

CDG-SMT | 1.10 | released for CDG-SMT | 2016-01-31

BSW Coding Guideline

www.bosch.com



BOSCH
Invented for life

25 **AUTOSAR**



30 **The Motor Industry Software Reliability Association**

35 **HIS AK Softwaretest**

ISO/IEC 9899:1990

40

45

50

55

60

2016-01-31 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.

65

70

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:

- 15 ▶ 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.
- 20 ▶ A **bugfix** is only a code correction in a small area. A bugfix contains no functional or non-functional extensions. Therfor there is no consideration of a new BSW Coding Guideline version for a bugfix.
- 25 ▶ **Released software components** should be adapted to a new BSW Coding Guideline version only if there is an explicit demand from a BBM (was UBK) product line.

Table of Contents

a Scope	12
a.1 Used Conventions.....	12
a.1.1 Used Keywords	12
a.1.2 Rule Structure.....	13
a.1.2.1 Rule Example.....	13
a.1.2.2 Rule Structure Description	13
a.1.3 Rule Set Structure	14
a.2 Used Guideline Artifacts	14
1 Introduction	16
2 Glossary	17
3 Rule Set: C Coding Guidelines	19
3.1 Rule Set: C Language Rules.....	19
3.1.1 General Language Topics.....	19
3.1.2 MISRA and HIS Metrics Conformity	23
3.1.3 Initializations	25
3.1.4 Expressions	29
3.1.5 Control Statement Expressions	37
3.1.6 Control Flow	41
3.1.7 Switch Statements	44
3.1.8 Structures and Unions	48
3.1.9 Preprocessing Directives	50
3.1.10 Macros.....	55
3.1.11 Essential Type Model	58
3.2 Rule Set: Compiler Abstraction / Portability	68
3.2.1 Main Rule	68
3.2.2 Prohibition of Language and Compiler Specifics	69
3.2.3 Abstraction of Addressing Keywords	71
3.2.4 Abstraction of Memory Mapping.....	75
3.2.5 Abstraction of Inline Functions	81
3.2.6 Handling of Assembler Instructions	82
3.2.7 Prohibition of Dynamic Memory	83

05	3.2.8 Pointer Type Conversions	84
10	3.2.9 Pointer Arithmetic and Arrays	88
15	3.2.10 Ensure Usability of BSW Modules in C++ Environments	92
20	3.2.11 Prohibition of Open Source Software	96
25	3.3 Rule Set: Types and Symbols	96
30	3.3.1 Base Integer and Float Data Types	96
35	3.3.2 Optimized Integer Data Types	98
40	3.3.3 Standard Symbols	99
45	3.3.4 Specific Types and Symbols for Communication Software	102
50	3.3.5 Module Specific Add Ons	108
55	3.4 Rule Set: Naming Convention.....	109
60	3.5 Rule Set: Module Design and Implementation	122
65	3.5.1 Basis Set of Module Files	122
70	3.5.2 Header Include Concept	127
75	3.5.3 BSW Service Module with and without RTE	133
80	3.5.4 Design of APIs (Application Programming Interfaces)	136
85	3.5.5 Design of Processes and Interrupt Service Routines	144
90	3.5.6 Definition and Declaration of Functions and Objects	145
95	3.5.7 Code Re-entrancy	150
100	3.5.8 Providing Version Information	152
105	3.5.9 BSW Scheduler and Exclusive Areas	155
110	3.6 Rule Set: Style Guide.....	156
115	3.6.1 Common File Style.....	156
120	3.6.2 Block Style	160
125	3.6.3 Comments	162
130	4 Rule Set: ECU Configuration.....	165
135	4.1 Introduction to ECUCconfig	165
140	4.1.1 Authoring of ECUC artifacts	166
145	4.1.2 How ECUC Values Are Processed	166
150	4.2 ECUC Values	167
155	4.2.1 ECUC Values: Content Responsibility	167
160	4.2.2 ECUC Values: Editing, Checking, and Reviewing	169

05	4.2.3 ECUC Values: Testing	169
10	4.2.4 ECUC Values: Content-related Procedures	170
15	4.3 ECUC Parameter Definitions.....	170
20	4.3.1 ECUC ParamDefs: Content Responsibility	170
25	4.3.2 ECUC ParamDefs: Editing, Checking, and Reviewing	170
30	4.3.3 ECUC ParamDefs: Testing	171
35	4.3.4 ECUC ParamDefs: Content-related Procedures	171
40	4.4 ECUC Processors	175
45	4.4.1 ECUC Processors: Content Responsibility and Language	175
50	4.4.2 ECUC Processors: Editing, Checking, and Reviewing	175
55	4.4.3 ECUC Processors: Testing	175
60	4.4.4 ECUC Processors: Documentation	176
65	4.4.5 ECUC Processors: General Content-related Procedures	176
70	4.4.6 ECUC Processors: oAW-specific Procedures	179
	4.4.7 ECUC Processors: Perl-specific Procedures	181
	4.5 Build Action Manifests	189
	4.5.1 BuildAction Declarations	189
	4.5.2 Input/Output Declarations	191
	4.5.2.1 Artefact Declarations	191
	4.5.2.2 Model Reference Declarations	193
	4.5.3 Action Specific Declarations	195
	5 Rule Set: Perl Coding Rules	198
	5.1 Code Layout and Comments	198
	5.2 Naming Conventions	200
	5.3 Coding Conventions.....	204
	5.4 Programming Tips.....	205
	5.5 Templates	209
	5.6 Usage of perltidy.....	210
	6 Rule Set: oAW Rules	211
	6.1 General oAW rules	211
	6.2 Model Workflow Environment	211

05	6.3 Configuration Validator files (*.chk)	212
10	6.4 Extensions (Xtend files, *.ext)	213
15	6.5 Generator templates (Xpand, *.xpt)	216
20	6.6 ID Generator templates	218
25	6.7 Auxilliary rules	219
30	7 Rule Set: Data Description	220
35	7.1 Data Types	220
40	7.1.1 Interaction between DataPrototype and DataTypes	220
45	7.1.2 ApplicationDataTypes	221
50	7.1.3 Primitive ApplicationDataTypes applicable for both BSW and ASW	222
55	7.1.3.1 ApplicationDataType_Value	222
60	7.1.3.1.1 Rules for ApplicationDataType of Category Value	225
65	7.1.3.2 ApplicationDataType_Boolean	227
70	7.1.3.2.1 Rules for ApplicationDataType of Category Boolean	230
75	7.1.3.3 ApplicationDataType_ValueBlock	230
80	7.1.3.3.1 Rules for ApplicationDataType of CategoryValueBlock	231
85	7.1.3.4 ApplicationDataType_String	231
90	7.1.3.4.1 Rules for ApplicationDataType of Category String	231
95	7.1.4 Complex Application DataType	231
100	7.1.4.1 ApplicationDataType of Category Array	231
105	7.1.4.1.1 Rules for ApplicationDataType of Category Array	234
110	7.1.4.2 ApplicationDataType of Category Structure	234
115	7.1.4.2.1 Rules for ApplicationDataType of Category Structure	237
120	7.1.5 ImplementationDataType	237
125	7.1.5.1 Rules for ImplementationDataType	238
130	7.1.5.2 ImplementationDataType of Category Value	239
135	7.1.5.2.1 Rules for ImplementationDataType of Category Value	239
140	7.1.5.3 ImplementationDataType of Category Array	240
145	7.1.5.3.1 Rules for ImplementationDataType of Category Array	241
150	7.1.5.4 ImplementationDataType of Category Structure	242
155	7.1.5.4.1 Rules for ImplementationDataType of Category Structure	243
160	7.1.6 Data Type Mapping	243
165	7.1.6.1 Rules for DataTypeMappingSet	245
170	7.2 DataPrototypes	246
175	7.2.1 Introduction	246

05	7.2.2 ParameterDataPrototype (Calibration Parameters)	246
	7.2.2.1 Usage of SW-CALIBRATION-ACCESS	246
	7.2.2.1.1 Rules for SwCalibAccess	247
10	7.2.2.2 Usage of SW-IMPL-POLICY	247
	7.2.2.2.1 Rules for SwImplPolicy	248
15	7.2.3 ParameterDataPrototype applicable for both ASW and BSW	248
	7.2.3.1 ParameterDataPrototype of Category Value	248
	7.2.3.1.1 Rules for ParameterDataPrototype of Category Value	249
20	7.2.4 Complex ParameterDataPrototype	252
	7.2.4.1 ParameterDataPrototype of Category Array	252
	7.2.4.1.1 Rules for ParameterDataPrototype of Category Array	252
25	7.2.4.2 ParameterDataPrototype of Category Structure	252
	7.2.4.2.1 Rules for ParameterDataPrototype of Category Structure	253
30	7.2.5 VariableDataPrototype (Measurement Points)	253
	7.2.6 VariableDataPrototype applicable for both ASW and BSW of Category Value	253
	7.2.6.1 VariableDataPrototype of Category Value	253
	7.2.6.1.1 Rules for VariableDataPrototype of Category Value	254
35	7.2.6.2 VariableDataPrototype of Category Boolean	254
	7.2.6.2.1 Rules for VariableDataPrototype of Category Boolean	254
40	7.2.7 Complex VariableDataPrototype	255
	7.2.7.1 VariableDataPrototype of Category Array	255
	7.2.7.1.1 Rules for VariableDataPrototype of Category Array	255
	7.2.7.2 VariableDataPrototype of Category Structure	255
	7.2.7.2.1 Rules for VariableDataPrototype of Category Structure	256
45	7.3 Usage of Attributes of SwDataDefProps	256
	7.3.1 Rules for SwDataDefPropsUsage	256
50	7.4 Initial Values	258
	7.4.1 Description	259
	7.4.1.1 Initial Values for Calibration Parameter Overview	259
55	7.4.1.2 Assigning Initial Values for Calibration Parameter	259
	7.4.1.2.1 Initial Value assigned to individual Calibration Parameter	260
60	7.4.2 Rules for InitValues	260
65	7.5 Data Constraints	260
	7.5.1 Types of DataConstraint	262
	7.5.2 Limits in DataConstraint	262
	7.5.3 Rules for DataConstraint	262

05	7.6 Computation Methods	264
10	7.6.1 Rules for CompuMethods	269
15	7.6.2 IDENTICAL CompuMethod	272
20	7.6.2.1 Rules for IdenticalCompuMethod	272
25	7.6.3 LINEAR CompuMethod	273
30	7.6.3.1 Rules for LINEAR CompuMethod.....	273
35	7.6.4 RAT_FUNC CompuMethod	275
40	7.6.4.1 Rules for RatFuncCompuMethod	275
45	7.6.5 TEXTTABLE CompuMethod	276
50	7.6.5.1 Handling of Enumeration Data Types.....	277
55	7.6.5.2 Rules for TextTableCompuMethod	278
60	7.7 Record Layout	279
65	7.8 Addressing Methods	279
70	7.8.1 Rules	279
75	7.9 Unit and PhysicalDimension	279
80	7.9.1 Unit.....	279
85	7.9.2 Physical Dimension	281
90	7.9.3 Rules for Units and PhysicalDimension	282
95	7.10 UnitGroup	285
100	7.10.1 Rules forUnitGroup	285
105	7.11 Overview of AUTOSAR Categories, Data types and related Elements.....	286
110	8 Rule Set: Basis Software Module Description (BSWMD)	290
115	8.1 Common Aspects of BSWMD	290
120	8.2 BSWMD Model View	294
125	8.2.1 Top Layers of BSWMD	294
130	8.2.1.1 BswModuleDescription	295
135	8.2.1.2 BswInternalBehavior	299
140	8.2.1.3 BswImplementation	301
145	8.2.2 Splitting of BSWMD Files	305
150	8.2.3 API Description and Visibility from Model Point of View	310
155	8.2.4 Differences between BSWMD and SWCD	311
160	8.3 BSWMD Use Cases.....	312

05	8.3.1 BSW Measurement and Calibration Support.....	313
	8.3.1.1 Definition of Measurement Variables and Measurement Points	314
	8.3.1.2 Definition of Calibration Parameters	316
10	8.3.2 Scheduling of BSW via RTE.....	319
	8.3.2.1 Mandatory Elements for Scheduling	320
15	8.3.2.2 BSW Scheduler Manager (SchM)	321
	8.3.2.3 Mapping Between BSW and a Corresponding SWC	321
20	8.3.3 BSW Documentation.....	325
	8.3.4 BSW Interface Elements	326
	8.3.4.1 BSW Module API Description (BswModuleEntry)	326
25	8.3.4.1.1 Common Attributes of a BswModuleEntry.....	326
	8.3.4.1.2 Definition of a Return Value and Arguments of a BswModuleEntry	332
	8.3.4.1.3 Export of a BswModuleEntry	345
30	8.3.4.2 Properties of a Code Fragment (BswModuleEntity)	347
	8.3.4.3 BswEvents	356
35	8.3.4.4 BswModes	364
	8.3.4.4.1 BSW Mode Management on the Mode Manager Side	365
	8.3.4.4.2 BSW Mode Management on the Mode User Side.....	367
	8.3.4.4.3 BSW Mode Access	372
40	8.3.4.5 BswTriggers	375
	8.3.4.5.1 BswExternalTriggers on the Sender side	375
	8.3.4.5.2 BswExternalTriggers on the Receiver side	377
	8.3.4.5.3 BswInternalTriggers.....	379
45	8.3.5 BSW Service Needs.....	382
	8.3.6 BSW Exclusive Areas	382
50	9 Rule Set: Execution Order Constraint	383
	9.1 Introduction to ExecutionOrderConstraint	383
	9.2 ExecutionOrderConstraint Description.....	384
	9.3 Upcoming of ExecutionOrderConstraint	391
55	10 Rule Set: AUTOSAR Package Structure	393
	10.1 Introduction to Use of ARPackage	393
60	10.1.1 Splitting ARPackages into different files.....	394
	10.1.2 Referencing ARElements	394
	10.1.3 ReferenceBases	394
65	10.2 Main Structure of ARPackages	395

05	10.3 ARPackage for Basic Software.....	402
10	10.4 Referencing Elements	405
15	10.4.1 References to other ARElements	405
20	10.4.2 Use of ReferenceBases	406
25	11 Rule Set: Central Elements	408
30	11.1 Centralization of Element Kinds	408
35	11.1.1 Tabular Overview of Central Elements	408
40	11.2 How to use the Central Elements	409
45	11.3 Delivery Aspects	416
50	12 Rule Set: CLF (Classification File)	417
55	12.1 Introduction	417
60	12.1.1 CLF projects overview	417
65	12.1.2 Tool Environment	417
70	12.1.3 References	418
75	12.2 Guidelines	418
80	12.2.1 Structural Conventions	418
85	12.2.2 Naming Conventions	421
90	12.3 Examples for Variants Usage	424
95	12.3.1 Simple enabling / disabling of folders	424
100	12.3.2 Using configuration variants	424
105	A Rules summary	430
110	B Rules Derivation	499
115	C List of Basic Software Modules	508
120	D Physical and Logical Types	511
125	E HIS Metrics Overview	513
130	F References	515
135	F.1 Robert Bosch GmbH AUTOSAR Guidelines	515
140	F.2 Robert Bosch GmbH AUTOSAR Artifacts	515
145	F.3 AUTOSAR Main Documents.....	515
150	F.4 AUTOSAR Basic Software Architecture and Runtime Documents.....	515
155	F.5 AUTOSAR Methodology and Templates Documents.....	516
160	F.6 AUTOSAR Application Interfaces Documents.....	516

05	F.7 Others	516
10	G Approval of Guideline	517
15	H Documentation	518
20	I Technical Terms	519
25	ACTIVITY	519
30	AUTOSAR	519
35	AUTOSAR ITEMS	519
40	AUTOSAR RULES	522
45	BOSCH RULES	523
50	Codes	523
55	Control elements	524
60	MISRA_RULES	524
65	Others	525
70	Products	525
50	Standard	525
55	Tool	525
60	Variables	526
65	XML/SGML-Attributes	526
70	XML/SGML-Tags	526
50	J Version Information	528

a Scope

These guidelines are obligatory for all BSW software delivered by CDG-SMT but also may be used for BSW development outside CDG-SMT. An usage outside of the CDG-SMT has currently limits because some topics (e.g. naming convention) are still CDG-SMT specific. A rework for a fully usage within BBM could be done but must be requested.

Existing software developed within CDG-SMT 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 the development of BSW software which are standardized from AUTOSAR and BSW modules where no specification from AUTOSAR is available. 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 conform 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.

Hint 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.

05 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.)

15 a.1.2 Rule Structure

20 Rules are defined for restricting the usage of the AUTOSAR meta model or to add additional rules like for example naming rules to make modeling with AUTOSAR applicable and consistent.

25 The rules follow a detailed and precise structure as described below.

30 For all rules defined in a guideline a rule summary will be created at the end of the document. Rules from other documents may be referenced but will not be listed. An overview of all possible rule prefixes is given in [/Guidelines \[Guidelines\]](#).

35 First we give an example for a rule and then the formal structure of a rule.

40 a.1.2.1 Rule Example

45 Rule Exa_007: Driving on the right side of the road

Instruction

50 **Scope:** Germany, Austria, France

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

60 This is absolutely helpful for your health.

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

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

75 a.1.2.2 Rule Structure Description

80 {Rule_Id}

85 Each rule has a unique Id.

90 Rule Ids will never change. When a rule is deleted its id will not be reused. If numbers are part of the rule id they might be not in a sequence.

95 {Rule Title}

100 The headline of the rule.

105 {Scope of the rule} (optional)

110 If the rule is applicable only under certain restrictions this is given as scope. Scope may be an organisational unit, an ECU domain, a specific AUTOSAR release etc..

120 If the rule is applicable to whole UBK / BBM, this is not expressed by scope. Our guidelines have this scope in general.

125 If the rule is applicable for all sub releases of AUTOSAR Release 4, this is not expressed by scope. Our guidelines have this scope in general.

130 The values of scope follow c-identifier syntax (but a "." is allowed additionally, so that version numbers could be expressed).

05 A union set for scopes is expressed by a comma separated list (e.g. "AS, PS"). An intersection is expressed by "/". (The "/" has to be evaluated first.) E.g: "AS, PS/ASW" is to be interpreted as: "Rule is to be followed in AS completely and in PS only for ASW".

10 The set of valid scope values is not strictly defined. It could express organization (should be avoided), product lines, AR releases.

15 Release ranges will be expressed as "[From|To]{ARRelease}", e.g. "FromAR4.0.3" or "FromAR4.0.3/ToAR4.1.1". "From" and "To" are to be interpreted as *including* this release. If "From" or "To" is missing, no restriction is meant.

20 {Class of the rule} (optional) The (currently) only usage of rule's class is to mark a rule as naming convention related. In this case a rule is marked as **Class:** NamingConvention.

25 Other classes might be defined in the future.

20 {Status of the rule} (optional) The status of the rule could be one of the values: preliminary, valid, obsolete, removed. (This list of possible states is derived from AUTOSAR's life cycle states.)

25 The status "valid" is default, if no status is given explicitly the rule is valid.

25 A rule in status "removed" still remains in our guideline for back tracing. Typically the content of the rule is removed and the title of the rule is also marked as "removed".

30 {DerivedFrom} (optional, multiple) If a rule is derived from another Bosch rule or from an AUTOSAR rule (constraint, specification item) this is mentioned here. The id and the "long name" (title) is given. Each referenced rule is given in a separate line.

35 Example: says that "State may be used to describe the status of an object." A derived rule might define that a specific object shall have a state and which values are allowed.

35 {RelatedTo} (optional, multiple) If a rule is related to another Bosch rule or from an AUTOSAR rule (constraint, specification item) this is mentioned here. The id and the "long name" (title) is given. Each referenced rule is given in a separate line.

40 Example: A rule expressing the file structure for an ASW (application software) component might be related to a rule expressing the file structure for a BSW (basic software) module.

