energie</i>

\*) optional unit

In *italic lines* physical types are indicated which are normally not used within BSW software. But these types are a part of UBK list of physical types.

Table 175 Logical Types for Naming Convention <pp>

pp-Name	Display Unit	Short-Name of Display Unit	Long Name (English)	Long Name (German)
adr	–	No Unit	address	Adresse
cntr	–	No Unit	counter	Zähler
data	–	No Unit	data	Daten
	–	No Unit	data	Daten
flg	–	No Unit	bit, binary message or variable	binäre Botschaft oder Variable ("Bedingung")
has	–	No Unit	boolean	Boolean
id	–	No Unit	Identifier	Bezeichner
idx	–	No Unit	index	Index
is	–	No Unit	boolean	Boolean
nr	–	No Unit	number, count	Nummer, Anzahl
opt	–	No Unit	command line option (only for Perl)	Kommandozeilenübergabe (nur bei Perl)
posn	–	No Unit	position	Position
reg	–	No Unit	copy of a register	Abbildung eines Registers
st	–	No Unit	status, state	Status, Zustand, Bitleiste
swt	–	No Unit	switch	Schalter
x	–	No Unit	others	Sonstige

## E HIS Metrics Overview

In [Table 176](#) all HIS metrics are listed.

Table 176 Overview HIS Metrics

Name of Metric	Description	Value	Compliance
<b>COMF</b> Comment Density	Relationship of the number of comments (outside of and within functions) to the number of statements.	> 0.2	yes
<b>PATH</b> Number of paths	Number of non cyclic remark paths (i.e. minimum number of necessary cases of test)	1 – 80	yes
<b>GOTO</b> Number of goto statements	Number of goto statements	0	yes
<b>v(G)</b> Cyclomatic Complexity	In accordance with the Cyclomatic Number	1 – 10	yes
<b>CALLING</b> Number of calling functions	By how many subfunctions is this function called?	0 – 5	no <sup>*1</sup>
<b>CALLS</b> Number of called functions	How many different functions does this function call? Calling the same subfunction counts only once.	0 – 7	yes
<b>PARAM</b> Number of function parameters	How complex is the function interface?	0 – 5	yes
<b>STMT</b> Number of instructions per function	How complex is the function?	1 – 50	yes
<b>LEVEL</b> Number of call levels	Depth of nesting of a function.	0 – 4	yes
<b>RETURN</b> Number of return points	Number of return points within a function	0 – 1	yes
<b>Si</b> The stability index	The stability index supplies a measure of the number of the changes (changes, deletions, additions) between two versions of a piece of software.	<= 1	no <sup>*2</sup>
<b>VOCF</b> Language scope	The language scope is an indicator of the cost of maintaining/changing functions. $\text{VOCF} = (N1 + N2) / (n1 + n2),$ where n1 = Number of different operators N1 = Sum of all operators n2 = Number of different operands N2 = Sum of all operands	1 – 4	no <sup>*3</sup>
<b>NOMV</b> Number of MISRA violations	Total number of the violations of the Rules from MISRA	0	yes
<b>NOMVPR</b> Number of MISRA violations per rule	Number of violations of each rule of the MISRA rule set	0	yes



Name of Metric	Description	Value	Compliance
<b>ap_cg_cycle</b> Number of recursi- ons	Call graph recursions	0	yes

\*<sup>1</sup>: This metric is deactivated because it is in conflict with the basic principle or goal of AUTOSAR to use services as often as possible. The number of function calls should not be restricted.

\*<sup>2</sup>: This metric cannot be measured and the effort to do that is too high. Necessary changes of a software should not be restricted. Each developer is responsible that changes are made as clear and as small as possible.

\*<sup>3</sup>: In normal case BSW software uses a lot of language scopes. A limitation for BSW could be to restrictively. Additionally a measurement by tool cannot be done untill now.

Reference: [Document "HIS Metrics Definition" \[HIS\]](#) .