45 {Rule Definition}

45 Short and concise description of the rule.

45 {Additional Information}

45 Here you find detailed explanations, reasons, backgrounds, use cases and examples for the defined rule.

50 a.1.3 Rule Set Structure

50 Rule Set Description

55 The BSW coding guideline is 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.

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

65 a.2 Used Guideline Artifacts

65 This Guideline includes contents of the following AUTOSAR guideline artifacts:

- 05 ▶ AR Packages Guideline (V1.5.2), [Chapter 10 "Rule Set: AUTOSAR Package Structure"](#), provided by Klaus Hünerfeld (CDG-SMT/ESS1)
- 10 ▶ CEL Central Elements (V1.5), [Chapter 11 "Rule Set: Central Elements"](#), provided by Klaus Hünerfeld (CDG-SMT/ESS1)
- 15 ▶ ECU Configuration Guideline (V1.2), [Chapter 4 "Rule Set: ECU Configuration"](#), provided by Frank Neukam (CDG-SMT/ESE1)
- 20 ▶ Pragma Concept (V1.3.0), [Chapter 3.2.3 "Abstraction of Addressing Keywords"](#) and [Chapter 3.2.4 "Abstraction of Memory Mapping"](#), provided by Gerd Grass (CDG-SMT/ESM2)
- 25 ▶ DataDescription (V1.8) [Chapter 7 "Rule Set: Data Description"](#), provided by Shwetha Rangaswamy (RBEI/EMT5)
- 30 ▶ DocumentReferences (V2.0) [Chapter F "References"](#)
- 35 ▶ GenericAspects (V1.3) [Chapter a.1.1 "Used Keywords"](#) and [Chapter a.1.2 "Rule Structure"](#), provided by Klaus Hünerfeld (CDG-SMT/ESS1)
- 40 ▶ ExecutionOrderConstraint (V1.0) [Chapter 9 "Rule Set: Execution Order Constraint"](#), provided by Claus-Peter Pflieger (DGS-EC/ESB2)

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

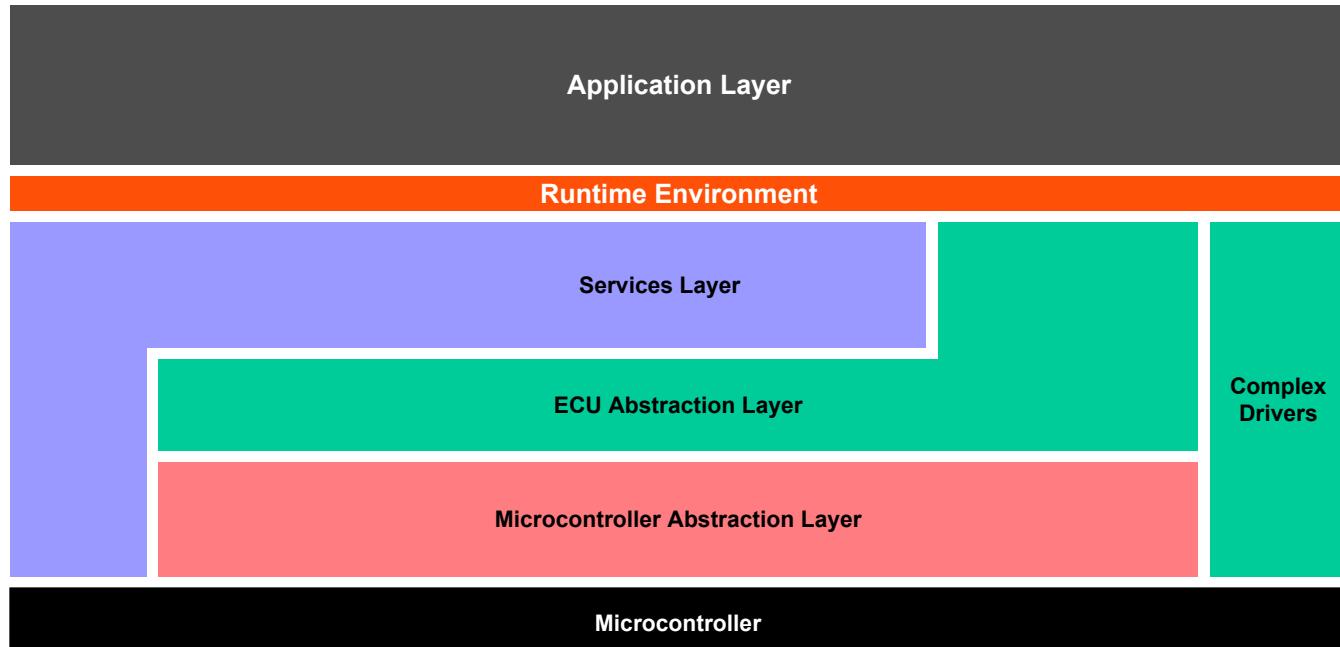
- 45 ▶ [Chapter 6 "Rule Set: oAW Rules"](#), provided by Marc Schreier (CDG-SMT/ESS2)
- 50 ▶ [Chapter 3.5.7 "Code Re-entrancy"](#) and [Chapter 3.5.9 "BSW Scheduler and Exclusive Areas"](#), provided by Jörg Häcker (CDG-SMT/ESS2)
- 55 ▶ [Chapter 3.2.10 "Ensure Usability of BSW Modules in C++ Environments"](#), provided by Jost Brachert (CDG-SMT/ESS2)
- 60 ▶ [Chapter 12 "Rule Set: CLF \(Classification File\)"](#), provided by Marc Schreier (CDG-SMT/ESS2)

All other chapters and rule sets are provided by Volker Kairies (CDG-SMT/ESM1).

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 µC. Implementation is µC dependent, interfaces are standardized and µC independent.

APIs of the **ECU Abstraction Layer** offer access to peripherals and devices regardless of their location (µC internal/external) and their connection to the µC (port pins, type of interface). This layer makes the higher software layers independent from the *ECU* hardware layout. Implementation is µC independent and interfaces are independent from the µC 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 excluding I/O, those are located in ECU Abstraction Layer). Implementation is mostly *ECU* and µC 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, µC 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 **Basic 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
BBM	Business Sector Mobility Solutions
BSW	Basic Software
BSWMD	Basis Software Module Description
BSWMĐT	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
C90	ISO/IEC 9899:1999 Programming languages – C
cdgb	CDG Build
chk	Check script for oAW based code generation (ECU Configuration)
CLF	Classification File
component	Is used synonymously to module
CUBAS	Common UBK BAsic Software
Dem	Diagnostic Event Manager
DSERAP	Dynamic SERial Application with Programing
ECU	Electronic Control Unit
ECUC	ECU Configuration
EOC	Execution Order Constraint
FIM	Function Inhibition Manager
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, Script language for ECU Configuration
PDU	Protocol Data Units
RTE	Runtime Environment
SDU	Service Data Unit
SWC	Software Component
SWCT	Software Component Template
TP	Transport Protocol

Terms	Definitions
UBK	Germ., Unternehmensbereich Kraftfahrzeugtechnik (Automotive Technology Business Sector) --> Is now replaced by BBM
Xpand	Template for oAW based code generation (ECU Configuration)
Xtend	Template for oAW based code generation extensions (ECU Configuration)
µC	Microcontroller

3 Rule Set: C Coding Guidelines

This rule set contains rules for basic elements of the C language. Generally all software shall be written conforming to ISO C90/C99, but not all features and possibilities of the C language may be used. The restrictions are made to increase quality, maintainability and readability of the code. In this context MISRA C standard is also used. The usage of that standard supports to avoid 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 C language based rules, other rule sets fulfill other technical topics.

3.1 Rule Set: C Language Rules

3.1.1 General Language Topics

Rule CCode_001: Conformity to C Standard

Instruction

DerivedFrom: MISRA C:2012 Rule 1.1

DerivedFrom: MISRA C:2012 Rule 1.2

The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits. Language extensions should not be used.

With the MISRA C: 2012 standard the allowed C Standard languages are C90 and C99. That means not that all features and extensions of C99 are allowed. There are still restrictions which are limited by the BSW Coding Guideline. 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 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
- long long is 64 bit (if long long is available)

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 [\[CCode_Types_001\] p. 96](#)) is the smallest addressable memory unit
- unsigned char (resp. uint8 corresponding to [\[CCode_Types_001\] p. 96](#)) 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

- 05 – 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
- 10 4. Used language extensions
 – If the long long data type is available then the LL suffix for 64 bit constant definition is used (see [\[CCode_Types_004\] p. 98](#))
 – Inline (encapsulated in a central macro definition) (see [\[Abstr_Inline_001\] p. 82](#))
 – "://" C++ style comments (see [\[CCode_Comments_002\] p. 162](#))
 – Line ends are defined as CR/LF (DOS/Windows style) (see [\[CCode_002\] p. 20](#))
 – The basic source and execution character set is extended with "\$" (0x24) and "@" (0x40). (see [\[CCode_002\] p. 20](#))
 – Any element of a unit can be initialized and not only the first one (C99 chapter 6.7.8 constraint 16).(This is the base for [\[CCode_Struct_001\] p. 48](#) to be able to read from an union member after it is written resp. initialized.)
- 15 25
 30
 35

Rule CCode_002: Character Set, Escape Sequences and Trigraphs

Instruction

40 **DerivedFrom:** MISRA C:2012 Rule 4.2

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

45 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").

50 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.

55 **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 ??]".

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

65 For the source character set there is a small exception to ISO C90. Tabulator characters are not allowed because they cause sometimes problems in the tool chain. Instead of a tabulator whitespace shall be used (see [\[CCode_Style_005\]](#)).

70 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:

26 uppercase letters of the Latin alphabet:

A (0x41) **B** (0x42) **C** (0x43) **D** (0x44) **E** (0x45) **F** (0x46) **G** (0x47) **H** (0x48) **I** (0x49) **J** (0x4A)
K (0x4B) **L** (0x4C) **M** (0x4D) **N** (0x4E) **O** (0x4F) **P** (0x50) **Q** (0x51) **R** (0x52) **S** (0x53) **T** (0x54)
U (0x55) **V** (0x56) **W** (0x57) **X** (0x58) **Y** (0x59) **Z** (0x5A)

26 lowercase letters of the Latin alphabet:

a (0x61) **b** (0x62) **c** (0x63) **d** (0x64) **e** (0x65) **f** (0x66) **g** (0x67) **h** (0x68) **i** (0x69) **j** (0x6A)
k (0x6B) **l** (0x6C) **m** (0x6D) **n** (0x6E) **o** (0x6F) **p** (0x70) **q** (0x71) **r** (0x72) **s** (0x73) **t** (0x74)
u (0x75) **v** (0x76) **w** (0x77) **x** (0x78) **y** (0x79) **z** (0x7A)

10 decimal digits:

0 (0x30) **1** (0x31) **2** (0x32) **3** (0x33) **4** (0x34) **5** (0x35) **6** (0x36) **7** (0x37) **8** (0x38) **9** (0x39)

30 graphic characters:

! (0x21) **"** (0x22) **#** (0x23) **%** (0x25) **&** (0x26) **'** (0x27) **(** (0x28) **)** (0x29) ***** (0x2A) **+** (0x2B)
, (0x2C) **-** (0x2D) **.** (0x2E) **/** (0x2F) **:** (0x3A) **;** (0x3B) **<** (0x3C) **=** (0x3D) **>** (0x3E) **?** (0x3F)
[(0x5B) **** (0x5C) **]** (0x5D) **^** (0x5E) **_** (0x5F) **{** (0x7B) **|** (0x7C) **}** (0x7D) **~** (0x7E) **Space** (0x20)

Additional graphic characters (not defined in ISO C90 but in ASCII):

\$ (0x24) **@** (0x40)

Escape sequences (to be used within execution character set):

\a (0x07) alert
\b (0x08) backspace
\f (0x0C) form feed
\n (0x0A) new line
\r (0x0D) carriage return
\t (0x09) horizontal tab
\v (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

DerivedFrom: MISRA C:2012 Rule 4.1

DerivedFrom: MISRA C:2012 Rule 7.1

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

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. The following initializations will show few examples.

```
code[1] = 109;      /* OK: Equivalent to decimal 109      */
code[2] = 100;      /* OK: Equivalent to decimal 100      */
code[3] = 052;      /* Not OK: Equivalent to decimal 42  */
code[4] = 071;      /* Not OK: 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.

A hexadecimal escape sequence shall be terminated by either:

- ▶ The start of another escape sequence, or
- ▶ The end of the character constant or the end of a string literal

There is potential for confusion if a hexadecimal escape sequence is followed by other characters. For example, the character constant '\x1f' consists of a single character whereas the character constant '\x1g' consists of the two characters '\x1' and 'g'. The manner in which multi-character constants are represented as integers is implementation-defined. The potential for confusion is reduced if every hexadecimal escape sequence in a character constant or string literal is terminated.

Example:

```
const char *s1 = "\x41g";      /* Not OK */
const char *s2 = "\x41" "g";   /* OK: Terminated by end of literal */
const char *s3 = "\x41\x67";   /* OK: Terminated by another escape */
```

Rule CCode_004: Usage and Handling of Bit Fields

Instruction

DerivedFrom: MISRA C:2012 Rule 6.1

DerivedFrom: MISRA C:2012 Rule 6.2

Bit-fields shall only be declared with an appropriate type ("unsigned int" or "signed int"). 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 and no certain order of the bits inside the bit field shall be assumed.

Single-bit named bit fields shall not be of a signed type.

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

According to the C99 Standard Section 6.2.6.2, a single-bit signed bit-field has a single (one) sign bit and no (zero) value bits. In any representation of integers, 0 value bits cannot specify a meaningful value. A single-bit signed bit-field is therefore unlikely to behave in a useful way and its presence is likely to indicate programmer confusion. Although the C90 Standard does not provide so much detail regarding the representation of types, the same considerations apply as for C99.

Rule CCode_005: Documentation of Implementation-defined Behaviour

Instruction

DerivedFrom: MISRA C:2012 Dir 1.1

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.

05
Rule CCode_006: File Handles are not Allowed10
Instruction15
DerivedFrom: MISRA C:2012 Rule 22.320
DerivedFrom: MISRA C:2012 Rule 22.425
DerivedFrom: MISRA C:2012 Rule 22.530
DerivedFrom: MISRA C:2012 Rule 22.635
File handles and file streams shall not be used.40
File handles and file streams are not useful for BSW software. There is no use case known where file handles are needed. AUTOSAR provides other concepts like the NVRAM manager to store data on an ECU. With this rule some MISRA C:2012 rules are implicitly fulfilled because it is not allowed to use file handles and file streams. In detail the rules are:

- 45
-
- ▶ The same file shall not be open for read and write access at the same time on different streams
-
- ▶ There shall be no attempt to write to a stream which has been opened as read-only (mandatory rule)
-
- ▶ A pointer to a FILE object shall not be dereferenced
-
- ▶ The value of a pointer to a FILE shall not be used after the associated stream has been closed

50
Rule CCode_007: Check of Validity of Values Passed to Functions55
Instruction60
DerivedFrom: MISRA C:2012 Dir 4.1165
The validity of values passed to functions shall be checked.70
This rule is on the one side relevant for the call of a function and on the other side for the implementation of a function. But there is only the need to check the validity of parameters on one of both sides. The software developer shall provide appropriate checks of input values for all functions which have a restricted input domain. Because a check costs run time the necessity of such a check has to be verified. If a check is made internally in function this shall be mentioned in the description of a function. In this case a user must not place a similar check in the code before the function is called. Otherwise if it is known, for example, that a called function cannot handle negative values such values shall be excluded by an explicit check before the function is called. Is is better to have one check to much than one to little.75

3.1.2 MISRA and HIS Metrics Conformity

80
Rule CCode_MisraHIS_001: Conformity to MISRA Standard and HIS Metrics85
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.90
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:2012 have to be followed. Deviations are allowed only in exceptional cases. Each deviation 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 deviations from MISRA are documented in a special deviation 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.95
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.

05 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.

10 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.

15 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.

20 Rule CCode_MisraHIS_002: Commenting MISRA Violations in C Code

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

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

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

```
/* MR12 RULE X.Y VIOLATION: Reason why the MISRA rule could not be followed in this special case */
or
/* MR12 DIR X.Y VIOLATION: Reason why the MISRA directive could not be followed in this special case */
X.Y represents the MISRA rule number.
```

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

45 The MISRA rule or directive number is different to the rule number of this coding guidelines. The reason is that all rules within this guideline are referring to the existing rule sets which are independent from the derivation of the rules. It is mentioned in the instruction of a rule with the tag "DerivedFrom" if a rule of the BSW Coding Guideline is derived from one or more MISRA C:2012 rule or directive. Additionally an overview of the derivations is also given in the list "[Rules Derivation](#)" in the appendix. But no panic, in most cases an exception of a MISRA rule or directive is found by a static check (QAC Tool). Here the corresponding MISRA rule is stated.

50 With the comment shown above also multiple MISRA rules or directives can be commented (if needed). In this case more than one rule number can be set separated by commas. In this case also multiple reasons has to be mentioned. Same for directives or a mix of rules and directives. Example:

```
/* MR12 RULE 8.1, 16.1, 20.5 VIOLATION: ... */
```

55 In general a MISRA comment is active until the next MISRA comment is placed or a new line is set.

60 **Hint** With the introduction of MISRA C:2012 standard the comment for MISRA rules violations was changed. The previous definition of the comment is still processible from the tool but shall be replaced by the now comment step by step. The old definition was specified as: /* MISRA RULE X.Y VIOLATION: Reason... */.

65 Rule CCode_MisraHIS_003: Commenting MISRA Violations for a Complete Module

70 **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.

05 Only in technically reasonable and exceptional cases MISRA violations are permissible. Such violations of MISRA rules shall be clearly identified and documented in the integration chapter 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.

10 It is important, that any restriction in the usage of the component is clearly documented. E.g. bit field members are used with a size greater than 16 bits which is violation of [\[CCode_004\]](#).

Rule CCode_MisraHIS_004: Commenting HIS Metric Violations in C Code

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

20 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.

25 Because the related function name is mentioned within the comment the comment itself can be placed somewhere in the file where the related function is specified. The syntax of the comment is specified as followed:

/* HIS METRIC XY VIOLATION IN FunctionName: Reason **why** a HIS Metric limit is exceeded in **this** case */

XY represents the name of the HIS Metric. The name can be found in the "["HIS Metrics Overview"](#)" list in the appendix. If more than one HIS metric is violated by a function more than one HIS Metrics can be set separated by a comma.

30 FunctionName: Name of function (in camel case, similar to the real name of the function) where the HIS metric is violated. It is only allowed that function based HIS metrics could be commented.

35 Modifications in code which are done to fulfill the HIS Metrics can sometimes cause higher resource consumption. This could be a reason to comment a HIS Metric instead of having enlarged code. But it should be considered 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 worsened (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.

40 3.1.3 Initializations

Rule CCode_Init_001: Initialization of Variables

45 **Instruction**

DerivedFrom: MISRA C:2012 Rule 9.1

All variables shall not be read before they have been set.

50 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 an unspecified value. An initialization has to be done but this does not necessarily require that the initialization has to be done at declaration time. A distinction can be made between the following types of variables:

55 ▶ 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).

65 ▶ 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

05 explicitly at all times before such variables are used within the function. Such variables are normally located to registers of the processor.

Examples:

```

10 #define MYMODULE_START_SEC_VAR_INIT_32
# include "MyModule_MemMap.h"
static uint32 staticvar1_u32 = 10L; /* Explicitly initialized variable */
...
15 #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 */
...
20 #define MYMODULE_STOP_SEC_VAR_CLEARED_32
# include "MyModule_MemMap.h"

void MyModule_Func(void)
{
    25     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. */
    ...
30     localvar2_u32 = localvar1_u32; /* Not OK: Uninitialized variable is used */
ExtModule_Func(localvar1_u32); /* Not OK: External function called with */
    ...
    /* uninitialized variable */
35     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 */
...
40     localvar3_u32 = staticvar2_u32; /* OK, staticvar2_u32 is at least set to 0 */
    ...
}

```

45 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 integration chapter in the product documentation. It shall be noticed, that this is in this case a violation of MISRA C:2012 Rule 1.1.

Rule CCode_Inits_002: Initialization of Enumerators

Instruction

55 **DerivedFrom:** MISRA C:2012 Rule 8.12

Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.

60 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

05 to ensure that all values are in the required and allowed range, and that values are not unintentionally duplicated. This rule requires that any replication of enumeration constants be made explicit, thus making the intent clear.

10 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.

15 Examples:

```

typedef enum
{
    MYCODE_OK_E,
    MYCODE_NOT_OK_E,
    MYCODE_BUSY_E
} MyModule_Enum_ten;
/* OK because automatic initialization */
/* is used. */

```

```

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 */

```

```

typedef enum
{
    MYCODE_OK_E = -1,
    MYCODE_NOT_OK_E = -2,
    MYCODE_BUSY_E = -3
} MyModule_Enum_ten;
/* OK because all elements are */
/* initialized */

```

```

typedef enum
{
    MYCODE_OK_E = 3,
    MYCODE_NOT_OK_E = 4,
    MYCODE_BUSY_E = 5
} MyModule_Enum_ten;
/* OK because all elements are */
/* initialized */

```

```

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 */

```

```

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 */

```

40 Additional information about the location of enumerators:

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

```

typedef enum
{
    RBA_GTM_SYNCPWM_MODE_DSBL_E,
    RBA_GTM_SYNCPWM_MODE_ASYNC_E,
    RBA_GTM_SYNCPWM_MODE_SYNC_E
} rba_Gtm_SyncPwm_Mode_ten;

```

```
rba_Gtm_SyncPwm_Mode_ten TestCd_SyncPwm_Mode_en = RBA_GTM_SYNCPWM_MODE_DSBL_E;
```

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

```
IFX: 0x7000ef0b 0x7000ef0b      1 g TestCd_SyncPwm_Mode_en
JDP: 0x40007890 0x40007893      4 g TestCd_SyncPwm_Mode_en
```

55 When changing the enumeration definition to an init value of 300

```

typedef enum
{
    RBA_GTM_SYNCPWM_MODE_DSBL_E = 300,
    RBA_GTM_SYNCPWM_MODE_ASYNC_E,
    RBA_GTM_SYNCPWM_MODE_SYNC_E
} rba_Gtm_SyncPwm_Mode_ten;

```

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

```
IFX: 0x70002bce 0x70002bcf      2 g TestCd_SyncPwm_Mode_en
```

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

Rule CCode_Inits_003: Initializer Lists without Side Effects

Instruction

Scope: C99 based code

DerivedFrom: MISRA C:2012 Rule 13.1

Initializer lists shall not contain persistent side effects.

C90 constrains the initializers for automatic objects with aggregate types to contain only constant expressions. However, C99 permits automatic aggregate initializers to contain expressions that are evaluated at run-time. It also permits compound literals which behave as anonymous initialized objects. The order in which side effects occur during evaluation of the expressions in an initializer list is unspecified and the behaviour of the initialization is therefore unpredictable if those side effects are persistent.

Examples:

```

volatile uint16 GlobVar1_u16;

void MyFunc1(void)
{
    uint16 a[2] = {GlobVar1_u16, 0};      /* Not OK: Volatile access is persistent side effect */
}

void MyFunc2(uint16 x, uint16 y)
{
    uint16 a[2] = {x + y, x - y};        /* OK: No side effects */
}

uint16 GlobVar2_u16 = 0u;
extern void MyFunc3(uint16 a[2]);

void MyFunc4(void)
{
    MyFunc3((uint16[2]) {GlobVar1_u16++, GlobVar1_u16++}); /* Not OK: Two side effects */
}

```

Rule CCode_Inits_004: Singular Initializations of Objects

Instruction

Scope: C99 based code

DerivedFrom: MISRA C:2012 Rule 9.4

An element of an object shall not be initialized more than once.

This rule applies to initializers for both objects and subobjects. The provision of designated initializers in C99 allows the naming of the components of an aggregate (structure or array) or of a union to be initialized within an initializer list and allows the object's elements to be initialized in any order by specifying the array indices or structure member names they apply to (elements having no initialization value assume the default for uninitialized objects).

Care is required when using designated initializers since the initialization of object elements can be inadvertently repeated leading to overwriting of previously initialized elements. The C99 Standard does not specify whether the side effects in an overwritten initializer occur. In order to allow sparse arrays and structures, it is acceptable to only initialize those which are necessary to the application.

Examples:

```

sint16 arr1[5] = { -5, -4, -3, -2, -1 };      /* OK array is -5, -4, -3, -2, -1
/* Similar behaviour using designated initializers                                         */

```

```

05    sint16 arr2[5] = {[0] = -5, [1] = -4, [2] = -3, [3] = -2, [4] = -1 };      /* OK      */
10
15    /* Repeated designated initializer element values overwrite earlier ones          */
16    sint16 arr3[5] = {[0] = -5, [1] = -4, [2] = -3, [2] = -2, [4] = -1 };      /* Not OK */
17
18    struct MyStruct
19    {
20        uint32 a;
21        uint32 b;
22        uint32 c;
23        uint32 d;
24    };
25
26    struct MyStruct s1 = { 100, -1, 42, 999 }; /* OK struct is 100, -1, 42, 999      */
27
28    /* Similar behaviour using designated initializers                            */
29    struct MyStruct s2 = {.a = 100, .b = -1, .c = 42, .d = 999};      /* OK      */
30
31    /* Repeated designated initializer element values overwrite earlier ones          */
32    struct MyStruct s3 = {.a = 100, .b = -1, .a = 42, .d = 999};      /* Not OK */
33

```

25 Rule CCode_Inits_005: Usage of Designated Initializers for Arrays

Instruction

Scope:

30 **DerivedFrom:** MISRA C:2012 Rule 9.5

Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.

35 The rule applies equally to an array subobject that is a flexible array member. If the size of an array is not specified explicitly, it is determined by the highest index of any of the elements that are initialized. When using designated initializers it may not always be clear which initializer has the highest index, especially when the initializer contains a large number of elements. To make the intent clear, the array size shall be declared explicitly. This provides some protection if, during development of the program, the indices of the initialized elements are changed as it is a constraint violation (C99 Section 6.7.8) to initialize an element outside the bounds of an array.

40 Examples:

```

45    uint32 Array1[] = { [ 0 ] = 1 }; /* Not OK: probably unintentional to have single element */
46
47    uint32 Array2[10] = { [ 0 ] = 1 }; /* OK */

```

3.1.4 Expressions

50 Rule CCode_Expr_001: Precedence of Operators within Expressions

Instruction

55 **DerivedFrom:** MISRA C:2012 Rule 12.1

The precedence of operators within expressions should be made explicit.

60 The following table is used in the definition of this rule.

Table 3 Precedences of all Operators and Operands

Description	Operator or Operand	Precedence
Primary	identifier, constant, string literal, (expression)	16 (high)
Postfix	[] () (function call) . -> ++ (post-increment) -- (post-decrement) () {} (C99: compound literal)	15
Unary	++ (pre-increment) -- (pre-decrement) & * + - ^ ! sizeof defined (preprocessor)	14

Description	Operator or Operand	Precedence
Cast	(())	13
Multiplication	* / %	12
Additive	+ -	11
Bitwise shift	<< >>	10
Relational	< > <= >=	9
Equality	== !=	8
Bitwise AND	&	7
Bitwise XOR	^	6
Bitwise OR		5
Logical AND	&&	4
Logical OR		3
Conditional	?:	2
Assignment	= *= /= %= += -= <<= >>= &= ^= =	1
Comma	,	0 (low)

The precedences used in this table are chosen to allow a concise description of the rule. They are not necessarily the same as those that might be encountered in other descriptions of operator precedence. For the purposes of this rule, the precedence of an expression is the precedence of the element (operand or operator) at the root of the parse tree for that expression.

The C language has a relatively large number of operators and their relative precedences are not intuitive. This can lead less experienced programmers to make mistakes. Using parentheses to make operator precedence explicit removes the possibility that the programmer's expectations are incorrect. It also makes the original programmer's intention clear to reviewers or maintainers of the code.

It is recognized that overuse of parentheses can clutter the code and reduce its readability. This rule aims to achieve a compromise between code that is hard to understand because it contains either too many or too few parentheses.

Examples:

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

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

x = a == b ? a : a - b;           /* Not OK: confusing */
x = (a == b) ? a : (a - b);     /* OK: non confusing */
```

Rule CCode_Expr_002: Value of Expressions

Instruction

DerivedFrom: MISRA C:2012 Rule 13.2

The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders

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

05 evaluation of an expression at which all previous side effects can be guaranteed to have taken place are called 'sequence points'. Sequence points are summarized in Annex C of both the C90 and C99 standards. The sequence points in C90 are a subset of those in C99. Full expressions are defined in Section 6.6 of the C90 standard and Section 6.8 of the C99 standard.

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

15 Examples:

▶ Increment and decrement operators

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

20 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.

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

▶ Functions

25 The order of evaluation of function arguments is unspecified

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

30 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.

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

35 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.

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

40 Could be written as:

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

▶ Nested assignment statements

45 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. The following examples are not recommended:

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

Rule CCode_Expr_003: Usage of Sizeof Operator

Instruction

55 **DerivedFrom:** MISRA C:2012 Rule 13.6

55 The operand of the sizeof operator shall not contain any expression which has potential side effects.

60 Any expressions appearing in the operand of a sizeof operator are not normally evaluated. This rule mandates that the evaluation of any such expression shall not contain side effects, whether or not it is actually evaluated. A function call is considered to be a side effect for the purposes of this rule.

65 The operand of a sizeof operator may be either an expression or may specify a type. If the operand contains an expression, a possible programming error is to expect that expression to be evaluated when it is actually not evaluated in most circumstances. The C90 standard states that expressions appearing in the operand are not evaluated at run-time. In C99, expressions appearing in the operand are usually not evaluated at run-time. However, if the operand contains a

variable-length array type then the array size expression will be evaluated if necessary. If the result can be determined without evaluating the array size expression then it is unspecified whether it is evaluated or not.

An expression of the form `sizeof (V)`, where `V` is an lvalue with a volatile qualified type that is not a variable-length array, is permitted.

10 Examples:

```

05 uint16 Var1_u16;
uint16 Var2_u16;
uint16 Var3_u16;
10 volatile uint16 Var4_u16;

15 Var1_u16 = sizeof(Var2_u16);      /* OK. The size of the uint16 variable is returned */
Var2_u16 = sizeof(Var3_u16++);     /* Not OK. sizeof over an expression           */
19 Var3_u16 = sizeof(MyFunc());      /* Not OK. sizeof over a function call        */
Var1_u16 = sizeof(Var4_u16);       /* OK. This is an exception                  */
20

```

Rule CCode_Expr_004: Logical and Conditional Operators without Side Effects

Instruction

25 **DerivedFrom:** MISRA C:2012 Rule 13.5

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

30 The evaluation of the right-hand operand of the `&&`, `||` and `?:` operators is conditional on the value of the left-hand operand. If the right-hand operand contains side effects than those side effects may or may not occur which may be contrary to programmer expectations. If evaluation of the right-hand operand would produce side effects which are not persistent at the point in the program where the expression occurs then it does not matter whether the right-hand operand is evaluated or not.

35 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.

40 ▶ Logical operators '`&&`' and '`||`:

45 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.

50 Examples:

```

55 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 */

```

55 ▶ 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;
```

60 `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

65 **DerivedFrom:** MISRA C:2012 Rule 12.1

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

05 '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
 10 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
 15 only logical `||`, extra parentheses are not required. See also rule [\[CCode_Expr_001\]](#) to get an overview of the precedences
 of the different operators and operands.

20 Examples:

```
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

25 **Instruction**

DerivedFrom: MISRA C:2012 Rule 10.1

25 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 '`? :`'.

30 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 the
 35 following arithmetic or bitwise operators shall not be used: `+`, `++`, `-`, `--`, `*`, `/`, `<<`, `>>`, `~`.

35 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 */
```

40 45 This rule is also related to the Essential Type Model "[Essential Type Model](#)" p. 58 and is a sub use case of rule [\[CCode_Essential_001\]](#) p. 61 .

Rule CCode_Expr_007: Type of Bitwise Operators

50 **Instruction**

DerivedFrom: MISRA C:2012 Rule 10.1

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

60 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
 65 bit.

65 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.

70 This rule is also related to the Essential Type Model "[Essential Type Model](#)" p. 58 and is a sub use case of rule [\[CCode_Essential_001\]](#) p. 61 .

Rule CCode_Expr_008: Usage of Shift Operator

Instruction

DerivedFrom: MISRA C:2012 Rule 12.2

The right hand operand of a shift operator shall lie in the range: "zero" to "width in bits of essential type minus one" of the left hand operand.

If the right hand operand is negative, or greater than or equal to the width of the left hand operand, then the behaviour is undefined. If, for example, the left hand operand of a left-shift or right-shift is a 16-bit integer, then it is important to ensure that this is shifted only by a number in the range 0 to 15. See "[Essential Type Model](#)" p. 58 for a description of essential type and the limitations on the essential types for the operands of shift operators. 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 */

1u << 10u;          /* Not OK: 1u is essentially unsigned char */
((uint16)1u) << 10u;  /* OK because of cast */
1UL << 10u;        /* OK: 1UL is essentially unsigned long */
```

Rule CCode_Expr_009: Usage of Unary Minus Operator

Instruction

DerivedFrom: MISRA C:2012 Rule 10.1

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 */
```

This rule is also related to the Essential Type Model "[Essential Type Model](#)" p. 58 and is a sub use case of rule [\[CCode_Essential_001\]](#) p. 61 .

Rule CCode_Expr_010: Usage of Comma Operator

Instruction

DerivedFrom: MISRA C:2012 Rule 12.3

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:

```

05 var1_u8 = (var2_u8++, var3_u8++);      /* Not OK because code is abstruse          */
                                             /* OK because separate statements are clearer */
var2_u8++;
var1_u8 = var3_u8++;

10 #define MYMODULE_MACRO(Val, Struct) ((Struct)->X = 0, (State)->Y = (Val)) /* OK   */
for (I = 0; I++, K++; I < 100)           /* OK, K is an additional counter which is */
...                                         /* used within the for-loop                 */

```

15 Rule CCode_Expr_011: Usage of Constant Unsigned Integer Expression

Instruction

DerivedFrom: MISRA C:2012 Rule 12.4

20 Evaluation of constant expressions should not lead to unsigned integer wrap-around

This rule applies to expressions that satisfy the constraints for a constant expression, whether or not they appear in a context that requires a constant expression. If an expression is not evaluated, for example because it appears in the right 25 operand of a logical AND operator whose left operand is always false, then this rule does not apply.

Unsigned integer expressions do not strictly overflow, but instead wrap-around. Although there may be good reasons to 30 use modulo arithmetic at run-time, it is less likely for its use to be intentional at compile-time. 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 35 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:

```

35 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 */
                                             /* */

40 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 */
                                             /* 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; */
                                             /* */


```

50 Rule CCode_Expr_012: Usage of Increment and Decrement Operator

Instruction

DerivedFrom: MISRA C:2012 Rule 13.3

55 A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator.

60 A function call is considered to be a side effect for the purposes of this rule. All sub-expressions of the full expression are treated as if they were evaluated for the purposes of this rule, even if specified as not being evaluated by the C standard.

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

05

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

10

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.

15

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

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

20

The following sequence is clearer and therefore safer:

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

25

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.

25

Examples:

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

30

Rule CCode_Expr_013: Value of Floating Complex Expression

35

Instruction

Status: removed

ReplacedBy: [\[CCode_Essential_008\]](#)

35

Rule CCode_Expr_014: Implicit Integer Type Conversions

40

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

45

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

50

- ▶ 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

50

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.

55

Examples:

60

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

65

```
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 */
/* */
```

70

```
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:

```
uint8 Var_u8;  
uint16 Var_u16;
```

```
Var_u16 = 100;
Var_u8 = (uint8)Var_u16;      /* OK: Value of Val_u16 is in range of uint8 */

Var_u16 = 1000;
Var_u8 = (uint8)Var_u16;      /* Not OK: Value of Val_u16 is outside of range of uint8 */
```

3.1.5 Control Statement Expressions

Rule CCode Control 001: Boolean Expressions

Instruction

DerivedFrom: MISRA C:2012 Rule 13.4

The result of an assignment operator should not be used.

This rule applies even if the expression containing the assignment operator is not evaluated. The use of assignment operators, simple or compound, 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 making it more difficult to avoid an undefined behaviour

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:

```
a[x] = a[x = y];           /* Not OK because the value of x = y is used */

a = b = c = 0;             /* Not OK values of c = 0 and b = c = 0 are used */

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 the 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 || 16)

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

Rule CCode_Control_002: Value Test Expression

Instruction

DerivedFrom: MISRA C:2012 Rule 14.4

The controlling expression of an if-statement and the controlling expression of an iteration-statement shall have essentially Boolean type.

Strong typing requires the controlling expression of an if statement or iteration–statement to have essentially Boolean type. This rule is in the interests of clarity, and makes clear the distinction between integers and logical values. E.g. 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.

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:

```
25  if (x_u32 != 0)      /* OK, test if x_u32 is non zero          */
if (x_u32)           /* Not OK, explicit test value is missing      */
                    /* No boolean context                         */
```

Examples for boolean context:

```
30  boolean var_b;

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

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

Hint The comparison (var_b == TRUE) is not ok because a check for equality to TRUE could be ambiguous. The representation for the logical true state can be everything unequal to 0. Only the logical false state is defined with one explicit representation which is 0. TRUE could be defined in different ways (e.g. "1", or "!FALSE"). The comparison between var_b and TRUE works only if var_b and TRUE is provided/defined identically. var_b could be the result of an equation or the compiler sets it to something unequal to 0 to express a logical true state. This could not match to TRUE which is e.g. specified with "1". That is the reason why the comparison (var_b == TRUE) shall not be used. Only the comparison (var_b != FALSE) works every time to check the logical true state. This comparison is independent from the definition of TRUE and everything unequal to 0 represents unambiguous the logical true state.

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:

```

05 float32 x_f32, y_f32;

if (x_f32 == y_f32)      /* Not OK because of float comparison */
...
10 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      */

15 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 meantime 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.

25

Rule CCode_Control_004: For Loop Expression

Instruction

DerivedFrom: MISRA C:2012 Rule 14.2

A for loop shall be well-formed: The three expressions of a for-statement shall be concerned only with loop control.

The three clauses of a for statement are the:

First clause which

- ▶ Shall be empty, or
- ▶ Shall assign a value to the loop counter

Second clause which

- ▶ Shall be an expression that has no persistent side effects, and
- ▶ Shall use the loop counter and optionally loop control flags, and
- ▶ Shall not use any other object that is modified in the for loop body

Third clause which

- ▶ Shall be an expression whose only persistent side effect is to modify the value of the loop counter, and
- ▶ Shall not use objects that are modified in the for loop body

There shall only be one loop counter in a for loop, which shall not be modified in the for loop body. A loop control flag is defined as a single identifier denoting an object with essentially Boolean type that is used in the Second clause. The behaviour of a for loop body includes the behaviour of any functions called within that statement. The for statement provides a general-purpose looping facility. Using a restricted form of loop makes code easier to review and to analyse.

Exception: All three clauses may be empty, for example for (; ;), so as to allow for infinite loops.

Examples:

```

60 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 */
...
65 for (;;)                         /* OK, infinite loop                  */
...
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 is not initialized      */
                                              /* Not OK, if x_u8 is not used within the loop */
```

Rule CCode_Control_005: For Loop Iteration

Instruction

DerivedFrom: MISRA C:2012 Rule 14.2

A for loop shall be well-formed: 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 */
}
```

Rule CCode_Control_006: Non Floating Loop Counters

Instruction

DerivedFrom: MISRA C:2012 Rule 14.1

A loop counter shall not have essentially floating type.

When using a floating-point loop counter, accumulation of rounding errors may result in a mismatch between the expected and actual number of iterations. This can happen when a loop step that is not a power of the floating-point radix is rounded to a value that can be represented. Even if a loop with a floating-point loop counter appears to behave correctly on one implementation, it may give a different number of iterations on another implementation. This rule is valid for all kinds of loops (like for, while, do-while loops).

Example:

```
for (x_f32 = 0.0F; x_f32 < 10.0f; x_f32++) /* Not OK because of floating types */

uint32 counter_u32 = 0;
for (x_f32 = 0.0f; x_f32 < 1.0f; x_f32 += 0.001f)
{
    counter_u32++;                         /* Not OK, the value of counter is unlikely */
                                             /* to be 1000 at the end of the loop */

float32 x_f32 = 0.0f;
while (x_f32 < 1.0f)
{
    x_f32 += 0.001f;                      /* Not OK, x_f32 is a loop counter here */
```

Rule CCode_Control_007: Non Invariant Controlling Expressions

Instruction

DerivedFrom: MISRA C:2012 Rule 14.3

Controlling expressions shall not be invariant.

This rule applies to:

- ▶ Controlling expressions of if, while, for, do ... while and switch statements
- ▶ The first operand of the ?: operator

If a controlling expression has an invariant value, it is possible that there is a programming error. Any code that cannot be reached due to the presence of an invariant expression may be removed by the compiler. This might have the effect of removing defensive code, for instance, from the executable.

Exceptions:

- ▶ Invariants that are used to create infinite loops are permitted
- ▶ A do ... while loop with an essentially Boolean controlling expression that evaluates to 0 is permitted

Example:

```

Var_s8 = (Var_u16 < 0u) ? 0 : 1; /* Not OK: uint16 is always >= 0 */
30
if (Var_u16 <= 0xffffu)           /* Not OK: Is always true */ ...
{
    ...
}
35
if ( 2 > 3 )                   /* Not OK: Is always false */ ...
{
    ...
}
40
for (Var_s8 = 0;Var_s8 < 130;Var_s8++)
{
    /* Not OK: Abort criterion is always true */
    ...
}
45
while(TRUE)                     /* OK: Compliant by exception 1 */
{
    ...
}
50
do                            /* OK: Compliant by exception 2 */
{
    ...
} while (0);

```

3.1.6 Control Flow

Rule CCode_CntrFlow_001: Unreachable Code

Instruction

DerivedFrom: MISRA C:2012 Rule 2.1

A project shall not contain unreachable code.

05 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.

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

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

Rule CCode_CntrFlow_002: Duplication of Code

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

25 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

30 **Instruction**

DerivedFrom: MISRA C:2012 Rule 2.2

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

40 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

45 **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.

50 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. The 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.

55 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 ; */  
}  
  
; 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 Statement

Instruction

DerivedFrom: MISRA C:2012 Rule 2.6

DerivedFrom: MISRA C:2012 Rule 15.1

The 'goto' statement shall not be used. Additionally goto labels shall not be defined.

Any program that uses a 'goto' statement can be rewritten to give the same control flow without this statement. Unconstrained use of goto could lead to programs that are unstructured and extremely difficult to understand. Also HIS Metric Measurement disallows the usage of goto statements (see [Chapter E "HIS Metrics Overview" p. 513](#)).

Rule CCode_CntrFlow_006: Iteration Statement

Instruction

DerivedFrom: MISRA C:2012 Rule 15.4

There should be no more than one break statement used to terminate any iteration statement.

This rule is in the interests of good structured programming. Restricting the number of exits from a loop helps to minimize visual code complexity. The use of one break statement allows a single secondary exit path to be created when early loop termination is required.

Both of the following nested loops are compliant as each has a single break used for early loop termination.

```
for (x = 0; x < 10; ++x)
{
    if (ExitNow(x))
    {
        break;
    }
    for (y = 0; y < x; ++y)
    {
        if (Exit Now (10 - y))
        {
            break;
        }
    }
}
```

Rule CCode_CntrFlow_007: If ... Else

Instruction

DerivedFrom: MISRA C:2012 Rule 15.7

All if ... else if constructs shall be terminated with an else statement.

A final else statement shall always be provided whenever an if statement is followed by a sequence of one or more else if constructs. The else statement shall contain at least either one side effect or a comment. Terminating a sequence of if ... else if constructs with an else statement is defensive programming and complements the requirement for a default clause in a switch statement (see Rule [CCode_Switch_004](#) p. 46). The else statement is required to have a side effect or a comment to ensure that a positive indication is given of the desired behaviour, aiding the code review process.

Note: A final else statement is not required for a simple if statement.