## F References

The AUTOSAR clusters are build according to *[Document Release 4.0 Overview and Revision History / Name: R4Overview / Publisher: AUTOSAR]*.

The abbreviations chosen for the AUTOSAR documents are those defined in *[TR\_PDN]*.

### F.1 Robert Bosch GmbH AUTOSAR Guidelines

- ▶ *[Document Guideline Publishing and Roadmap / Name: Guidelines / Publisher: Robert Bosch GmbH internal link / URL: <http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view/Tools/GuideLines>]*
- ▶ *[Document List of the Guidelines and Artifacts / Name: GuidelineList / Publisher: Robert Bosch GmbH internal link / URL: <http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view/Tools/GuideLinesList>]*
- ▶ *[Document DGS AUTOSAR Coding and Data Description Guideline / Name: G\_ArCaDe / Publisher: Robert Bosch GmbH]*
- ▶ *[Document Generic Aspects Guideline / Name: G\_Generic / Publisher: Robert Bosch GmbH]*
- ▶ *[Document Integration Guideline / Name: G\_Integ / Publisher: Robert Bosch GmbH]*

### F.2 Robert Bosch GmbH AUTOSAR Artifacts

- ▶ *[Document Blueprint Handling / Name: A\_Blueprint / Publisher: CDG-SMT/EMT]*
- ▶ *[Document Central Elements / Name: A\_CEL / Publisher: CDG-SMT/EMT]*
- ▶ *[Document AUTOSAR Data Description / Name: A\_Data / Publisher: CDG-SMT/EMT]*
- ▶ *[Document Application Interfaces / Name: A\_AppII]*

### F.3 AUTOSAR Main Documents

"Main Documents' are general AUTOSAR documents facilitating a global view on requirements, concepts and terms *[R4Overview]*."

- ▶ *[Document Specification of Predefined Names in AUTOSAR / Name: TR\_PDN / Publisher: AUTOSAR]*

### F.4 AUTOSAR Basic Software Architecture and Runtime Documents

"Documents belonging to this Release cluster provide descriptions, requirements and specifications of the AUTOSAR Software Architecture and the Runtime Environment *[R4Overview]*."

- ▶ *[Document List of Basic Software Modules / Name: TR\_BSWModuleList / Publisher: AUTOSAR]*
- ▶ *[Document General Requirements on Basic Software Modules / Name: SRS\_BSWGeneral / Publisher: AUTOSAR]*
- ▶ *[Document Platform Types / Name: TR\_Platform / Publisher: AUTOSAR]*
- ▶ *[Document Standard Types / Name: TR\_Std / Publisher: AUTOSAR]*

- ▶ *[Document Communication Stack Types / Name: TR\_Comtype / Publisher: AUTOSAR]*
- ▶ *[Document Specification of Memory Mapping / Name: SWS\_MEM / Publisher: AUTOSAR]*
- ▶ *[Document Specification of Compiler Abstraction / Name: SWS\_Compiler / Publisher: AUTOSAR]*
- ▶ *[Document Specification of Fixed Point Interpolation Routines (IFx Library) / Name: SWS\_Ifx / Publisher: AUTOSAR]*
- ▶ *[Document Specification of Floating Point Interpolation Routines (IFI Library) / Name: SWS\_Ifl / Publisher: AUTOSAR]*

## F.5 AUTOSAR Methodology and Templates Documents

"Documents belonging to this Release cluster provide requirements, specifications, templates and guidelines on the AUTOSAR methodology and tool chain [\[R4Overview\]](#)."

- ▶ *[Document Software Component Template / Name: TPS\_SWCT / Publisher: AUTOSAR]*
- ▶ *[Document Specification of BSW Module Description Template / Name: TPS\_BSWMDT / Publisher: AUTOSAR]*
- ▶ *[Document Generic Structure Template / Name: TPS\_GST / Publisher: AUTOSAR]*
- ▶ *[Document Specification of C Implementation Rules / Name: TR\_CIMPL / Publisher: AUTOSAR]*

## F.6 Others

- ▶ *[Document AUTOSAR Tool Platform User Group / Name: ARTOP / Publisher: Artop / URL: <http://www.artop.org/>]*
- ▶ *[Document AI Specification / Name: AISpec / URL: [file:///Si8256/autosar\\$/SVN3-COPY/24\\_Sources/tags/R3.1.005/MOD\\_AIForXMLSchemaR3.0\\_310/AUTOSAR\\_ApplicationInterfaces\\_ForXMLSchema\\_R3.0.arxml](file:///Si8256/autosar$/SVN3-COPY/24_Sources/tags/R3.1.005/MOD_AIForXMLSchemaR3.0_310/AUTOSAR_ApplicationInterfaces_ForXMLSchema_R3.0.arxml)]*
- ▶ *[Document HIS Metrics Definition / Name: HIS / URL: [http://portal.automotive-his.de/images/pdf/SoftwareTest/his-sc--metriken.1.3.1\\_e.pdf](http://portal.automotive-his.de/images/pdf/SoftwareTest/his-sc--metriken.1.3.1_e.pdf)]*
- ▶ *[Document oAW Reference / Name: oAWRef / URL: <http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.xpand.doc/help/ch01.html>]*
- ▶ *[Document CLF Documentation / Name: CLFDocu / URL: <https://inside-wiki.bosch.com/confluence/display/CDGSMT/CLF+documentation>]*

## G Approval of Guideline

See *[Document AUTOSAR Methoden TWG / Publisher: Robert Bosch GmbH, not published / URL: <http://abt-ismtwiki.abt.de.-bosch.com/twiki/bin/view/Tools/ArMethods>]*.

Review was done according to the defined review process. Final approval of Artifact was given in Jira-Issue *[Document / Name: ARMETH-1163 / Publisher: Robert Bosch GmbH, not published]* .

Review documents are available at BOSCH CDG-SMT/EMT: the "call for review" mail(s), the release candidate, and the findings list.

# H Documentation

The BSW Coding Guidelines are published on following locations:

*[Document FOSWiki Intranet: / URL: <http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view/Tools/GuideLines>]*

*[Document Release Directory of CDG Migration Project: / URL: \\bosch.com\dfsrb\DfsDE\DIV\CDG\Prj\CDG-Migration\99\_-Release\WG04\_CodingRules]*

*[Document Subversion System (SVN, login needed): / URL: [http://cdgsvn.apps.intranet.bosch.com:8080/emt\\_repo/work/13-Methodology/004\\_Autosar-Guidelines/020\\_Guidelines/BSWCoding/](http://cdgsvn.apps.intranet.bosch.com:8080/emt_repo/work/13-Methodology/004_Autosar-Guidelines/020_Guidelines/BSWCoding/)]*

# I Technical Terms

## ACTIVITY

### C

Calibration 192

## AUTOSAR ITEMS

### —

«atpSplitable» 234, 234, 291, 291, 291

### A

ApplicationDataType 189, 365

AliasNameSet 304, 304, 381

Application-Array-Data-Type 185, 365

ApplicationArrayDataType 184, 184, 184, 185, 187, 365, 365

ApplicationDataType 180, 180, 180, 180, 180, 180, 181, 181, 182, 182, 182, 182, 183, 183, 183, 184, 187, 187, 190, 192, 201, 206, 206, 212, 227, 227, 364, 364, 364, 364, 364, 365, 368, 370

ApplicationPrimitiveDataType 183, 183, 364

ApplicationRecordDataType 185, 187, 365

ApplicationRecordElement 185, 185

ARElement 290, 290, 290, 290, 291, 291, 291, 291, 292, 295, 295, 295, 298, 298, 299, 303, 305, 378, 378, 378, 379, 380