Example:

```
if (x_u8 < 0)
{
```

```

05     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 */
}

```

15

3.1.7 Switch Statements

20 Rule CCode_Switch_007: Conformance of Switch Statements

25 Instruction

DerivedFrom: MISRA C:2012 Rule 16.1

25 All switch statements shall be well-formed.

30 A switch statement shall be considered to be well-formed if it conforms to the subset of C switch statements that is specified by the syntax rules of this chapter. A switch statement shall only contain switch labels and switch clauses but no other code (see also Rule [\[CCode_Switch_001\] p. 45](#)). The style guide has to be considered too (see Rule [\[CCode_BlockStyle_001\] p. 160](#)).

```

35 switch(var_u8)
{
    uint32 var_u32;          /* Not OK: Variable definition within switch */
    var_u32 = 0;             /* Not OK: Expression outside switch clause */

40    case 1:
    {
        /* Do something */
    }
    break;

45    case 2:                  /* OK: Empty switch clause */
    case 3:
    {
        /* Do something */
    }
    break;

50    default:
    {
        /* Do something */
    }
    break;

55    case 4:                  /* Not OK: Switch case after default case */
    {
        /* Do something */
    }
    break
}

```

60 The syntax for the switch statement in C is not particularly rigorous and can allow complex, unstructured behaviour. The following rules impose a simple and consistent structure on the switch statement.

65

Rule CCode_Switch_001: Prohibition of Declarations or Definitions between Case and Default Clauses

Instruction

DerivedFrom: MISRA C:2012 Rule 16.1

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:

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

Rule CCode_Switch_002: Position of Switch Labels

Instruction

DerivedFrom: MISRA C:2012 Rule 16.2

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

The C standard permits a switch label, i.e. a case label or default label, to be placed before any statement contained in the body of a switch statement, potentially leading to unstructured code. In order to prevent this, a switch label shall only appear at the outermost level of the compound statement forming the body of a switch statement.

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

DerivedFrom: MISRA C:2012 Rule 16.3

An unconditional break statement shall terminate every switch-clause even if 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

05 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:

```
10    switch(var_u8)
{           case 0:          /* OK because of empty case clause */
15      case 1:
{           ...
}
break;          /* OK: unconditional 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 Label

30 Instruction

DerivedFrom: MISRA C:2012 Rule 16.4

DerivedFrom: MISRA C:2012 Rule 16.5

35 Every switch statement shall have a default label.

The default label shall appear as either the first or the last switch label of a switch statement.

40 The switch-clause following the default label shall, prior to the terminating break statement, contain either a statement or a comment.

45 The requirement for a default label is defensive programming. Any statements following the default label are intended to take some appropriate action. If no statements follow the label then the comment can be used to explain why no specific action has been taken.

```
45    switch(var_u8)
{
    case 0:
    case 1:
    {
        /* Do something */
    }
break;

...
55

default:          /* OK: default is exactly the last clause */
{
    /* Do something */
}
break;
}

switch(var_u8)
{
    default:          /* OK: default can also be the first clause */
{
    /* Do something */
}
```

```

05     }
06     break;

07     case 0:
08     case 1:
09     {
10         /* Do something */
11     }
12     break;

13     ...
14 }

}

```

Rule CCode_Switch_005: Condition for Switch Expressions

Instruction

DerivedFrom: MISRA C:2012 Rule 16.7

A switch-expression shall not have essentially Boolean type.

The C standard requires the controlling expression of a switch statement to have an integer type. Since the type that is used to implement Boolean values is an integer, it is possible to have a switch statement controlled by a Boolean expression. In this instance an if-else construct would be more appropriate.

Examples:

```

30 switch (var1_u8 == var2_u8)      /* Not OK because boolean expression */
31 ...
32 switch (var_b)                  /* Not OK because boolean variable */
33 ...
34 switch (var1_u8)                /* OK because of non-boolean variable */
35 ...
36 switch (var_en)                /* OK because of non-boolean variable */
37 ...

```

Rule CCode_Switch_006: Usage of Switch Statements

Instruction

DerivedFrom: MISRA C:2012 Rule 16.6

Every switch statement shall have at least two switch-clauses.

A switch statement with a single path is redundant and may be indicative of a programming error. There is one exception, 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.

Examples:

```

55 switch (var1_u8)
56 {
57     default:                      /* Not OK switch is redundant */
58     {
59         ...
60     }
61     break;
62 }

63 switch (var2_u8)
64 {
65     case 1:
66     default:                      /* Not OK switch is redundant */
67     {
68         ...
69     }
70 }

```

```

05      }
       break;
    }

10     switch (var3_u8)
    {
        case 1:
        {
            ...
        }
        break;
        default:           /* OK two cases available */
        {
            ...
        }
        break;
    }

```

3.1.8 Structures and Unions

Rule CCode_Struct_001: Usage of Unions

Instruction

30 **DerivedFrom:** MISRA C:2012 Rule 19.1

30 **DerivedFrom:** MISRA C:2012 Rule 19.2

35 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.

40 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. However, if a union member is written and then a different union member is read back, the behaviour depends on the relative sizes of the members:

- ▶ If the member read is wider than the member written then the value is unspecified
- ▶ Otherwise, the value is implementation-defined

45 Additionally the behaviour is undefined when two objects are created which have some overlap in memory and one is assigned or copied to the other.

The following example will show the allowed and forbidden behaviour:

```

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

55 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;
}

65 uint8 MyModule_Func2(void)
{
    MyModule_MyUnion_tun TstUnion_un;

```

```

05     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.      */
10    Tst_u8 = TstUnion_un.Part_u8[2];
15
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. */
}

```

20 Rule CCode_Struct_002: Typedef Declarations for Structures and Unions

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

25 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
{
30     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
{
40     uint32 Dword_u32;
     uint8 Byte_au8[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];

```

50 Rule CCode_Struct_003: Completion of Structure and Union Types

Instruction

DerivedFrom: MISRA C:2012 Rule 1.3

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

60 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.

65 Example:

```

05    struct tnode * pt;           /* Not OK, tnode is incomplete at this point */
...
10
11    struct tnode
12    {
13        int count;
14        struct tnode *left;    /* OK, pointer to incomplete type is allowed */
15        struct tnode * right; /* OK, pointer to incomplete type is allowed */
16    };                      /* OK, type tnode is now complete */
...
19    struct MyStruct           /* Not OK, no members in this struct */
20    {
21    };

```

Rule CCode_Struct_004: Avoidance of Alignment Gaps inside Structures

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

25 Some µCs 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.

30 Example (based on a µC which only supports natural alignment):

```

30
31    typedef struct
32    {
33        uint8 xTestval1_u8;          /* Uses 1 byte in memory */
34        /*- Here a gap byte can occur */
35        uint16* adrData_pu16;       /* Pointer uses 4 bytes in memory */
36        uint8 xTestval2_u8;          /* Uses 1 byte in memory */
37        /*- Here a gap byte can occur */
38        uint16 stMachine1_u16;       /* Uses 2 bytes in memory */
39    } Example_tst;
40
41
42    Possible alternative 1:           Possible alternative 2:
43    typedef struct
44    {
45        uint16* adrData_pu16;
46        uint16 stMachine1_u16;
47        uint8 xTestval1_u8;
48        uint8 xTestval2_u8;
49    } Example_tst;

```

3.1.9 Preprocessing Directives

Rule CCode_Prop_001: Handling of #include Statements

55 **Instruction**

DerivedFrom: MISRA C:2012 Rule 20.1

#include directives should only be preceded by preprocessor directives or comments.

60 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.

65 An exception is the include of memmap header files ("<Mip>_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_Prop_002: Prohibition of Non-Standard Characters in #include Directives

Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 20.2

The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.

The behaviour is undefined if:

- ▶ The ', " or \ characters, or the /* or // character sequences are used between < and > delimiters in a header name preprocessing token
- ▶ The ' or \ characters, or the /* or // character sequences are used between the " delimiters in a header name preprocessing token

Note: Although use of the \ character results in undefined behaviour, many implementations will accept the / character in its place.

Example:

```
#include "fi'le.h"      /* Not OK */
```

Rule CCode_Prop_003: #include followed by a File Name

Instruction

DerivedFrom: MISRA C:2012 Rule 20.3

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_Prep0_004: Prohibition of #undef

Instruction

DerivedFrom: MISRA C:2012 Rule 20.5

#undef should 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_Prep0_005: Handling of Preprocessor Operator "defined"

Instruction

DerivedFrom: MISRA C:2012 Rule 1.3

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           */
#define 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_Prep0_006: Correctness of Preprocessing Directives

Instruction

DerivedFrom: MISRA C:2012 Rule 20.13

A line whose first token is # shall be a valid preprocessing directive.

05 A preprocessor directive may be used to conditionally exclude source code until a corresponding #else, #elif or #endif directive is encountered. A malformed or invalid preprocessing directive contained within the excluded source code may not be detected by the compiler, possibly leading to the exclusion of more code than was intended. Requiring all preprocessor directives to be syntactically valid, even when they occur within an excluded block of code, ensures that
10 this cannot happen.

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

25 Rule CCode_Prepo_007: Hold Preprocessor Directives together

Instruction

30 **DerivedFrom:** MISRA C:2012 Rule 20.14

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

35 Confusion can arise when blocks of code are included or excluded by the use of conditional compilation directives which are spread over multiple files. Requiring that a #if directive be terminated within the same file reduces the visual complexity of the code and the chance that errors will be made during maintenance.

40 Note: #if directives may be used within included files provided they are terminated within the same file.

45 Example:

```
file.c
#define A
...
45 #ifdef A
    #include "file1.h"
#endif
...
50 #if 1
    #include "file2.h"
...
EOF

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

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

Rule CCode_Propre_008: Defined Identifiers in control Expressions

Instruction

DerivedFrom: MISRA C:2012 Rule 20.9

All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.

As well as using a #define preprocessing directive, identifiers may effectively be #define'd in other, implementation-defined, ways. For example some implementations support:

- ▶ Using a compiler command-line option, such as -D to allow identifiers to be defined prior to translation
- ▶ Using environment variables to achieve the same effect
- ▶ Pre-defined identifiers provided by the compiler

If an attempt is made to use a macro identifier in a preprocessing directive, and that identifier has not been defined, then the preprocessor will assume that it has a value of zero. This may not meet developer expectations.

Examples:

The following macros are defined to be used within a preprocessing directive:

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

The preprocessing directive:

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

The preprocessor generates the following result:

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

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

Hint: U will be recognized.

```
#if (M == 0)           /* Not OK: does 'M' expand to zero or is it undefined? */
#endif

#if defined ( M )      /* OK: M is not evaluated          */
#if (M == 0)           /* OK: M is known to be defined   */
/* 'M' must expand to zero.        */
#endif
#endif
```

Rule CCode_Propre_009: Evaluation of #if and #elif Preprocessing Directives

Instruction

DerivedFrom: MISRA C:2012 Rule 20.8

The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.

This rule does not apply to controlling expressions in preprocessing directives which are not evaluated. Controlling expressions are not evaluated if they are within code that is being excluded and cannot have an effect on whether code is excluded or not. Strong typing requires the controlling expression of conditional inclusion preprocessing directives to have a Boolean value.

Examples:

```
#if FALSE           /* OK      */
#endif

#if 10              /* Not OK */

```

```

05 #endif

60 #if !defined(X) /* OK */
#endif

65 #if A > B      /* OK if assuming A and B are numeric */
#endif

```

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:

```

35 #define PI          3.14159F           /* OK, constant           */
40 #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
45             {
50                 DISABLE_INTERRUPTS();
55                 time_now = (uint32)TIMER_HI << 16;
60                 time_now = time_now | (uint32)TIMER_LO;
65                 ENABLE_INTERRUPTS();
70             } while (0)      /* OK, example of do-while-zero */
75
80 #define int32_t    long   /* Not OK, use typedef instead */
85 #define CAT        PI     /* Not OK, non parenthesised expression */
90 #define MY_IF      if(   /* Not OK, unbalanced () and language redefinition */

```

Rule CCode_Macro_002: #undef of C Macros

Instruction

Status: removed

Rule CCode_Macro_003: Handling of Function-like Macros

Instruction

DerivedFrom: MISRA C:2012 Rule 20.7

05 Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses unless they are used as operands of # or ##.

10 If the expansion of any macro parameter produces a token, or sequence of tokens, that form an expression then that expression, in the fully-expanded macro, shall either:

- ▶ Be a parenthesized expression itself or
- ▶ Be enclosed in parentheses

15 Note: this does not necessarily require that all macro parameters are parenthesized; it is acceptable for parentheses to be provided in macro arguments.

If parentheses are not used, then operator precedence may not give the desired results when macro substitution occurs.

20 If a macro parameter is not being used as an expression then the parentheses are not necessary because no operators are involved.

```
#define MACRO1(x, y) (x * y)
val = MACRO1(1 + 2, 3 + 4);
/* Expanded macro: */
val = (1 + 2 * 3 + 4); /* Not OK: This was not the intention of the macro */

#define MACRO2(x,y) ((x) * (y)) /* OK: Now the macro calculates correctly */
```

25 The following example is compliant because the first expansion of x is as the operand of the ## operator, which does not produce an expression. The second expansion of x is as an expression which is parenthesized as required.

```
#define MACRO3(x) a ## x = (x)
uint16 MACRO3(0)
/* Expanded macro: */
uint16 a0 = 0;
```

30 The following example is compliant because expansion of the parameter M as a member name does not produce an expression. Expansion of the parameter S produces an expression, with structure or union type, which does require parentheses.

```
#define GET_MEMBER(Structure,Member) (Structure).Member
v = GET_MEMBER(s1,minval);
/* Expanded macro: */
v = s1.minval;
```

45 Rule CCode_Macro_004: Calling of Function-like Macros

Instruction

DerivedFrom: MISRA C:2012 Rule 1.3

50 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.

55 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

DerivedFrom: MISRA C:2012 Rule 20.6

Tokens that look like a preprocessing directive shall not occur within a macro argument.

An argument containing sequences of tokens that would otherwise act as preprocessing directives leads to undefined behaviour.

Example:

```
#define MAX(A, B)    (((A) > (B)) ? (A) : (B))
...
20
sint16 MyModule_Func(sint16 arg1_s16, sint16 arg2_s16)
{
    sint16 value_s16;

    value_s16 = MAX(
        #ifdef MAGIC_SWITCH
            arg1_s16,
        #else
            arg2_s16,
        #endif
        10);           /* Not OK, because of preprocessor */
                    /* within argument list of macro. */
    return value_s16;
}
25
30
35
```

Rule CCode_Macro_006: Macros are no Replacements for Keywords

Instruction

DerivedFrom: MISRA C:2012 Rule 20.4

A macro shall not be defined with the same name as a keyword.

This rule applies to all keywords, including those that implement language extensions. Using macros to change the meaning of keywords can be confusing. The behaviour is undefined if a standard header is included while a macro is defined with the same name as a keyword.

Example:

```
#include "Std_Types.h"
#define uint8 some_other_type          /* Not OK: Redefined of existing type */

#define while(val) for ( ; (val) ; ) /* Not OK: Redefined while keyword */
Consider also rule \[CCode\_Macro\_001\] for valid macro definitions.
```

Rule CCode_Macro_007: Handling of # and ## Operators within Macros

Instruction

DerivedFrom: MISRA C:2012 Rule 20.11

A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.

The order of evaluation associated with multiple #, multiple ## or a mix of # and ## preprocessing operators is unspecified. In particular, the result of a # operator is a string literal and it is extremely unlikely that pasting this to any other preprocessing token will result in a valid token. Therefore the # and ## operator shall be used with care.

Examples:

```
#define A(x) #x      /* OK */
#define B(x,y) x ## y /* OK */
#define C(x,y) #x ## y /* Not OK */
```

Rule CCode_Macro_008: Handling of Macro Parameters used as Operands to # or ## Operators

Instruction

DerivedFrom: MISRA C:2012 Rule 20.12

A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators.

A macro parameter that is used as an operand of a # or ## operator is not expanded prior to being used. The same parameter appearing elsewhere in the replacement text is expanded. If the macro parameter is itself subject to macro replacement, its use in mixed contexts within a macro replacement may not meet developer expectations.

In the following non-compliant example, the macro parameter x is replaced with AA which is subject to further macro replacement when not used as the operand of ##.

```
#define AA 0xffff
#define BB(x) (x) + wow ## x
void ByFunc(void)
{
    uint32 wowAA = 0;

    wowAA = BB(AA);           /* Not OK: Macro is called with another macro. */
                           /* This yield to the forbidden usage. */
    /* Expands as wowAA = (0xffff) + wowAA; */
}
```

In the following compliant example, the macro parameter X is not subject to further macro replacement.

```
uint32 speed;
uint32 speed_scale;
uint32 scaled_speed;

#define SCALE(X) ((X) * X ## _scale)

scaled_speed = SCALE(speed); /* OK: Parameter is not a macro and is
                           /* only used as operand of the ## operator */
/* Expands to scaled_speed = ((speed) * speed_scale); */
```

3.1.11 Essential Type Model

The essential type model is defined and introduced from MISRA C:2012 and will be applied within BSW completely. The essential type model replaces the underlying type concept from MISRA C:2004 which was not introduced for BSW.

The rules in this chapter collectively define the essential type model and restrict the C type system so as to:

1. Support a stronger system of type-checking
2. Provide a rational basis for defining rules to control the use of implicit and explicit type conversions
3. Promote portable coding practices
4. Address some of the type conversion anomalies found within ISO C.

The essential type model does this by allocating an essential type to those objects and expressions which ISO C considers to be of arithmetic type. For example, adding an int to a char gives a result having essentially character type rather than the int type that is actually produced by integer promotion.

05 The essential type of an object or expression is defined by its essential type category and size. The essential type category of an expression reflects its underlying behaviour and may be:

- 10 ▶ Essentially Boolean
 ▶ Essentially character
 ▶ Essentially signed
 ▶ Essentially unsigned
 ▶ Essentially floating
 ▶ Essentially enum

15 When comparing two types of the same type category, the terms *wider* and *narrower* are used to describe their relative sizes as measured in bytes. Two different types are sometimes implemented with the same size. The following [Table 4](#) shows how the standard integer types map on to essential type categories.

20

20 Table 4 Essential Type Categories Related to C and BSW Types

	Essential Type Category					
	Boolean	character	signed	unsigned	floating	enum<i>
25 C Types	_Bool	char	signed char signed short signed int signed long signed long long	unsigned char unsigned short unsigned int unsigned long unsigned long long	float double long double	named enum
30 BSW Types	boolean	char	sint8 sint16 sint32	uint8 uint16 uint32	float32 float64	named enum

35 **Hint** Within BSW code only the BSW types are allowed to be used (conform to ["Rule Set: Types and Symbols" p. 96](#)).
 The C types are only shown for completeness and because they are mentioned in the MISRA C:2012 documentation.

40 The signed and unsigned type of lowest rank (STLR and UTLR)

45 The concept of "rank" is particularly relevant when describing the set of signed and unsigned types; signed and unsigned long long share the "highest" rank and signed and unsigned char share the "lowest" rank. In the ISO C99 standard, "rank" is a term applied only to integer types.

50 The Essential Type Model introduces the concept of the *Signed Type of Lowest Rank* (the *STLR*) and the *Unsigned Type of Lowest Rank* (the *UTLR*). These allow integer constant expressions and bit-fields to be used within expressions having an essential type with a rank lower than that of int. In the case of integer constant expressions, this is a convenience as it avoids the need to cast the expression, or its operands, to a type with lower rank. Similarly, for an implementation that does not permit bit-fields to have a type whose rank is lower than that of int, it avoids the need to cast bit-fields in some situations.

- 55 1. The STLR is the signed type having the lowest rank required to represent the value of a particular integer constant expression or both the maximum and minimum values of a bit-field
2. The UTLR is the unsigned type having the lowest rank required to represent the value of a particular integer constant expression or the maximum value of a bit-field.

60 The essential type of bit-fields

65 The essential type of a bit-field is determined by the first of the following that applies:

1. For a bit-field which is implemented with an *essentially Boolean type* it is *essentially Boolean*
2. For a bit-field which is implemented with a *signed type* it is the *STLR* which is able to represent the bit-field
3. For a bit-field which is implemented with an *unsigned type* it is the *UTLR* which is able to represent the bit-field

The essential type of enumerations

Two distinct types of enumeration need to be considered:

1. A *named enum type* is an enumeration which has a *tag* or which is used in the definition of any object, function or type
2. An *anonymous enum type* is an enumeration which does not have a *tag* and which is not used in the definition of any object, function or type. This will typically be used to define a set of constants, which may or may not be related

A named enum type is distinct from all other *named enum types*, even if declared in an inner scope with exactly the same tag and enumeration constants. Each instance of a *named enum type* is denoted as *enum*<*i*> in this document, with the *i* being different for each such type. An alias for a *named enum type* created using *typedef* denotes that *named enum type* and is not a new type. The *essential type* of an *anonymous enum type* is its *standard type*.

The essential type of literal constants

The ISO C99 standard defines the following constants of integer type:

Integer constants

1. If the standard type of an integer constant is signed int then its essential type is the STLR
2. If the standard type of an integer constant is unsigned int then its essential type is the UTLR

Enumeration constants

The standard type of an enumeration constant in C is always int, regardless of the implemented type of the enumeration and regardless of the type of any initializer expression used to define its value. For example, if an enumeration constant is initialized with an unsigned value (e.g. 500U), the constant will still be considered to have a standard type of signed int. The essential type of an enumeration constant is determined as follows:

1. If an enumeration defines a named enum type then the essential type of its enumeration constants is enum<*i*>
 - 1.1 If a named enum type is used to define an essentially Boolean type then the essential type of its enumeration constants is essentially Boolean.
2. If an enumeration defines an anonymous enum type then the essential type of each enumeration constant is the STLR of its value.

Character constants

The standard type of a character constant (e.g. 'q', 'xy') is int, not char.

1. If a character constant consists of a single character then its essential type is char
2. Else the essential type is the same as its standard type

Boolean constants

The C99 standard provides no syntax to explicitly define a constant of boolean type. Within the BSW software the terms TRUE and FALSE are available to be used in connection with a boolean type (see rule [\[CCode_Symbols_001\]](#) p. [100](#)). The use of a cast to define a constant expression of Boolean type (e.g. (boolean)0) is only appropriate if the expression is not to be used in a #if or #elif preprocessing directive. Use of a cast within a preprocessing directive is a syntax error.

The essential type of expressions

The essential type of any expression not listed in this section is the same as its standard type.

Comma (,)

The essential type of the result is the essential type of the right hand operand.

Relational (< <= >= >), Equality (== !=) and Logical (&& || !)

The result of the expression is essentially Boolean.

Shift (<< >>)

1. If the left hand operand is essentially unsigned then:

1.1 If both operands are integer constant expressions then the essential type of the result is the UTLR of the result

1.2 Else the essential type of the result is the essential type of the left hand operand

2. Else the essential type is the standard type

Bitwise complement (~)

1. If the operand is essentially unsigned then:

1.1 If the operand is an integer constant expression then the essential type of the result is the UTLR of the result

1.2 Else the essential type of the result is the essential type of the operand

2. Else the essential type is the standard type

Unary plus (+)

1. If the operand is essentially signed or essentially unsigned then the essential type of the result is the essential type of the operand

2. Else the essential type is the standard type

Unary minus (-)

1. If the operand is essentially signed then:

1.1 If the expression is an integer constant expression then the essential type of the result is the STLR of the whole expression

1.2 Else the essential type of the result is the essential type of the operand.

Else the essential type is the standard type

Conditional (?:)

1. If the essential type of the second and third operands is the same then the result has the same essential type

2. Else if the second and third operands are both essentially signed then the essential type of the result is the essential type of the one with the highest rank

3. Else if the second and third operands are both essentially unsigned then the essential type of the result is the essential type of the one with the highest rank

4. Else the essential type is the standard type

Operations subject to the usual arithmetic conversions (* / % + - & | ^)

1. If the operands are both essentially signed then:

1.1 If the expression is an integer constant expression then the essential type of the result is the STLR of the result

1.2 Else the essential type of the result is the essential type of the operand with the highest rank.

2. Else if the operands are both essentially unsigned then:

2.1 If the expression is an integer constant expression then the essential type of the result is the UTLR of the result

2.2 Else the essential type of the result is the essential type of the operand with the highest rank.

3. Else the essential type is the standard type

Addition with char

1. If one operand is essentially character and the other is essentially signed or essentially unsigned then the essential type of the result is char

2. Else the essential type is the standard type

Subtraction with char

1. If the first operand is essentially character and the second is essentially signed or essentially unsigned then the essential type of the result is char

2. Else the essential type is the standard type

Rule CCode_Essential_001: Operands Conformity to Essential Type

Instruction

DerivedFrom: MISRA C:2012 Rule 10.1

Operands shall not be of an inappropriate essential type.

In the following table a number within a cell indicates where a restriction applies to the use of an essential type as an operand to an operator. These numbers correspond to the Rationale section below and indicate why each restriction is imposed.

Table 5 Overview of Rationales to Essential Type Categories

Operator	Operand	Essential type category of arithmetic operand					
		Boolean	character	signed	unsigned	floating	enum
[]	integer	3	4			1	
+ (unary)		3	4				5
- (unary)		3	4		8		5
+ -	either	3					5
* /	either	3	4				5
%	either	3	4			1	5
< > <= >=	either	3					
== !=	either						
! &&	any		2	2	2	2	2
<< >>	left	3	4	6		1	5,6
<< >>	right	3	4	7		1	7
~ \$ ^	any	3	4	6		1	5,6
?:	1st		2	2	2	2	2
?:	2nd and 3rd						

Under this rule the ++ and -- operators behave the same way as the binary + and - operators. Other rules place further restrictions on the combination of essential types that may be used within an expression.

Rationale:

1. The use of an expression of essentially floating type for these operands is a constraint violation.
2. An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value.
3. An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value.
4. An operand of essentially character type should not be used where an operand is interpreted as a numeric value. The numeric values of character data are implementation-defined.
5. An operand of essentially enum type should not be used in an arithmetic operation because an enum object uses an implementation-defined integer type. An operation involving an enum object may therefore yield a result with an unexpected type. Note that an enumeration constant from an anonymous enum has essentially signed type.
6. Shift and bitwise operations should only be performed on operands of essentially unsigned type. The numeric value resulting from their use on essentially signed types is implementation defined.
7. The right hand operand of a shift operator should be of essentially unsigned type to ensure that undefined behaviour does not result from a negative shift.
8. An operand of essentially unsigned type should not be used as the operand to the unary minus operator, as the signedness of the result is determined by the implemented size of int.

Exception: A non-negative integer constant expression of essentially signed type may be used as the right hand operand to a shift operator.

```
typedef enum Hugo_en
{
    A1,
    A2,
    A3
} Hugo1_en, Hugo2_en; /* Essentially enum<Hugo_en>
typedef enum
{
```

```
05      K1 = 1,
      K2 = 2
} Hugo_en;           /* Essentially signed */
```

The following examples are non-compliant. The comments refer to the numbered rationale item that results in the non-compliance.

```
10 Hugo_f32 & 2U          /* Rationale 1 - constraint violation */
Hugo_f32 << 2            /* Rationale 1 - constraint violation */
Hugo_chr && Hugo_b        /* Rationale 2 - char type used as a Boolean value */
Hugo_en ? a1 : a2         /* Rationale 2 - enum type used as a Boolean value */
Hugo_s8 && Hugo_b        /* Rationale 2 - signed type used as a Boolean value */
Hugo_u8 ? a1 : a2         /* Rationale 2 - unsigned type used as a Boolean value */
Hugo_f32 && Hugo_b        /* Rationale 2 - floating type used as a Boolean value */
Hugo1_b * Hugo2_b         /* Rationale 3 - Boolean used as a numeric value */
Hugo1_b > Hugo2_b         /* Rationale 3 - Boolean used as a numeric value */
Hugo1_chr & Hugo2_chr     /* Rationale 4 - char type used as a numeric value */
Hugo_chr << 1             /* Rationale 4 - char type used as a numeric value */
Hugo_en--                  /* Rationale 5 - enum type used in arithmetic operation */
Hugo_en * a1                /* Rationale 5 - enum type used in arithmetic operation */
Hugo_en += a1               /* Rationale 5 - enum type used in arithmetic operation */
Hugo_s8 & 2                 /* Rationale 6 - bitwise operation on signed type */
50 << 3U                   /* Rationale 6 - shift operation on signed type */
Hugo_u8 << Hugo_s8         /* Rationale 7 - shift magnitude uses signed type */
Hugo_u8 << -1              /* Rationale 7 - shift magnitude uses signed type */
-Hugo_u8                   /* Rationale 8 - unary minus on unsigned type */
```

30 The following examples are compliant:

```
35 Hugo1_b && Hugo2_b
Hugo_b ? Hugo1_u8 : Hugo2_u8
Hugo1_chr - Hugo2_chr
Hugo1_chr > Hugo2_chr
Hugo_en > a1
Hugo_s8 + Hugo_s16          /* Both variables are from essentially signed category */
-(Hugo1_s8) * Hugo2_s8
Hugo_s8 > 0
--Hugo_s16
Hugo_u8 + Hugo_u16
Hugo_u8 & 2U
Hugo_u8 > 0U
Hugo_u8 << 2U
Hugo_u8 << 1               /* Compliant by exception */
```

40

45

Rule CCode_Essential_002: Expression of Essentially Character Types

50 Instruction

DerivedFrom: MISRA C:2012 Rule 10.2

55 Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.

55

The appropriate uses are:

60

1. For the + operator, one operand shall have essentially character type and the other shall have essentially signed type or essentially unsigned type. The result of the operation has essentially character type.
2. For the – operator, the first operand shall have essentially character type and the second shall have essentially signed type, essentially unsigned type or essentially character type. If both operands have essentially character type then the result has the standard type (usually int in this case) else the result has essentially character type.

65

Expressions with essentially character type (character data) shall not be used arithmetically as the data does not represent numeric values. The uses above are permitted as they allow potentially reasonable manipulation of character data. For example:

70

- 05 ▶ Subtraction of two operands with essentially character type might be used to convert between digits in the range '0' to '9' and the corresponding ordinal value
- 10 ▶ Addition of an essentially character type and an essentially unsigned type might be used to convert an ordinal value to the corresponding digit in the range '0' to '9'
- 15 ▶ Subtraction of an essentially unsigned type from an essentially character type might be used to convert a character from lowercase to uppercase

Examples:

```

15   '0' + Hugo_u8      /* OK: Convert Hugo_u8 to digit      */
Hugo_s8 + '0'        /* OK: Convert Hugo_s8 to digit      */
Hugo_chr - '0'        /* OK: Convert Hugo_chr to ordinal    */
'0' - Hugo_s8        /* OK: Convert -Hugo_s8 to digit      */

20   Hugo_s16 - 'a'      /* Not OK: Hugo_s16 is outside of range of a digit */
'0' + Hugo_f32        /* Not OK: Arithmetic with float type      */
Hugo_chr + ':'         /* Not OK: Arithmetic with character type   */
Hugo_chr - Hugo_en     /* Not OK: Arithmetic with enumeration type */

```

Rule CCode_Essential_003: Possible Value Assignments

Instruction

DerivedFrom: MISRA C:2012 Rule 10.3

30 The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.

35 The C language allows the programmer considerable freedom and will permit assignments between different arithmetic types to be performed automatically. However, the use of these implicit conversions can lead to unintended results, with the potential for loss of value, sign or precision. The use of stronger typing, as enforced by the MISRA essential type model, reduces the likelihood of these problems occurring.

Exception:

- 40 1. A non-negative integer constant expression of essentially signed type may be assigned to an object of essentially unsigned type if its value can be represented in that type
2. The initializer { 0 } may be used to initialize an aggregate or union type

Examples:

```

45   Hugo_u8 = 0;          /* OK: By exception                  */
Hugo_b = (boolean)0;      /* OK: 0 is explicitly casted to proper type */
Hugo_b = TRUE;           /* OK: TRUE is essentially Boolean    */
Hugo_b = (Hugo1_u8 > Hugo2_u8); /* OK: Result of comparison is a boolean type */

50   Hugo_u8 = 2;          /* OK: By exception                  */
Hugo_u8 = 2 * 24;        /* OK: By exception                  */
Hugo_chr += 1;           /* OK: Hugo_chr = Hugo_chr + 1      */
                        /* assigns character to character    */
55   Hugo1_u8 = Hugo2_u8; /* OK: Same essential type          */
Hugo1_u8 = Hugo2_u8 + Hugo3_u8; /* OK: Same essential type          */
Hugo_u8 = (uint8)Hugo_s8; /* OK: Cast gives same essential type */
Hugo_u32 = Hugo_u16;     /* OK: Assignment to a wider essential type */
Hugo_u32 = 2U + 125U;   /* OK: Assignment to a wider essential type */

60   Hugo_u8 = 1.0f;       /* Not OK: unsigned and floating    */
Hugo_b = 0;              /* Not OK: boolean and signed       */
Hugo_chr = 7;             /* Not OK: character and signed     */
Hugo_u8 = 'a';            /* Not OK: unsigned and character   */
Hugo_u8 = 1 - 2;          /* Not OK: unsigned and signed       */
Hugo_u8 += 'a';           /* Not OK: Hugo_u8 = Hugo_u8 + 'a'  */
                        /* assigns character to unsigned    */

65

```

Rule CCode_Essential_004: Operands of Operators and Essential Type

Instruction

DerivedFrom: MISRA C:2012 Rule 10.4

Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.

This rule applies to operators that are described in usual arithmetic conversions (see C90 Section 6.2.1.5, C99 Section 6.3.1.8). This includes all the binary operators, excluding the shift, logical **&&**, logical **||** and comma operators. In addition, the second and third operands of the ternary operator are covered by this rule. The increment and decrement operators are not covered by this rule.

The C language allows the programmer considerable freedom and will permit conversions between different arithmetic types to be performed automatically. However, the use of these implicit conversions can lead to unintended results, with the potential for loss of value, sign or precision. The use of stronger typing, as enforced by the MISRA essential type model, allows implicit conversions to be restricted to those that should then produce the answer expected by the developer.

Exception:

The following are permitted to allow a common form of character manipulation to be used:

1. The binary **+** and **+=** operators may have one operand with essentially character type and the other operand with an essentially signed or essentially unsigned type
2. The binary **-** and **--** operators may have a left-hand operand with essentially character type and a right-hand operand with an essentially signed or essentially unsigned type

Example:

```

typedef enum          typedef enum
{                   {
    A1,             B1,
    A2,             B2,
    A3             B3
} HugoA_ten;         } HugoB_ten;

HugoA_ten HugoA_en;
HugoB_ten HugoB_en;

if (HugoA_en > A1)      /* OK: Because of same essential type category */
if (HugoB_en > A1)      /* Not OK: Different enum types */
if (HugoA_en == HugoB_en) /* Not OK: Different enum types */
Hugo_u8 + Hugo_u16       /* OK: Because of same essential type category */
Hugo_s8 += Hugo_u8        /* Not OK: Because of a mix of signed and unsigned */
Hugo_u8 + 2              /* Not OK: 2 is interpreted as signed */

```

Rule CCode_Essential_005: Value of Expressions and Essential Type

Instruction

DerivedFrom: MISRA C:2012 Rule 10.5

The value of an expression should not be cast to an inappropriate essential type.

The casts which should be avoided are shown in the following table, where values are cast (explicitly converted) to the essential type category of the first column.

Table 6 Overview of Casts Which Should be Avoided

Essential type category	from						
	to	Boolean	character	enum	signed	unsigned	floating
Boolean		Avoid		Avoid	Avoid	Avoid	Avoid
character							Avoid
enum	Avoid	Avoid		Avoid*	Avoid	Avoid	Avoid
signed	Avoid						
unsigned	Avoid						
floating	Avoid	Avoid					

* Note: an enumerated type may be cast to an enumerated type provided that the cast is to the same essential enumerated type. Such casts are redundant.

Casting from void to any other type is not permitted as it results in undefined behaviour.

An explicit cast may be introduced for legitimate functional reasons, for example:

- ▶ To change the type in which a subsequent arithmetic operation is performed
- ▶ To truncate a value deliberately
- ▶ To make a type conversion explicit in the interests of clarity

However, some explicit casts are considered inappropriate:

- ▶ In C99, the result of a cast or assignment to boolean is always 0 or 1. This is not necessarily the case when casting to another type which is defined as essentially Boolean
- ▶ A cast to an essentially enum type may result in a value that does not lie within the set of enumeration constants for that type
- ▶ A cast from essentially Boolean to any other type is unlikely to be meaningful
- ▶ Converting between floating and character types is not meaningful as there is no precise mapping between the two representations

Exception: An integer constant expression with the value 0 or 1 of either signedness may be cast to a type which is defined as essentially Boolean. This allows the implementation of non-C99 Boolean models.

Examples:

```
(uint32) 3U      /* OK: Compliant */
(boolean) 0      /* OK: Compliant - by exception */
(boolean) 3U      /* Not OK: Non-compliant */
(sint32) Hugo_en /* OK: Compliant */
```

Rule CCode_Essential_006: Composite Expression and Essential Type

Instruction

DerivedFrom: MISRA C:2012 Rule 10.6

The value of a composite expression shall not be assigned to an object with wider essential type.

Some type safety issues with C can be avoided by restricting the implicit and explicit conversions that may be applied to non-trivial expressions. These include:

- ▶ The confusion about the type in which integer expressions are evaluated, as this depends on the type of the operands after any integer promotion. The type of the result of an arithmetic operation depends on the implemented size of *int*
- ▶ The common misconception among programmers that the type in which a calculation is conducted is influenced by the type to which the result is assigned or cast. This false expectation may lead to unintended results

05 In addition to the previous rules [\[CCode_Essential_001\]](#) to [\[CCode_Essential_005\]](#), the essential type model places further restrictions on expressions whose operands are composite expressions, as defined below. The following are defined as composite operators:

- 10 ▶ Multiplicative (*, /, %)
 ▶ Additive (binary +, binary -)
 ▶ Bitwise (&, |, ^)
 ▶ Shift (<<, >>)
 15 ▶ Conditional (?:) if either the second or third operand is a composite expression

20 A compound assignment is equivalent to an assignment of the result of its corresponding composite operator. A composite expression is defined in this document as a non-constant expression which is the direct result of a composite operator. Note:

- 25 ▶ The result of a compound assignment operator is not a composite expression
 ▶ A parenthesized composite expression is also a composite expression
 ▶ A constant expression is not a composite expression

25 Examples:

```
Hugo_u16 = Hugo1_u16 + Hugo2_u16;           /* OK: Same essential type */
Hugo_u32 = (uint32)Hugo1_u16 + Hugo2_u16;   /* OK: Cast causes addition in uint32 context */
Hugo_u32 = Hugo1_u16 + Hugo2_u16;           /* Not OK: Implicit conversion on assignment */
                                                /* The addition is done in an uint16 context. */
                                                /* This could lead to an overrun which is not */
                                                /* expected. To avoid such an effect an cast */
                                                /* has to be set as shown in the example above. */
                                                /* Not OK: Implicit conversion on assignment */
                                                /* if parameter is not an uint16 */
```

Rule CCode_Essential_007: Composite Expression as Operand and Essential Type

Instruction

40 **DerivedFrom:** MISRA C:2012 Rule 10.7

45 If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type.

50 The same rationale description from Rule [\[CCode_Essential_006\]](#) is also valid for this rule.

55 Restricting implicit conversions on composite expressions means that sequences of arithmetic operations within an expression must be conducted in exactly the same essential type. This reduces possible developer confusion.

55 Note: this does not imply that all operands in an expression are of the same essential type.

The expression u32a + u16b + u16c is compliant as both additions will notionally be performed in type uint32. In this case only non-composite expressions are implicitly converted. The expression (u16a + u16b) + u32c is non-compliant as the left addition is notionally performed in type uint16 and the right in type uint32, requiring an implicit conversion of the composite expression u16a + u16b to uint32.

Examples:

```
Hugo_u32 * Hugo_u16 + Hugo_u16           /* OK: No composite conversion */
(Hugo_u32 * Hugo_u16) + Hugo_u16         /* OK: No composite conversion */
Hugo_u32 * ((uint32) Hugo_u16 + Hugo_u16) /* OK: Both operands of * have */
                                              /* same essential type */
Hugo1_u32 += (Hugo2_u32 + Hugo_u16);     /* OK: No composite conversion */

Hugo_u32 * (Hugo_u16 + Hugo_u16) /* Not OK: Implicit conversion of (Hugo_u16 + Hugo_u16) */
Hugo_u32 += (Hugo_u16 + Hugo_u16); /* Not OK: Implicit conversion of (Hugo_u16 + Hugo_u16) */
```

Rule CCode_Essential_008: Composite Expression as Operand and Essential Type

Instruction

DerivedFrom: MISRA C:2012 Rule 10.8

The value of a composite expression shall not be cast to a different essential type category or a wider essential type.

The same rationale description from Rule [\[CCode_Essential_006\]](#) is also valid for this rule.

Casting to a wider type is not permitted as the result may vary between implementations. Consider the following:

(uint32) (u16a + u16b)

On a 16-bit machine the addition will be performed in 16 bits with the result wrapping modulo-2 before it is cast to 32 bits. However, on a 32-bit machine the addition will take place in 32 bits and would preserve high-order bits that would have been lost on a 16-bit machine. Casting to a narrower type with the same essential type category is acceptable as the explicit truncation of the result always leads to the same loss of information.

Examples:

(uint16) (Hugo1_u32 + Hugo2_u32)	/* OK: Compliant */
(uint16) (Hugo1_s32 + Hugo2_s32)	/* Not OK: Different essential type category */
(uint16) Hugo_s32	/* OK: Hugo_s32 is not composite */
(uint32) (Hugo1_u16 + Hugo2_u16)	/* Not OK: Cast to wider essential type */

3.2 Rule Set: Compiler Abstraction / Portability

The motivation for compiler abstraction is portability. Source code must be compiler-independent because software modules are used on several µC platforms (BBM 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 → 16bit → 32bit) whereas downward portability seems not so important (32bit → 16bit → 8bit) because it is the more difficult form.

MCAL functions are always µC 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 µC or compiler. This has to be documented and such software modules shall not be provided for other µCs 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

DerivedFrom: MISRA C:2012 Rule 1.3

The source code of software modules (at least above the µC 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.

Any program that gives rise to undefined or unspecified behaviour may not behave in the expected manner. In many cases, the effect is to make the program non-portable but it is also possible for more serious problems to occur. For example, undefined behaviour might affect the result of a computation. If correct operation of the software is dependent on this computation then system safety might be compromised. The problem is particularly difficult to detect if the undefined behaviour only manifests itself on rare occasions.

3.2.2 Prohibition of Language and Compiler Specifics

Rule Abstr_Main_002: Prohibition of Compiler Specific Headers and Libraries

Instruction

DerivedFrom: MISRA C:2012 Dir 1.1

DerivedFrom: MISRA C:2012 Rule 1.3

DerivedFrom: MISRA C:2012 Rule 21.4

DerivedFrom: MISRA C:2012 Rule 21.5

DerivedFrom: MISRA C:2012 Rule 21.6

DerivedFrom: MISRA C:2012 Rule 21.7

DerivedFrom: MISRA C:2012 Rule 21.8

DerivedFrom: MISRA C:2012 Rule 21.9

DerivedFrom: MISRA C:2012 Rule 21.10

DerivedFrom: MISRA C:2012 Rule 21.11

DerivedFrom: MISRA C:2012 Rule 21.12

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.

Following points are considered with this rule because they lead to undefined and unspecified behaviour:

- ▶ The standard header file <setjmp.h>, <signal.h> shall not be used
- ▶ The Standard Library input/output functions shall not be used (provided by <stdio.h> and <wchar.h>)
- ▶ The 'atof', 'atoi', 'atol', 'atoll', 'abort', 'exit', 'getenv', 'system', 'bsearch' and 'qsort' functions of <stdlib.h> shall not be used
- ▶ The Standard Library time and date functions of <time.h> shall not be used
- ▶ The standard header file <tgmath.h> shall not be used
- ▶ The exception handling features of <fenv.h> should not be used

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: "_mfcr" 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

05 unit-specific and thus there is no general conclusion. Therefore the guideline goes a conservative way in disallowing the ANSI-C library because BSW software is delivered to different business units and it is not ensured that the needed headers of the ANSI-C library are really available there.

10 Rule Abstr_Main_003: Prohibition of Compiler Specific Keywords

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

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

20 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 [\[Abstr_Inline_001\]](#).

25 Also #warning is a compiler extension. Only #error is a preprocessing directive which is specified in the C standard. #error can be used but #warning shall be avoided.

25 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.

30 Examples (not to be used):

30 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
```

35 Hardware Configuration specific code:

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

40 #waring is not allowed because it is not specified in the C standard:

```
#if !defined Hugo
    #warning Hugo is not defined
#endif
```

50 Rule Abstr_Main_004: Prohibition of Redefinition of Reserved Identifiers, Macros and Functions

55 **Instruction**

DerivedFrom: MISRA C:2012 Rule 21.1

DerivedFrom: MISRA C:2012 Rule 21.2

55 #define and #undef shall not be used on a reserved identifier or reserved macro name. A reserved identifier or macro name shall not be declared.