ArPackage 219, 219, 219, 219, 219, 224, 227, 229, 232, 233, 234, 235, 235, 235, 275, 290, 290, 290, 290, 290, 290, 290, 291, 291, 291, 291, 292, 292, 292, 292, 293, 294, 294, 294, 294, 294, 294, 294, 295, 295, 295, 295, 295,

295, 298, 298, 298, 298, 298, 299, 299, 299, 299, 299, 299, 299, 299, 300, 300, 300, 300, 300, 300, 300, 301, 301, 301, 301, 301, 302, 302, 302, 302, 303, 303, 303, 303, 304, 304, 304, 304, 304, 304, 305, 305, 305, 305, 305, 305, 306, 306, 306, 369, 369, 370, 371, 375, 378, 378, 378, 378, 378, 378, 378, 379, 379, 379, 379, 379, 379, 379, 379, 379, 379, 380, 380, 380, 380, 380, 380, 381, 381, 381, 381, 381, 381, 381, 381

ARRAY 184, 184, 184, 184, 184, 184

ArraySize 190, 190, 366

ArraySizeSemantics 184, 185, 185, 190, 190, 365, 365, 366

### B

BswImplementation 222, 228, 228, 229, 229, 230, 230, 231, 231, 232, 235, 235, 235, 370, 370, 370, 371, 371, 371, 371, 371, 371, 371

BswInternalBehavior 222, 226, 226, 227, 227, 231, 232, 233, 234, 234, 235, 238, 239, 241, 370, 370, 371, 371, 371, 371, 371, 372, 372

BswModuleDescription 222, 223, 223, 224, 224, 224, 225,

232, 233, 235, 369, 369, 369, 370, 371, 371

BswModuleEntry 274, 274, 274, 274, 274, 274, 275, 275, 275, 277, 278, 279, 281, 281, 281, 282, 282, 282, 283, 284, 285, 285, 285, 285, 286, 287, 288, 288, 375, 375, 376, 376, 376, 376, 376, 377, 377, 377, 377, 377, 377, 377

### C

Category 180, 183, 184, 184, 184, 185, 299, 299, 300, 300, 300, 300, 364, 379, 379, 379, 379, 379, 379

CompuMethod 290

### D

DataProtoType 189, 189, 365, 365

DataTypeMap 190

DataTypeMappingSet 190

### E

Element 184, 185, 185, 185, 187, 187, 187, 365, 365, 365, 365

### F

FibexElement 295, 378

FIXED-SIZE 184

### I

ImplementationDataType 185, 187, 187, 187, 187, 187, 188, 189, 189, 189, 189, 189, 365, 365, 365, 365, 365, 365

05

ImplementationDataType 190, 201,  
235, 236, 236, 283, 283,  
283, 283, 283, 283, 283,  
284, 284, 284, 285, 285,  
287, 287, 287, 287, 376,  
376

ImplementationDataTypeElement  
190, 366

15

InstantiationDataDefProps 203,  
203, 203, 367

**M**

20

MaxNumberOfElements 184, 184,  
184, 185, 185, 185, 365,  
365, 365

MCAL 15, 16, 44, 44, 58, 338

25

McSupport 238, 241, 301, 301,  
305, 305, 305

**P**

30

PackageRef 306

**AUTOSAR RULES**

35

**C**

constr\_1015 201

**T**

40

**BOSCH RULES**

45

**C**

**Codes**

50

—

.arxml 290, 290

55

/.../EcucModuleDefs 302

/AUTOSAR/EcucDefs 302

/AUTOSAR\_LinSM 302, 380

60

/RB/RBA/RBA\_Bernd 302, 380

\_Blueprint 295, 378

\_Example 295, 378

**A**

65

ParameterAccess 201

ParameterDataPrototype 201, 241

PerInstanceParameters 241, 372

**R**

recommendedPackage 295

ReferenceBase 292, 292, 305, 305,  
306, 306, 306, 306, 306,  
306, 306, 306, 306, 306,  
306, 306, 381, 381,  
381, 381, 381, 381, 381

**S**

Short-Name 183, 364

ShortName 180, 290, 295, 300,  
302, 302, 306, 306, 378,  
379, 380, 380, 381

StaticMemorys 238, 239, 239, 372

STRUCTURE 184, 185, 185

SwCalibrationAccess 187, 365

TPS\_GST\_00017 294, 300

TPS\_GST\_00049 295

TPS\_GST\_00082 302

CEL\_011 295

CEL\_020 295

ApplicationDataTypes\_Blueprint  
295, 378

AUTOSAR\_Rte 301

**B**

BLUEPRINT 299, 299, 300, 300,  
379, 379, 379

**C**

CalibrationParameterValueSets  
303, 380

CentralElements 299, 379

CUCEL\_RecordLayouts 306

SwDataDefProps 183, 183, 187,  
187, 187, 187, 201, 201,  
201, 201, 201, 202, 202,  
202, 203, 278, 283, 283,  
283, 284, 284, 284, 284,  
284, 284, 285, 286, 286,  
286, 287, 287, 287, 287,  
364, 365, 365, 365, 367,  
367, 376, 376, 376, 377,  
377, 377, 377, 377, 377,  
377, 377

SwSystemconsts 300

**U**

Unit 290

**V**

VALUE 184

VARIABLE-SIZE 184

VariableDataPrototype 201, 239,  
239

TPS\_GST\_00083 294

TPS\_GST\_00085 295

TR\_PDN\_00001 300, 300, 379

CEL\_061 182

**D**

DesignPatterns 299, 379

DocumentationPatterns 299, 379

**E**

ECU 304, 381

EcucModuleConfigurationValuess  
302, 380

EcuFlatView 303, 380

ElementKind 306, 306, 381, 381

00  
  
  
05  
  
10  
  
15  
  
20  
  
25  
  
30  
  
35  
  
40  
  
45  
  
50  
  
55  
  
60  
  
65  
  
70



EXAMPLE 299, 299, 300, 300, 379, 379, 379

F

FlatMaps 303, 380

I

ICS 299, 299, 300, 300, 379, 379, 379

N

NamePatterns 299, 379

NamingConventions 299, 379

P

Control elements

I

Others

–

«atpSplitable» 232

Products

E

Tool

I

ISOLAR-A 291

R

RTE 15, 16, 56, 95, 95, 96, 99, 99, 99, 99, 99, 99, 100,

Variables

C

Calibration and Measurement data 179

XML/SGML-Attributes

D

PackagePatterns 299, 379

PCFG\_ 304, 381

PCT\_ 303, 380

Platform 301

PlatformTypes 301

PreconfiguredValues 302, 380

Project 303, 380

PTCEL\_Units 306

R

RecommendedValues 302, 380

InternalBehavior 192

C

ECU 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 16, 44, 71, 71, 77, 116, 179, 192, 348

100, 100, 100, 100, 100, 100, 100, 101, 101, 101, 101, 101, 101, 101, 101, 101, 101, 102, 103, 103, 103, 103, 103, 110, 116, 116, 230, 236, 236, 237, 238,

Calibration parameter 195, 366

curves 179

DEST 291

RootSwCompo 303, 380

S

Scope 306, 306, 381, 381

STANDARD 299, 299, 300, 300, 301, 379, 379, 379

Std 301

SwArchitecture 299, 379

SwComponentTypes 303, 380

SYS 304, 381

U

UBK 303, 380

calibration 240, 241, 372

H

hex file 179, 238, 238

238, 238, 238, 238, 241, 241, 243, 244, 244, 244, 289, 339, 345, 345, 346, 346, 346, 346, 346, 346, 372