60 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. Reserved identifiers and reserved macro names are intended for use by the implementation. Removing or changing the meaning of a reserved macro may result in undefined behaviour. The implementation is permitted to rely on reserved identifiers behaving as described in The Standard and may treat them specially.

05 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.

10 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.

15 **Rule Abstr_Main_005:** Prohibition of Restrict Type Qualifier

Instruction

20 **Scope:** C99 based code

25 **DerivedFrom:** MISRA C:2012 Rule 8.14

30 The restrict type qualifier shall not be used.

25 When used with care the C99 restrict type qualifier may improve the efficiency of code generated by a compiler. It may also allow improved static analysis. However, to use the restrict type qualifier the programmer must be sure that the memory areas operated on by two or more pointers do not overlap. There is a significant risk that a compiler will generate code that does not behave as expected if restrict is used incorrectly. Therefore the C99 restrict type qualifier is not allowed.

3.2.3 Abstraction of Addressing Keywords

35 For 16 bit controllers the address bus is 16 bit. With 16 bit only 64 kB can be directly 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|42_Release4.2R0001\01_Standard\AUTOSAR_SWS_CompilerAbstraction.pdf](http://Si8256/autosar$|SVN3-COPY|22-Releases|42_Release4.2R0001\01_Standard\AUTOSAR_SWS_CompilerAbstraction.pdf)].

40 Some words before:

45 16 bit microcontroller abstraction is mainly important for generated code. For manually written code there are some disadvantages:

- 50
- ▶ 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 manually coded 16 bit microcontroller abstraction might be released buggy (whereas generated code usually is with much less bugs).

55 But to be AUTOSAR compliant also for manually written 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

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

65 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.

70 Specific keywords shall be encapsulated via special headers.

Rule Abstr_NearFar_002: AUTOSAR 16 bit Microcontroller Abstraction

Instruction To encapsulate 16 bit microcontroller specifics 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 7 "Overview of Memory Classes"](#)
- ▶ ptrclass is defined in [Table 8 "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

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

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

15 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

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

20 The pointer itself is not modifiable (fix address).

25 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

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

35 The pointer itself is not modifiable (fix address).

40 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

45 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

50 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

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

Available memclasses are shown in [Table 7 "Overview of Memory Classes"](#).

Available ptrclasses are shown in [Table 8 "Overview of Pointer Classes"](#).

60 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 7 Overview of Memory Classes

Memory Type	Syntax of memory class (memclass) macro parameter	Comments
Variable	{MODULE}_VAR_{INIT_POLICY}[_{SAFETY}]_{{SIZE}} {MODULE}_VAR_FAST_{INIT_POLICY}[_{SAFETY}]_{{SIZE}} {MODULE}_VAR_SLOW_{INIT_POLICY}[_{SAFETY}]_{{SIZE}} {MODULE}_INTERNAL_VAR_{INIT_POLICY}[_{SAFETY}]_{{SIZE}} {MODULE}_VAR_SAVED_ZONE{X}[_{SAFETY}]_{{SIZE}}	memclass follows the same rules than for the memory allocation keyword for variables – see table Table 12 "Overview of Memory Allocation Keywords for Variables"
Constant	{MODULE}_CONST[_{SAFETY}]_{{SIZE}} {MODULE}_CONST_FAST[_{SAFETY}]_{{SIZE}} {MODULE}_CONST_SLOW[_{SAFETY}]_{{SIZE}} {MODULE}_CALIB[_{SAFETY}]_{{SIZE}} and so on	memclass follows the same rules than for the memory allocation keyword for constants – see table Table 13 "Overview of Memory Allocation Keywords for Constants"
Code	{MODULE}_CODE[_{SAFETY}] {MODULE}_CODE_FAST[_{SAFETY}] {MODULE}_CODE_SLOW[_{SAFETY}] {MODULE}_CODE_LIB[_{SAFETY}] and so on	memclass follows the same rules than for the memory allocation keyword for code – see table Table 14 "Overview of Memory Allocation Keywords for Code"

Table 8 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).

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.

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;
```

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 [\[Document AUTOSAR_SWS_MemoryMapping / URL: ||Si8256|autosar\\$|SVN3-COPY|22-Releases|42_Release4.2R0001|01_Standard|AUTOSAR_SWS_MemoryMapping.pdf\]](#).

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 4.2.1. The main concept is already there with AUTOSAR 4.0 Rev. 0. But the macro names and the initialization policy are based on AUTOSAR version 4.1 (or higher). The safety suffix is based on AUTOSAR version 4.2

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 [\[Document AUTOSAR V4.1 Rev. 1 module list / URL: \\Si8256\autosar\\$\SVN3-COPY\22_Releases\41_Release4.1R0001\02_Auxiliary\AUTOSAR_TR_BSWModuleList.pdf\]](#) in UPPERCASE letters (e.g. 'EEP' or 'CAN').
- ▶ For ASW the short name of the **SoftWare Component (SWC)** in case sensitive letters.

{NAME} is the keyword of the specific section (all keywords can be found in tables [12](#), [13](#) and [14](#)).

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 [\[Document AUTOSAR V4.1 Rev. 1 module list / URL: \\Si8256\autosar\\$\SVN3-COPY\22_Releases\41_Release4.1R0001\02_Auxiliary\AUTOSAR_TR_BSWModuleList.pdf\]](#)), {vi} is its vendorId and {ai} is its vendorApInfix. The sub part in square brackets [_{vi}_{ai}] is omitted if no vendorApInfix is defined for the BSW Module (BSW00300, BSW00347).

Hint RTA-RTE is generating also MemMap header files for BSW modules with SWCD ARXML files with the naming convention "{Mip}_MemMap.h". In that case the BSW MemMap header files are provided with the name "{Mip}-**Cfg**_MemMap.h to avoid naming conflicts.

05 ▶ For ASW "{SWC}_MemMap.h" has to be included.

10 Examples:

```
10  /* Memory location of a static 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"

 20  /* This example is only valid for code generators: */
  /* Definition of variables in CAN module (if it is not done with RTE): */
  #define CAN_START_SEC_VAR_CLEARED_32
  #include "Can_MemMap.h"
  VAR(uint32, CAN_VAR_CLEARED_32) CanMessage1;
  VAR(uint32, CAN_VAR_CLEARED_32) CanMessage2;
  #define CAN_STOP_SEC_VAR_CLEARED_32
  #include "Can_MemMap.h"

 25  /* Referencing these CAN variables within EEP (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"

 30  /* For referencing: The module name of the defining module has to be used. */
  /* For manually written code the header file concept shall be followed */

 35  /* Memory location of a function for BSW: */
  Definition of function in CAN module:
```

```
40  #define CAN_START_SEC_CODE_SLOW
  #include "Can_MemMap.h"
  FUNC(void, CAN_CODE_SLOW) Can_ExampleFunction(void);
  #define CAN_STOP_SEC_CODE_SLOW
  #include "Can_MemMap.h"

 45  Referencing this function in EEP module:
  #define CAN_START_SEC_CODE_SLOW
  #include "Can_MemMap.h"
  extern FUNC(void, CAN_CODE_SLOW) Can_ExampleFunction(void);
  #define CAN_STOP_SEC_CODE_SLOW
  #include "Can_MemMap.h"

 50  /* For referencing: The module name of the defining module has to be used. */

 55
```

60 The start and stop memory allocation keywords are configured with section identifiers in "{Mip}_MemMap.h" or "{SWC}_-MemMap.h".

65 Table 9 Overview of Meaning of {INIT_POLICY} Postfix

Keyword	Description
NO_INIT	AUTOSAR usage: Used for variables that are never cleared and never initialized. RB usage: Not to be used.

Keyword	Description
CLEARED	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.
POWER_ON_CLEARED	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.
INIT	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.
POWER_ON_INIT	AUTOSAR usage: Used for variables that are initialized with values only after power on reset. RB usage: Not to be used.

Table 10 Overview of Meaning of **{SAFETY}** Postfix

Keyword	Description
QM	Suffix is optional. It might be used if there are no safety integrity levels required.
ASIL_A	To be used for safety integrity level ASIL A.
ASIL_B	To be used for safety integrity level ASIL B.
ASIL_C	To be used for safety integrity level ASIL C.
ASIL_D	To be used for safety integrity level ASIL D.

The **{SAFETY} postfix is optional** (this is marked in this document with these brackets: "[]"). The **{SAFETY}** postfix indicates the maximum possible safety level. Downscaling in the project is possible inside memory mapping header files. If no **{SAFETY}** keyword is added the default shall be treated as QM (without any ASIL qualification).

Table 11 Overview of Meaning of **{SIZE}** Postfix

Keyword	Description
BOOLEAN	Used for variables and constants of size 1 bit.
8	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.
16	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.
32	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.
UNSPECIFIED	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 12 Overview of Memory Allocation Keywords for Variables

Type of variables	Syntax of memory allocation keywords	Comments
Normal variables	{MODULE}_START_SEC_VAR_{INIT_POLICY}_{SAFETY}_{SIZE} {MODULE}_STOP_SEC_VAR_{INIT_POLICY}_{SAFETY}_{SIZE}	To be used for global or static variables.
Variables with frequent usage	{MODULE}_START_SEC_VAR_FAST_{INIT_POLICY}_{SAFETY}_{SIZE} {MODULE}_STOP_SEC_VAR_FAST_{INIT_POLICY}_{SAFETY}_{SIZE}	To be used for all global or static variables that have at least one of the following properties: <ul style="list-style-type: none"> ▶ frequently used ▶ high number of accesses in source code ▶ accessed bitwise

Type of variables	Syntax of memory allocation keywords	Comments
Variables with rare usage	<code>{MODULE}_START_SEC_VAR_SLOW_{INIT_POLICY}_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_VAR_SLOW_{INIT_POLICY}_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Variables for calibration	<code>{MODULE}_START_SEC_INTERNAL_VAR_{INIT_POLICY}_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_INTERNAL_VAR_{INIT_POLICY}_{SAFETY}_{SIZE}</code>	To be used for global or static variables accessible from a calibration tool.
RAM buffers in NVRAM	<code>{MODULE}_START_SEC_VAR_SAVED_ZONE{X}_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_VAR_SAVED_ZONE{X}_{SAFETY}_{SIZE}</code>	To be used for RAM buffers of variables saved in non volatile memory. Description for {X} is missing in AUTOSAR.

Table 13 Overview of Memory Allocation Keywords for Constants

Type of constants	Syntax of memory allocation keywords	Comments
Normal constants	<code>{MODULE}_START_SEC_CONST_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONST_{SAFETY}_{SIZE}</code>	To be used for global or static constants.
Constants with frequent usage	<code>{MODULE}_START_SEC_CONST_FAST_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONST_FAST_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Constants with rare usage	<code>{MODULE}_START_SEC_CONST_SLOW_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONST_SLOW_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Constants for calibration	<code>{MODULE}_START_SEC_CALIB_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CALIB_{SAFETY}_{SIZE}</code>	To be used for calibration constants.
Constants for calibration with frequent usage	<code>{MODULE}_START_SEC_CALIB_FAST_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CALIB_FAST_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Constants for calibration with rare usage	<code>{MODULE}_START_SEC_CALIB_SLOW_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CALIB_SLOW_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Constants for cartography	<code>{MODULE}_START_SEC_CARTO_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CARTO_{SAFETY}_{SIZE}</code>	To be used for cartography constants.
Constants from link-time configuration	<code>{MODULE}_START_SEC_CONFIG_DATA_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONFIG_DATA_{SAFETY}_{SIZE}</code>	Constants with attributes that show that they reside in one segment for link-time module configuration.
Constants from link-time configuration with frequent usage	<code>{MODULE}_START_SEC_CONFIG_DATA_FAST_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONFIG_DATA_FAST_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.
Constants from link-time configuration with rare usage	<code>{MODULE}_START_SEC_CONFIG_DATA_SLOW_{SAFETY}_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONFIG_DATA_SLOW_{SAFETY}_{SIZE}</code>	This is an extension to AUTOSAR.

Type of constants	Syntax of memory allocation keywords	Comments
Constants from post--build configuration	<code>{MODULE}_START_SEC_CONFIG_DATA_POSTBUILD_HEA-DER[_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CONFIG_DATA_POSTBUILD_HEA-DER[_{SAFETY}]</code>	Header with pointers that point to the constants with attributes that show that they reside in one segment for post build module configuration.
Constants from post build configuration	<code>{MODULE}_START_SEC_CONFIG_DATA_POSTBUILD[_{SAFE-TY}]_{SIZE}</code> <code>{MODULE}_STOP_SEC_CONFIG_DATA_POSTBUILD[_{SAFE-TY}]_{SIZE}</code>	Constants with attributes that show that they reside in one segment for post--build module configuration.
ROM buffers in NVRAM	<code>{MODULE}_START_SEC_VAR_SAVED_RECOVERY_ZONE{X}[_-{SAFETY}]</code> <code>{MODULE}_STOP_SEC_VAR_SAVED_RECOVERY_ZONE{X}[_-{SAFETY}]</code>	To be used for ROM buffers of variables saved in non volatile memory. Description for {X} is missing in AUTOSAR.

Table 14 Overview of Memory Allocation Keywords for Code

Type of code	Syntax of memory allocation keywords	Comments
Normal code	<code>{MODULE}_START_SEC_CODE[_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CODE[_{SAFETY}]</code>	To be used for mapping code to application block, boot block, external flash etc.
Fast code	<code>{MODULE}_START_SEC_CODE_FAST[_{NUM}][_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CODE_FAST[_{NUM}][_{SAFETY}]</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. The optional suffix _{NUM} is not used (and seems to be removed in future AUTOSAR releases).
Slow code	<code>{MODULE}_START_SEC_CODE_SLOW[_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CODE_SLOW[_{SAFETY}]</code>	To be used for code that shall go into slow code memory segments.
Library code	<code>{MODULE}_START_SEC_CODE_LIB[_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CODE_LIB[_{SAFETY}]</code>	To be used for code that shall go into library segments for BSW module or Software Component.
Callout code	<code>{MODULE}_START_SEC_CALLOUT_CODE[_{SAFETY}]</code> <code>{MODULE}_STOP_SEC_CALLOUT_CODE[_{SAFETY}]</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 (rule 19.15 in MISRA-C:2004, dir 4.10 in MISRA-C:2012). 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_004: Exception of Memory Mapping Concept: Inline Functions

Instruction For inline functions a memory mapping is not needed therefore memory mapping concept shall not be used.

For inline functions an own memory mapping is not needed. The inline function is located in that memory section where the caller function is located.

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 static variables or constants must not be defined inside of a function body when memory mapping is used for those static variables or constants.

Hint The [\[Abstr_MemMap_005\]](#) violates MISRA (rule 8.7 in MISRA-C:2004, rule 8.9 in MISRA-C:2012). 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. The 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

DerivedFrom: MISRA C:2012 Rule 8.10

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

DerivedFrom: MISRA C:2012 Dir 4.2

DerivedFrom: MISRA C:2012 Dir 4.3

Assembler instructions inside C code shall not be used unless they are encapsulated and isolated for special cases. All usage of assembly language should be documented, too.

Generally assembly language code is implementation-defined and therefore not portable because it is µC dependent and each µC 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.

A documentation of the assembler code implementation should be done by commenting the assembler implementation and a description of the interface between the C and assembly language code.

3.2.7 Prohibition of Dynamic Memory

Rule Abstr_DynMem_001: Prohibition of Dynamic Memory Allocation

Instruction

DerivedFrom: MISRA C:2012 Dir 4.12

DerivedFrom: MISRA C:2012 Rule 21.3

DerivedFrom: MISRA C:2012 Rule 22.2

Dynamic memory allocation shall not be used.

This rule applies to all dynamic memory allocation packages including:

1. Those provided by The Standard Library (e.g. memory allocation and deallocation functions of <stdlib.h> like calloc, malloc, realloc and free)
2. Third-party packages

The Standard Library's dynamic memory allocation and deallocation routines can lead to undefined behaviour. Any other dynamic memory allocation system is likely to exhibit undefined behaviours that are similar to those of The Standard Library. The specification of third-party routines shall be checked to ensure that dynamic memory allocation is not being used inadvertently.

The identifiers calloc, malloc, realloc and free shall not be used and no macro with one of these names shall be expanded. Use of dynamic memory allocation and deallocation routines provided by The Standard Library can lead to undefined behaviour, for example:

- ▶ Memory that was not dynamically allocated is subsequently freed
- ▶ A pointer to freed memory is used in any way
- ▶ Accessing allocated memory before storing a value into it

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.

05 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.

10 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 deviation that is raised against this rule.

15 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.

20

3.2.8 Pointer Type Conversions

25 Pointer types can be classified as follows:

- 30 ▶ Pointer to object
- ▶ Pointer to function
- ▶ Pointer to incomplete
- ▶ Pointer to void
- ▶ A null pointer constant, i.e. the value 0, optionally cast to void *

35 The only conversions involving pointers that are permitted by the C standard are:

- 40 ▶ A conversion from a pointer type into void
- ▶ A conversion from a pointer type into an arithmetic type
- ▶ A conversion from an arithmetic type into a pointer type
- ▶ A conversion from one pointer type into another pointer type

45 Although permitted by the language constraints, conversion between pointers and any arithmetic types other than integer types is undefined. The following permitted pointer conversions do not require an explicit cast:

- 50 ▶ A conversion from a pointer type into _Bool (C99 only)
- ▶ A conversion from a null pointer constant into a pointer type
- ▶ A conversion from a pointer type into a compatible pointer type provided that the destination type has all the type qualifiers of the source type
- ▶ A conversion between a pointer to an object or incomplete type and void *, or qualified version thereof, provided that the destination type has all the type qualifiers of the source type

55 In C99, any implicit conversion that does not fall into this subset of pointer conversions violates a constraint (C99 Sections 6.5.4 and 6.5.16.1). In C90, any implicit conversion that does not fall into this subset of pointer conversions leads to undefined behaviour (C90 Sections 6.3.4 and 6.3.16.1). The conversion between pointer types and integer types is implementation-defined.

60 Rule Abstr_PtrConv_001: Prohibited Function Pointer Conversions

Instruction

65 **DerivedFrom:** MISRA C:2012 Rule 11.1

Conversions shall not be performed between a pointer to a function and any other type.

05 A pointer to a function shall only be converted into or from a pointer to a function with a compatible type. The conversion of a pointer to a function into or from any of:

- ▶ Pointer to object
- ▶ Pointer to incomplete
- ▶ void *

10 results in undefined behaviour.

15 If a function is called by means of a pointer whose type is not compatible with the called function, the behaviour is undefined. Conversion of a pointer to a function into a pointer to a function with a different type is permitted by The Standard. Conversion of an integer into a pointer to a function is also permitted by The Standard. However, both are prohibited by this rule in order to avoid the undefined behaviour that would result from calling a function using an incompatible pointer type.

20 Exceptions:

- ▶ A null pointer constant may be converted into a pointer to a function
- ▶ A pointer to a function may be converted into void
- ▶ A function type may be implicitly converted into a pointer to that function type

25 Note: exception 3 covers the implicit conversions described in C90 Section 6.2.2.1 and C99 Section 6.3.2.1. These conversions commonly occur when: A function is called directly, i.e. using a function identifier to denote the function to be called of or a function is assigned to a function pointer.

30 Rule Abstr_PtrConv_002: Prohibited Object Pointer Conversions

35 Instruction

DerivedFrom: MISRA C:2012 Rule 11.2

Conversions shall not be performed between a pointer to an incomplete type and any other type.

40 A pointer to an incomplete type shall not be converted into another type. A conversion shall not be made into a pointer to incomplete type. Although a pointer to void is also a pointer to an incomplete type, this rule does not apply to pointers to void as they are covered by [\[Abstr_PtrConv_007\]](#).

45 Conversion into or from a pointer to an incomplete type may result in a pointer that is not correctly aligned, resulting in undefined behaviour. Conversion of a pointer to an incomplete type into or from a floating type always results in undefined behaviour. Pointers to an incomplete type are sometimes used to hide the representation of an object. Converting a pointer to an incomplete type into a pointer to object would break this encapsulation.

50 Exceptions:

1. A null pointer constant may be converted into a pointer to an incomplete type
2. A pointer to an incomplete type may be converted into void

55 Rule Abstr_PtrConv_003: Prohibition of Casts between Pointer and Void and Arithmetic Types

Instruction

DerivedFrom: MISRA C:2012 Rule 11.6

60 A cast shall not be performed between pointer to void and an arithmetic type.

65 Conversion of an integer into a pointer to void may result in a pointer that is not correctly aligned, resulting in undefined behaviour. Conversion of a pointer to void into an integer may produce a value that cannot be represented in the chosen integer type resulting in undefined behaviour. Conversion between any non-integer arithmetic type and pointer to void is undefined.

05 Exception: An integer constant expression with value 0 may be cast into pointer to void.

Example:

```
void * ptr_v;
uint32 u_u32;

ptr_v = (void *)0x1234u; /* Not OK: implementation-defined */
ptr_v = (void *)1024.0f; /* Not OK: undefined */
u_u32 = (uint32)ptr_v; /* Not OK: implementation-defined */
```

15 Rule Abstr_PtrConv_004: Prohibition of Casts between Object Pointer Types

Instruction

20 **DerivedFrom:** MISRA C:2012 Rule 11.3

25 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.

30 This rule applies to the unqualified types that are pointed to by the pointers. Casting a pointer to object into a pointer to a different object may result in a pointer that is not correctly aligned, resulting in undefined behaviour. Even if conversion is known to produce a pointer that is correctly aligned, the behaviour may be undefined if that pointer is used to access an object.

35 Exception: It is permitted to convert a pointer to object type into a pointer to one of the object types char, sint8 or uint8. The Standard guarantees that pointers to these types can be used to access the individual bytes of an object.

40 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. */
```

45 Rule Abstr_PtrConv_005: Preservation of Pointer Qualifications

Instruction

50 **DerivedFrom:** MISRA C:2012 Rule 11.8

55 A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.

60 Any attempt to remove the qualification associated with the addressed type by using casting is a violation of the principle of type qualification.