RTE generator 230, 230, 230, 230

M

maps 179



## XML/SGML-Tags

### D

DESC 133, 352

### E

ECUC-DEFINITION-COLLECTION  
132, 352

ECUC-MODULE-DEF 132, 352

ECUC-VALUE-COLLECTION 130,  
351

### I

INTRODUCTION 133, 352

### L

LONG-NAME 133, 352

### M

MULTIPLE-CONFIGURATION-CON-  
TAINER 139, 357

### O

ORIGIN 133, 134, 352

### S

SHORT-NAME 132, 132, 352, 352

### U

UNIT-REF 291

# J Version Information

## 1 version overview

Version	Date	Publisher	State
1.6	2014-02-21	Volker Kairies	released
1.5	2013-10-17	Volker Kairies	released
1.4	2012-05-21	Volker Kairies	released
1.3	2011-12-09	Volker Kairies	released
1.2	2011-03-25	Volker Kairies	released
1.1	2010-12-17	Volker Kairies	released
1.0	2010-12-01	Volker Kairies	released
0.5		Volker Kairies	under development
0.4		Volker Kairies	under development
0.3		Volker Kairies	under development
0.2		Volker Kairies	under development
0.1		Volker Kairies	under development

## 2 modifications

Version	Change	Related to
1.6	ARMETH-782: BSWMD shall allways contain module's longname; New rule BSWMD_ModuleDesc_003	Content
	ARMETH-1027: BSWMD: Specify a rule how the shorname of a module has to be set and when tags for vendor id and vendor api infix has to be set; New Rule BSWMD_Impl_003	Content
	ARMETH-1030: BSWMD: Add CodeDescriptors as sub-element from BSW-IMPLEMENTATION; New Rule BSWMD_Impl_004	Content
	ARMETH-1031: BSWMD: Specifiy rules for handling of splitable; New chapter 8.2.2 and new rules Rule BSWMD_Split_001 to Rule BSWMD_Split_006chapter	Content
	ARMETH-1032: BSWMD: Give information for BSW-IMPLEMENTATION: Use Case, Tag structure, explanation, minimum set, etc.; New chapter 8.2.1.3 and new Rules Rule BSWMD_Impl_001 to Rule BSWMD_Impl_008, modified Rule BSW_VersionInfo_004	Content
	ARMETH-1033: BSWMD: Give information for BSW-MODULE-DESCRIPTION and BSW-INTERNAL-BEHAVIOR: Use Case, Tag structure, explanation, minimum set, etc.; New chapter 8.2.1.1 and 8.2.1.2 and new rules Rule BSWMD_ModuleDesc_001 to Rule BSWMD_ModuleDesc_004 and Rule BSWMD_IntBehav_001 to Rule BSWMD_IntBehav_002	Content
	ARMETH-1038: Rule CCode_Prepro_005: Show #ifdef and #ifndef in the example because both are not forbidden; Modified Rule CCode_Prepro_005	Content
	ARMETH-1062: Rule BSWMD_MCSupport_005: Add naming pattern for structure and value block; Modified Rule BSWMD_MCSupport_005	Content
	ARMETH-1077: BSWMD_MCSupport_001, Rule BSWMD_MCSupport_003: Specify connection to BswInternalBehavior; Modified rules Rule BSWMD_MCSupport_001 and Rule BSWMD_MCSupport_003	Content
	ARMETH-1078: Create a rule for prohibition of open source software; New Rule Abstr_OSS_001	Content
	ARMETH-1080: Restructuring of the initialization chapter for BSWMD; Restructured chapters 8.1 and 8.2, new rules Rule BSWMD_Common_002 to Rule BSWMD_Common_005, modified Rule BSWMD_Common_001, Udate of the file header definition	Content
	ARMETH-1081: Rule CCode_Style_002, Rule OAW_Xtend_001, Rule OAW_Xpand_001, Rule Perl_001: Update years date in the file header template. Explanation added that the year represents the creation respectively the first release of the corresponding file.	Content
	ARMETH-1084: Create rules to fill initial data with neutral values	Content
	ARMETH-1094: BSWMD: Specify use case for BSW Module API Description (BswModuleEntry); New chapter 8.3.4 and new rules Rule BSWMD_Entry_001 to Rule BSWMD_Entry_008 and Rule BSWMD_EntryRetArg_001 to Rule BSWMD_EntryRetArg_014	Content
	ARMETH-1095: BSW Measurement and Calibration Support: Harmonize terms, rules and explanations; Modified chapter 8.3.1 and modified rules Rule BSWMD_MCSupport_002, Rule BSWMD_MCSupport_004, Rule BSWMD_MCSupport_005 and Rule BSWMD_MCSupport_006	Content
	ARMETH-1099: Rule Perl_019: Remove the template from the guideline and provide it as template file; Removed Rule Perl_019	Content
	ARMETH-1100: Integrate DataDescription Artefact V1.5	Content
	More detailed list of changes --> see change description in SVN and CDG migration repository	Content

Version	Change	Related to
1.5	Update of Abstract chapter with "Consideration of BSW Coding Guidelines"	Content
	New chapter/rules for "BSW Service Module with and without RTE"	Content
	New chapter/rules for "Ensure Usability of BSW Modules in C++ Environments"	Content
	New BSWMD use case: "Scheduling of BSW via RTE"	Content
	New rule set: CLF (Classification File)	Content
	New chapters in Rule Set oAW for "ID Generator Templates" and "Auxiliary Rules"	Content
	New chapter/rules for "Design of Processes and Interrupt Service Routines"	Content
	Naming convention for measurement and calibration values added	Content
	Integration ECU Configuration V1.2	Content
	Integration AR Packages Guideline V1.4	Content
	Integration CEL Central Elements V1.2	Content
	Integration Pragma Concept (V1.2.5) (Abstraction of Addressing Keywords, module specific MemMap header)	Content
	Integration DocumentReferences V2.0	Content
	Over 65 JIRA requests are handled	Content
	Detailed list of changes --> see change description in SVN and CDG migration repository	Content
1.4	New directory structure of the guideline	Content
	Fixes and issues from JIRA processed	Content
	Update of rules for Multicore use case (chapter for reentrancy)	Content
	First release of rule set for Data Description (BSWMD ARXML, Measurement and Calibration Support)	Content
	First release of rule set for oAW Coding Rules	Content
	Integration ECU Configuration V1.1	Content
	Integration AR Packages Guideline V1.1	Content
	Integration CEL Central Elements V1.0	Content
	Integration Pragma Concept (V1.2.2) (Abstraction of Addressing Keywords, module specific MemMap header)	Content
	Detailed list of changes --> see change description in SVN and CDG migration repository	Content
1.3	List of changes --> see change description in SVN and CDG migration repository	Content
1.2	List of changes --> see change description in SVN and CDG migration repository	Content
1.1	Update of chapter "Scope"	Content
	Update of rule set "Naming convention"	Content
	Corrections and updates because of review findings	Content
	Second release for CDG-SMT/ESx and CDG migration project	Content
	Release in FosWiki: ( <a href="http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view-/Tools/GuideLines">http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view-/Tools/GuideLines</a> )	Content
1.0	First version of naming convention added	Content
	Corrections because of review findings	Content
	First release for CDG-SMT/ESA and CDG-SMT/ESB	Content

Version	Change	Related to
0.5	Update of all chapters: Adding MISRA rules, adding sub chapters, reorganisation of rule sets	Content
	Second draft version for CCB AUTOSAR Tooling / CUBAS Integration (PSA)	Content
	First version for review from CUBAS Fachteamleiter	Content
0.4	Adding MISRA rules in various rule sets, Rework of rule set "Module Design and Implementation"	Content
	Enhancement of rule set "Style Guide"	Content
	First draft version for CCB AUTOSAR Tooling / CUBAS Integration (PSA)	Content
0.3	Adding rule sets (Compiler abstraction / portability, Style Guide), Formal and structural changes	Content
0.2	First version for working group internal review and for presentation in CDG migration project review workshop	Content
0.1	Initial version	Content