65 Note: The qualification referred to here is not the same as any qualification that may be applied to the pointer itself.

70 Some of the problems that might arise if a qualifier is removed from the addressed object are:

- ▶ Removing a const qualifier might circumvent the read-only status of an object and result in it being modified
- ▶ Removing a const qualifier might result in an exception when the object is modified/written to
- ▶ Removing a volatile qualifier might result in accesses to the object being optimized away

Examples:

```
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_pv16 = 0; /* Pointer to volatile value */
```

```

05     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_pv16;  /* Not OK: volatile qualifier gets lost   */

```

Rule Abstr_PtrConv_006: Preservation of Conversion between a Pointer to Object and Integer Type

Instruction

DerivedFrom: MISRA C:2012 Rule 11.4

A conversion should not be performed between a pointer to object and an integer type.

A pointer should not be converted into an integer. An integer should not be converted into a pointer.

Conversion of an integer into a pointer to object may result in a pointer that is not correctly aligned, resulting in undefined behaviour. Conversion of a pointer to object into an integer may produce a value that cannot be represented in the chosen integer type resulting in undefined behaviour. Casting between a pointer and an integer type should be avoided where possible, but may be necessary when addressing memory mapped registers or other hardware specific features. If casting between integers and pointers is used, care should be taken to ensure that any pointers produced do not give rise to the undefined behaviour discussed under rule [\[Abstr_PtrConv_004\]](#).

Exception: A null pointer constant that has integer type may be converted into a pointer to object.

Examples:

```

uint8 *PORTA = (uint8 *)0x0002;      /* Not OK */
uint16 *ptr_u16;
uint32 addr_u32 = (uint32)p;        /* Not OK */
uint8 *ptr_u8 = (uint8 *)addr_u32;   /* Not OK */
boolean b = (boolean) ptr_u16;       /* Not OK */

```

Rule Abstr_PtrConv_007: Preservation of Conversion from Pointer to Void to Pointer to Object

Instruction

DerivedFrom: MISRA C:2012 Rule 11.5

A conversion should not be performed from pointer to void into pointer to object.

Conversion of a pointer to void into a pointer to object may result in a pointer that is not correctly aligned, resulting in undefined behaviour. It should be avoided where possible but may be necessary, for example when dealing with memory allocation functions. If conversion from a pointer to object into a pointer to void is used, care should be taken to ensure that any pointers produced do not give rise to the undefined behaviour discussed under rule [\[Abstr_PtrConv_004\]](#).

Exception: A null pointer constant that has type pointer to void may be converted into pointer to object.

Examples:

```

uint16 * p_u16;
float32 f_f32;
f = (float32) p_u16;      /* Not OK */
p = (uint16 *)f_f32;     /* Not OK */

```

Rule Abstr_PtrConv_008: Preservation of Pointer Qualifications

Instruction

DerivedFrom: MISRA C:2012 Rule 11.7

A cast shall not be performed between pointer to object and a non-integer arithmetic type.

05

For the purposes of this rule a non-integer arithmetic type means one of:

- ▶ Essentially Boolean
- ▶ Essentially character
- ▶ Essentially enum
- ▶ Essentially floating

10

Conversion of an essentially Boolean, essentially character or essentially enum type into a pointer to object may result in a pointer that is not correctly aligned, resulting in undefined behaviour. Conversion of a pointer to object into an essentially Boolean, essentially character or essentially enum type may produce a value that cannot be represented in the chosen integer type resulting in undefined behaviour. Conversion of a pointer to object into or from an essentially floating type results in undefined behaviour.

20

Examples:

```
uint32 *p_u32;
void *p_v;
uint16 *p_u16;

p_v = p_u32;          /* OK: Pointer to uint32 into pointer to void */
p_u16 = p_v;          /* Not OK */
p_v = (void *)p_u16;  /* OK:      */
p_u32 = (uint32 *)p_v; /* Not OK */
```

30

3.2.9 Pointer Arithmetic and Arrays

Rule Abstr_PtrArith_001: Allowed Form of Pointer Arithmetic

35

Instruction

DerivedFrom: MISRA C:2012 Rule 18.1

40

A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.

45

Creation of a pointer to one beyond the end of the array is well-defined by The Standard and is permitted by this rule. Dereferencing a pointer to one beyond the end of an array results in undefined behaviour and is forbidden by this rule. This rule applies to all forms of array indexing:

```
integer_expression + pointer_expression
pointer_expression + integer_expression
pointer_expression - integer_expression
pointer_expression += integer_expression
pointer_expression -= integer_expression
++pointer_expression
pointer_expression++
--pointer_expression
pointer_expression--
pointer_expression[integer_expression]
integer_expression[pointer_expression]
```

50

Note: a subarray is also an array.

55

Note: for purposes of pointer arithmetic, The Standard treats an object that is not a member of an array as if it were an array with a single element (C90 Section 6.3.6, C99 Section 6.5.6).

60

Although some compilers may be able to determine at compile time that an array boundary has been exceeded, no checks are generally made at run-time for invalid array subscripts. Using an invalid array subscript can lead to erroneous behaviour of the program. Run-time derived array subscript values are of the most concern since they cannot easily be checked by static analysis or manual review. Code of a defensive programming nature should, where possible and practicable, be provided to check such subscript values against valid ones and, if required, appropriate action be taken.

65

70

05 It is undefined behaviour if the result obtained from one of the above expressions is not a pointer to an element of the array pointed to by pointer_expression or an element one beyond the end of that array. See C90 Section 6.3.6 and C99 Section 6.5.6 for further information. Multi-dimensional arrays are "arrays of arrays". This rule does not allow pointer arithmetic that results in the pointer addressing a different subarray. Array subscripting over "internal" boundaries shall not be used, as such behaviour is undefined.

10

15 Examples:

```
void MyFunc(void)
{
    uint32 data = 0;
    uint32 b = 0;
    uint32 c[10] = {0};
    uint32 d[5][2] = {0};

    20     uint32 *p1 = &c[0];           /* OK: Points to first array element */
    uint32 *p2 = &c[10];           /* OK: Points to one beyond */
    uint32 *p3 = &c[11];           /* Not OK: Undefined, points to two beyond */

    data = *p2;                   /* Not OK: Undefined, dereference one beyond */
    p1++;
    /* OK */

    25     c[-1] = 0;               /* Not OK: Undefined, array bounds exceeded */
    data = c[10];               /* Not OK: Dereference of address one beyond */
    data = *(&data + 0);         /* OK: C treats data as an array of size 1 */
    d[3][1] = 0;               /* OK: Access inside array */
    data = d[2][3];             /* Not OK: Internal boundary exceeded */
    p1 = d[1];
    /* OK */

    30     data = p1[1];           /* OK: p1 addresses an array of size 2 */
}
```

35 Rule Abstr_PtrArith_002: Allowed Form of Pointer Subtraction

Instruction

40 **DerivedFrom:** MISRA C:2012 Rule 18.2

45 Subtraction between pointers shall only be applied to pointers that address elements of the same array.

This rule applies to expressions of the form:

pointer_expression_1 - pointer_expression_2

50 It is undefined behaviour if pointer_expression_1 and pointer_expression_2 do not point to elements of the same array or the element one beyond the end of that array.

55 Rule Abstr_PtrArith_003: Allowed Form of Comparison of Pointers

Instruction

DerivedFrom: MISRA C:2012 Rule 18.3

55 >, >=, <, <= 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.

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

65 Example:

```
void Func(void)
{
    65     uint32 Array1[10];
    uint32 Array2[10];
    uint32 *Ptr1 = Array1;
```

```

05      if ( p1 < a1 )          /* OK: Pointer points to the same array */
10      {
11      }
12      if ( p1 < a2 )          /* Not OK: Pointer points to different array */
13      {
14      }
15  }
```

Rule Abstr_PtrArith_004: Validity of Pointer Objects

Instruction

DerivedFrom: MISRA C:2012 Rule 18.6

The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist.

The address of an object might be copied by means of:

- ▶ Assignment
- ▶ Memory move or copying functions

The address of an object becomes indeterminate when the lifetime of that object expires. Any use of an indeterminate address results in undefined behaviour.

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

DerivedFrom: MISRA C:2012 Rule 18.5

Declarations should contain no more than two levels of pointer nesting.

No more than two pointer declarators should be applied consecutively to a type. Any `typedef-name` appearing in a declaration is treated as if it were replaced by the type that it denotes. The use of more than two levels of pointer nesting 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 */
```

```

05     uint8 ** Arg5[])
10             /* Not OK, because triple pointer to uint8      */
11             /* Arrays are converted to a pointer to the      */
12             /* initial element of the array                  */
13 {
14     uint8 * Ptr1;
15     uint8 ** Ptr2;
16     uint8 *** Ptr3;
17     uint8 * * const * const Ptr4;
18
19     uint8 ** Ptr5[10];
20     ...
21 }
```

Rule Abstr_PtrArith_006: Prohibition of flexible array members

Instruction

DerivedFrom: MISRA C:2012 Rule 18.7

Flexible array members shall not be declared.

Flexible array members are most likely to be used in conjunction with dynamic memory allocation which is banned by [Abstr_DynMem_001] p. 83. The presence of flexible array members modifies the behaviour of the sizeof operator in ways that might not be expected by a programmer. The assignment of a structure that contains a flexible array member to another structure of the same type may not behave in the expected manner as it copies only those elements up to but not including the start of the flexible array member.

Example:

```

struct
{
    uint16 len;
    uint32 data[];    /* Not OK: Flexible array member */
} MyStruct_st;
```

Rule Abstr_PtrArith_007: Prohibition of Variable-length Array Types

Instruction

DerivedFrom: MISRA C:2012 Rule 18.8

Variable-length array types shall not be used.

Variable-length array types are specified when the size of an array declared in a block or a function prototype is not an integer constant expression. They are typically implemented as a variable size object stored on the stack. Their use can therefore make it impossible to determine statically the amount of memory that must be reserved for a stack. If the size of a variable-length array is negative or zero, the behaviour is undefined. If a variable-length array is used in a context in which it is required to be compatible with another array type, possibly itself variable-length, then the size of the array types shall be identical. Further, all sizes shall evaluate to positive integers. If these requirements are not met, the behaviour is undefined. If a variable-length array type is used in the operand of a sizeof operator, under some circumstances it is unspecified whether the array size expression is evaluated or not.

Example:

```

void MyFunc(uint16 N)
{
    uint32 data[N];    /* Not OK: Variable-length array */
    ...
}
```

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, which 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 basic 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 Double Definitions

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:

05

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

10

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.

15

Rule Abstr_CPP_012: Don't let the Code Depend on the Type and Size of Enumeration Objects

20

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

25

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) */
```

30

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

35

Rule Abstr_CPP_013: Use a Cast or Separate Implementations to Copy Volatile Structs

40

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

45

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).

50

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++ */
```

55

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.

60

Rule Abstr_CPP_014: Don't use __STDC__

65

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

70

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

75

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

80

Rule Abstr_CPP_015: Encapsulate C code Conflicting with C++ and Provide C++ Substitution Code

85

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.

90

Example:

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

95

```
05
else
    /* Here C compliant code shall be provided */
#endif
```

05 **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.

10

3.2.11 Prohibition of Open Source Software

15 Rule Abstr_OSS_001: Prohibition of Open Source Software

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

20 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.

25 3.3 Rule Set: Types and Symbols

30 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"](#).

35 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).

40 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 11 "Rule Set: Central Elements"](#). Refer to that chapter to get more information about types and symbols in ARXML format.

45 3.3.1 Base Integer and Float Data Types

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

50 Rule CCode_Types_001: Platform Integer and Float Data Types

50 **Instruction**

55 **DerivedFrom:** MISRA C:2012 Dir 4.6

55 The 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.

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

60 An overview over all Platform integer and float data types is given in [Table 15](#).

60 Table 15 Overview Platform Integer and Float Data Types

Type	Range	Description
boolean	0...1 FALSE, TRUE	boolean unsigned integer

Type	Range	Description
uint8	0 ... 255 0x00 ... 0xFF	8 bit unsigned integer
sint8	-128 ... +127 0x80 ... 0x7F	8 bit signed integer
uint16	0 ... 65535 0x0000 ... 0xFFFF	16 bit unsigned integer
sint16	-32768 ... +32767 0x8000 ... 0x7FFF	16 bit signed integer
uint32	0 ... 4294967295 0x00000000 ... 0xFFFFFFFF	32 bit unsigned integer
sint32	-2147483648 ... +2147483647 0x80000000 ... 0x7FFFFFFF	32 bit signed integer
float32	1.17549435E-38 ... +3.40282347E38	Single precision (32 bit) floating point number (consider 2nd hint below)
float64	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 µCs 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 µC 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.

Hint Consider also the chapter "*Essential Type Model*" p. 58 which restricts the C type system.

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 (=, ==, !=).

Hint Consider also the chapter "*Essential Type Model*" p. 58 which restricts the C type system.

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 be 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 the 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

DerivedFrom: MISRA C:2012 Dir 4.6

The 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.

05 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".

10 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 16 Overview Optimized Data Types

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

30 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.

35 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.

40 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.

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

50 Rule CCode_Types_007: Handling of Optimized Integer Data Types

55 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.

60 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)

65 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 the 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 The 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 17 Overview Common Symbols

Symbol	Values	Description
CPU_TYPE	CPU_TYPE_8 CPU_TYPE_16 CPU_TYPE_32	indicates a 8 bit processor indicates a 16 bit processor indicates a 32 bit processor
CPU_BYTE_ORDER	HIGH_BYTE_FIRST LOW_BYTE_FIRST	indicates that within an uint16, the high byte is located before the low byte. 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 The following other common symbols may be used: E_OK, E_NOT_OK, STD_HIGH, STD_LOW, STD_ACTIVE, STD_IDLE, STD_ON, STD_OFF.

Table 18 Overview Other Common Symbols

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

DerivedFrom: MISRA C:2012 Rule 11.9

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.

The conformity to the MISRA C:2012 rule 11.9 is given because NULL_PTR is the only permitted form of a integer null pointer constant within the BSW development.

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 "PduldType" has to be used to define variables for unique identifiers for PDUs.

"PduldType" 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 Pduld is the count of PDUs minus 1 (256 PDUs --> range of Pdulds from 0 to 255), as defined by the corresponding type of operation within that module.

Variables of type "PduldType" 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 Pduld of the corresponding target module is being used for referencing.

Rule CCode_ComStackTypes_002: Handling of Types for PDU Identifiers

Instruction Variables of type "PduldType" 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 Pdulds 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 the 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 the following form. [Table 19](#) 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 19 Explanation of each Enumerator Values of Type "BufReq_ReturnType"

Symbol	Value	Description
BUFREQ_OK	0x00	Buffer request accomplished successful.
BUFREQ_E_NOT_OK	0x01	Buffer request not successful. Buffer cannot be accessed.
BUFREQ_E_BUSY	0x02	Temporarily no buffer available. It's up the requester to retry request for a certain time.
BUFREQ_E_OVFL	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 20](#) shows an overview of all valid ranges within this type.

Table 20 Overview of all Valid Ranges within Type "NotifResultType"

Range	Description
0x00 - 0x1E	General return codes. A detailed specification is listed in Table 21 . Please consider that there are small differences between AUTOSAR release R3.1 and R4.0 which are shown in table.
0x1F - 0x3C	Error notification codes specific for the communication system CAN. For a detailed definition please refer to the AUTOSAR specification of CAN TP [CANTP].
0x3D - 0x5A	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.
0x5B - 0x78	Error notification codes specific for the communication system FlexRay. For a detailed definition please refer to the AUTOSAR specification of FlexRay TP [FlexRayTP]
> 0x78	Currently values in this range are invalid. In future further return codes might be specified for other communication systems.

Table 21 Overview Valid General Return Codes for "NotifResultType"

Return code	Value	Description
NTFRSLT_OK	0x00	Action has been successfully finished: ► message sent out (in case of confirmation) ► message received (in case of indication)
NTFRSLT_E_NOT_OK	0x01	Error notification: ► message not successfully sent out (in case of confirmation) ► message not successfully received (in case of indication)
NTFRSLT_E_TIMEOUT_A	0x02	Error notification: ► timer N_Ar/N_As (according to ISO specification [ISONM]) has passed its time-out value N_Asmax/N_Armax. This value can be issued to service user on both the sender and receiver side.
NTFRSLT_E_TIMEOUT_BS	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.
NTFRSLT_E_TIMEOUT_CR	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.
NTFRSLT_E_WRONG_SN	0x05	Error notification: unexpected sequence number (PCI.SN) value received. This value can be issued to the service user on the receiver side only.
NTFRSLT_E_INVALID_FS	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.
NTFRSLT_E_UNEXP_PDU	0x07	Error notification: unexpected protocol data unit received. This value can be issued to the service user on both the sender and receiver side.
NTFRSLT_E_WFT_OVRN	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.
NTFRSLT_E_ABORT	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.
NTFRSLT_E_NO_BUFFER	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.
NTFRSLT_E_CANCELATION_OK	0x0B	Action has been successfully finished: Requested cancellation has been executed.
NTFRSLT_E_CANCELATION_NOT_OK	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.
NTFRSLT_PARAMETER_OK	0x0D	The parameter change request has been successfully executed.

Return code	Value	Description
NTFRSLT_E_PARAMETER_NOT_OK	0x0E	The request for the change of the parameter did not complete
NTFRSLT_E_RX_ON	0x0F	The parameter change request not executed successfully due to an ongoing reception
NTFRSLT_E_VALUE_NOT_OK	0x10	The parameter change request not executed successfully due to a wrong value
-	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

Class: NamingConvention

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 22](#) shows an overview of all valid ranges within this type.

Table 22 Overview of all valid ranges within type "BusTrcvErrorType"

Range	Description
0x00 - 0x1E	General return codes. A detailed specification is listed in Table 23
0x1F - 0x3C	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].
0x3D - 0x5A	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.

Range	Description
0x5B – 0x78	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 [-FTRCV].
> 0x78	Currently values in this range are invalid. In future it might be possible that further return codes are specified for other communication systems.

15
Table 23 Overview valid general return codes for "BusTrcvErrorType"

Return code	Value	Description
BUSTRCV_OK	0x00	There is no bus transceiver error seen by the driver, or the transceiver does not support the detection of bus errors.
BUSTRCV_E_ERROR	0x01	Bus transceiver detected an unclassified error.
-	0x02 -- 0x1E	Reserved values for future usage.

25

Rule CCode_ComStackTypes_009: Naming of Return Codes of a Bus Status Notification

30

Instruction

Class: NamingConvention

Return codes of a bus status notification shall be named as follows: BUSTRCV_E_<Communication System Abbreviation>_<Error Code Name>.

35
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

50
Instruction Type "TpDataStateType" shall be used to define variables to store the state of a Transport Protocol (TP) buffer.

"TpDataStateType" is defined in the following form. [Table 24](#) shows the explanation of each enumerator value.

55
Definition AR4.0.2:

```
typedef enum
{
    TP_DATACONF,
    TP_DATARETRY,
    TP_CONFPENDING,
    TP_NORETRY
} TpDataStateType;
```

65
Definition AR4.0.3:

```
typedef enum
{
```

```

05     TP_DATACONF,
06     TP_DATARETRY,
07     TP_CONF_PENDING
08 } TpDataStateType;

```

Table 24 Explanation of each enumerator values of type "TpDataStateType"

Element	Description
TP_DATACONF	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
TP_DATARETRY	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.
TP_CONF_PENDING	TP_CONF_PENDING indicates that the previously copied data shall remain in the TP.
TP_NORETRY	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 the following form:

```

30 typedef struct
31 {
32     TpDataStateType TpDataState;
33     PduLengthType TxTpDataCnt;
34 } 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:

```

55 typedef enum
56 {
57     STMIN,
58     BS
59 } TPPParameterType;

```

Definition AR4.0.3:

```

60 typedef enum
61 {
62     TP_STMIN,
63     TP_BS,

```

```
05      TP_BC  
06  } TPPParameterType;
```

Table 25 Explanation of each enumerator values of type "TPParameterType"

Element	Description
STMIN / TP_STMIN	Separation Time
BS / TP_BS	Block Size
TP_BC	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 26* 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 26 Overview of symbolic constants for maximum and minimum limits

Symbolic constant	Value	Definition	Description
<MODULE>_MAXUINT8	255 0xFF	(0xff)	Maximum value of standard data type "uint8"
<MODULE>_MINUINT8	0 0x00	(0x0)	Minimum value of standard data type "uint8"
<MODULE>_MAXSINT8	127 0x7F	(0x7f)	Maximum value of standard data type "sint8"
<MODULE>_MINSINT8	-128 0x80	(-(<MODULE>_MAXSINT8) - 1)	Minimum value of standard data type "sint8"
<MODULE>_MAXUINT16	65535 0xFFFF	(0xffff)	Maximum value of standard data type "uint16"
<MODULE>_MINUINT16	0 0x0000	(0x0)	Minimum value of standard data type "uint16"
<MODULE>_MAXSINT16	32767 0x7FFF	(0x7fff)	Maximum value of standard data type "sint16"
<MODULE>_MINSINT16	-32768 0x8000	(-(<MODULE>_MAXSINT16) - 1)	Minimum value of standard data type "sint16"
<MODULE>_MAXUINT32	4294967296 0xFFFFFFFF	(0xffffffffL)	Maximum value of standard data type "uint32"
<MODULE>_MINUINT32	0 0x00000000	(0x0uL)	Minimum value of standard data type "uint32"
<MODULE>_MAXSINT32	2147483647 0x7FFFFFFF	(0x7fffffffL)	Maximum value of standard data type "sint32"

Symbolic constant	Value	Definition	Description
<code><MODULE>_MINSINT32</code>	-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:

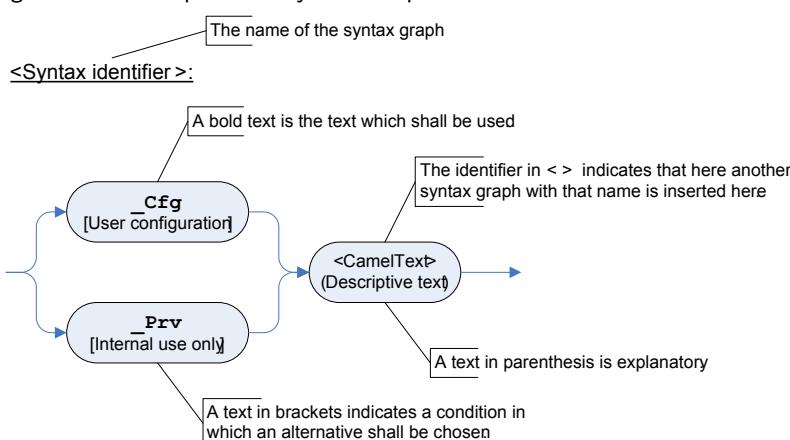
All other identifiers, e.g. automatic variables, typedefs, enum members

► ***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

05

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".

10

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".

15

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).

20

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 [\[CDGNaming_003\]](#).

25

Rule CDGNaming_001: Definition of a Component Prefix

Instruction

30

Class: NamingConvention

35

DerivedFrom: MISRA C:2012 Rule 5.8

Every name with a global visibility shall be prefixed with a component prefix.

40

One major goal of this naming convention is to avoid naming conflicts (internally and with SW used by customers of CDG). This is 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". This guarantees that identifiers that define objects or functions with external linkage are unique and helps to avoid confusion.

45

Building the component prefix follows one of the following criteria:

50

- ▶ CDG has an own name space represented by a name space prefix. The name space prefix for CDG is "rba_" and "RBA_". This name space prefix is used for all components which are not defined in a standard. An additional name follows this name space 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".

55

- ▶ 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 the name space prefix "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.

60

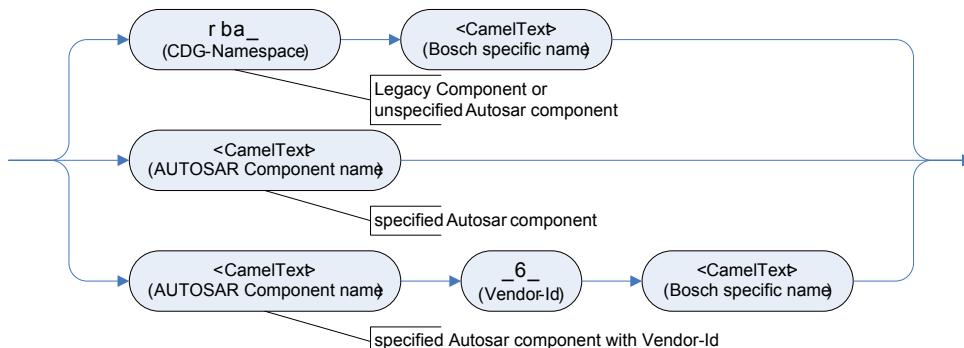
- ▶ For some cases AUTOSAR allows multiple implementations of drivers. This rule is related to BSW00347 in the AUTOSAR specification. An example is "Can". There are external CAN-devices. The drivers for such external devices can be implemented independently from the uC-internal Can-driver. So two or more Can-Driver are 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.

65

The following [Figure 3](#) shows the rule to create a component prefix in graphical form.

70

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

Class: NamingConvention

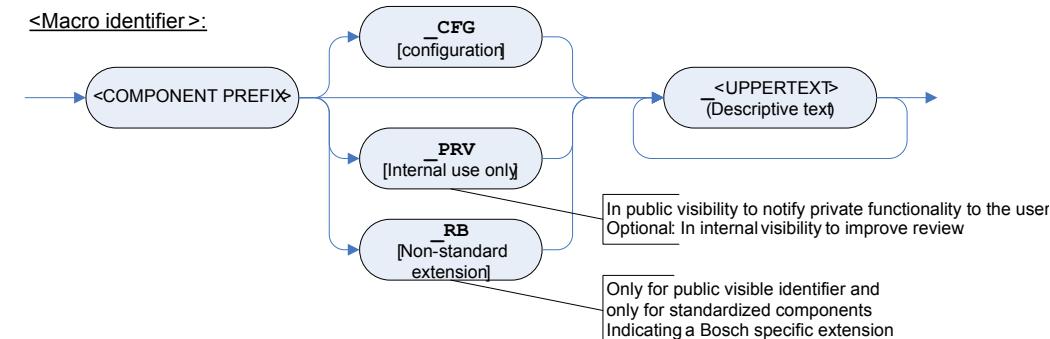
DerivedFrom: MISRA C:2012 Dir 4.5

Macro names shall be defined in the 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

Class: NamingConvention

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

- ▶ 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 symblicNameValue 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

Class: NamingConvention

DerivedFrom: MISRA C:2012 Dir 4.5

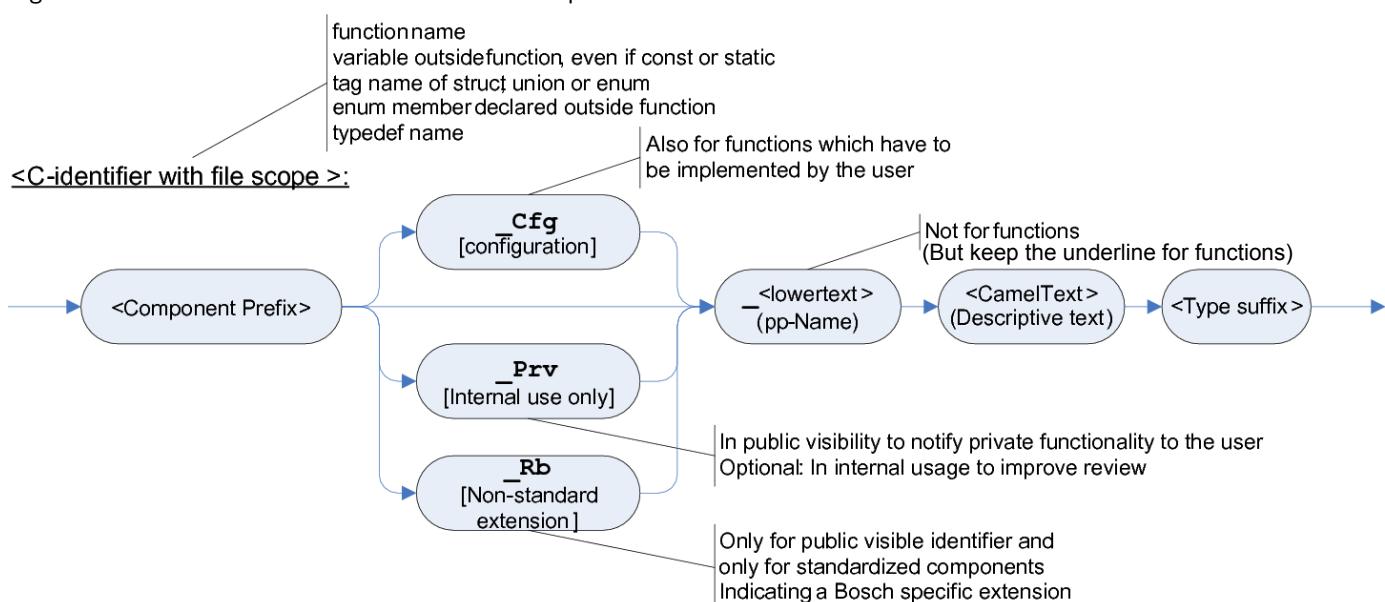
C-identifiers with file scope shall be defined in the following form: <Component prefix>{_identifier}<pp><Descriptive-Text>{_Typesuffix}.

This rule is relevant for the 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.

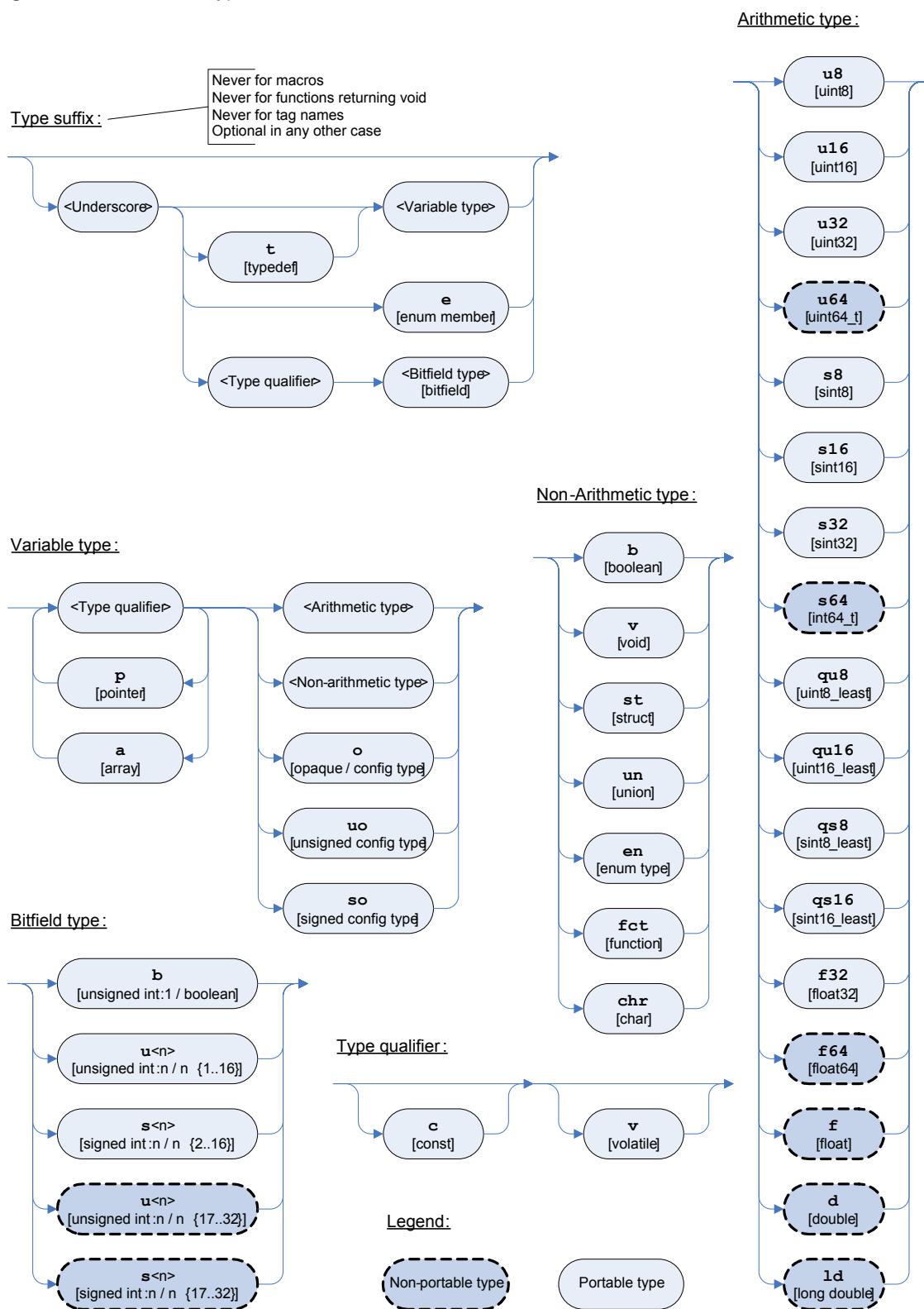
Figure 5 Creation of C-identifiers with File Scope



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.

05 The correct choice of type suffix depends on the change scenario. This is 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.

10 Examples:

- 15 ▶ 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);

20 ▶ 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.

- 25 ▶ An array with internal usage could have the following name: uint8 rba_PdmFs_Prv_dataMain_au8[10];
 In "Type suffix:" is chosen "<underscore>" and "<Variable type>" since these is a variable.

30 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.

- 35 ▶ 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).

40 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.

45 In "Non-arithmetic type:" is chosen "fct" since the dereferenced type is function. The concrete type of function shall not be considered. So the return type is not part of the resulting suffix "_pfct".

- 50 ▶ 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.

55 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.

60 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.

65 For the variable name is in second "Type qualifier:" chosen nothing since the array element, which is a pointer, is neither const nor volatile.

05 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.
 10 For the variable name we will get "_apcst" as suffix.

Rule CDGNaming_004: Typedefs as Unique Identifiers

Instruction

15 **Class:** NamingConvention

DerivedFrom: MISRA C:2012 Rule 5.6

20 A typedef name shall be a unique identifier.

25 A typedef name shall be unique across all name spaces and translation units. Multiple declarations of the same typedef name are only permitted by this rule if the type definition is made in a header file and that header file is included in multiple source files. Reusing a typedef name either as another typedef name or as the name of a function, object or enumeration constant, may lead to developer confusion.

Exception: The typedef name may be the same as the structure, union or enumeration tag name associated with the typedef.

30 Example:

```
typedef float32 mass;
void MyFunc(void)
{
    float32 mass = 0.0f;           /* Not OK: Reuse of typedef name */
}
```

Rule CDGNaming_005: Tag Names as Unique Identifiers

Instruction

40 **Class:** NamingConvention

DerivedFrom: MISRA C:2012 Rule 5.7

45 A tag name shall be a unique identifier.

50 The tag shall be unique across all name spaces and translation units. All declarations of the tag shall specify the same type. Multiple complete declarations of the same tag are only permitted by this rule if the tag is declared in a header file and that header file is included in multiple source files. Reusing a tag name may lead to developer confusion.

55 Example:

```
struct MyModule_xTag_tst
{
    uint16 stHugo_u16;
    uint16 stAnna_u16;
};

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

65 **Class:** NamingConvention

DerivedFrom: MISRA C:2012 Dir 4.5

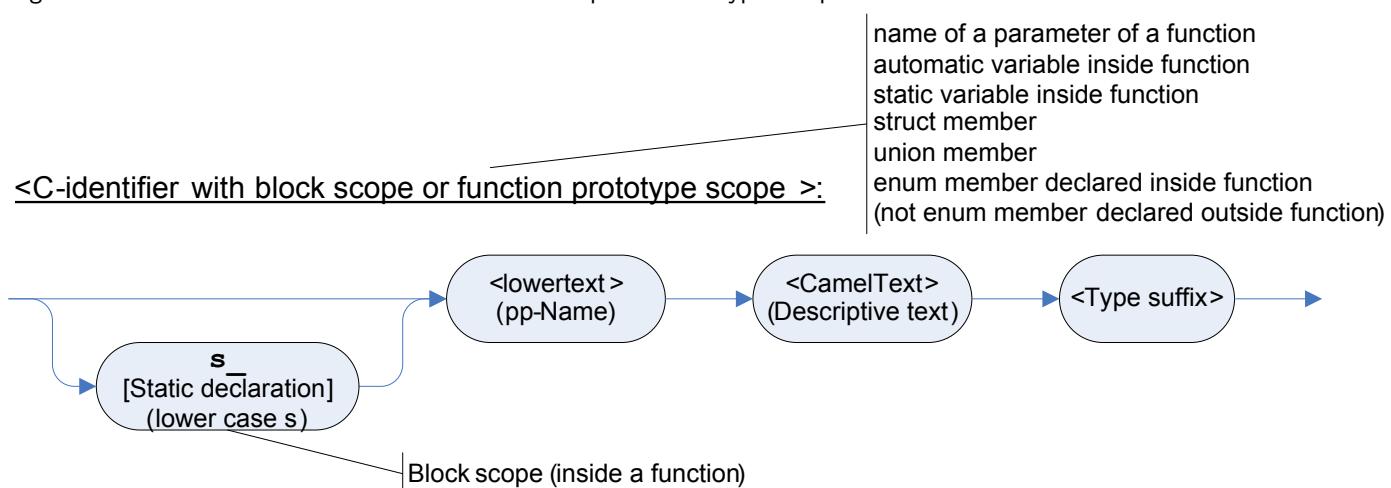
C-identifiers with block scope or function prototype scope shall be defined in the following form: {s_<pp><DescriptiveText>_<Typesuffix>.

This rule is relevant for the 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 of Function Identifier

Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 5.9

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. In case of static storage class specifier this rule helps that identifiers that define objects or functions with internal linkage should be unique.

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

Class: NamingConvention

DerivedFrom: MISRA C:2012 Dir 4.5

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.

Rule CDGNaming_009: Identifiers of Inner and Outer Scope

Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 5.3

An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.

An identifier declared in an inner scope shall be distinct from any identifier declared in an outer scope. The maximum number of non-significant characters is specified in rule [\[CDGNaming_010\]](#). If an identifier is declared in an inner scope but is not distinct from an identifier that already exists in an outer scope, then the inner-most declaration will "hide" the outer one. This may lead to developer confusion. But an identifier declared in one name space does not hide an identifier declared in a different name space. The terms outer and inner scope are defined as follows:

- ▶ Identifiers that have file scope can be considered as having the outermost scope
- ▶ Identifiers that have block scope have a more inner scope
- ▶ Successive, nested blocks, introduce more inner scopes

Example:

```
void MyFunc(void)
{
    uint16 i;          /* Declare an object "i" */
    {
        uint16 i;      /* Not OK: hides previous "i" */
        i = 3;         /* Could be confusing as to */
                      /* which "i" this refers */
    }
}
```

Rule CDGNaming_010: Significance of Identifiers

Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 5.1

DerivedFrom: MISRA C:2012 Rule 5.2

DerivedFrom: MISRA C:2012 Rule 5.4

DerivedFrom: MISRA C:2012 Rule 5.5

Identifiers (internal and external) shall not rely on the significance of more than 60 characters. Identifiers shall be distinct.

If two identifiers differ only in non-significant characters, the behaviour is undefined. 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. Long identifiers may impair the readability of code. While many automatic code generation systems produce long identifiers, there is a good argument for keeping identifier lengths well below this limit.

This rule shall apply across all name spaces:

- ▶ External identifiers shall be distinct
- ▶ Identifiers declared in the same scope and name space shall be distinct
- ▶ Macro identifiers shall be distinct
- ▶ Identifiers shall be distinct from macro names

Rule CDGNaming_011: File and Directory Names

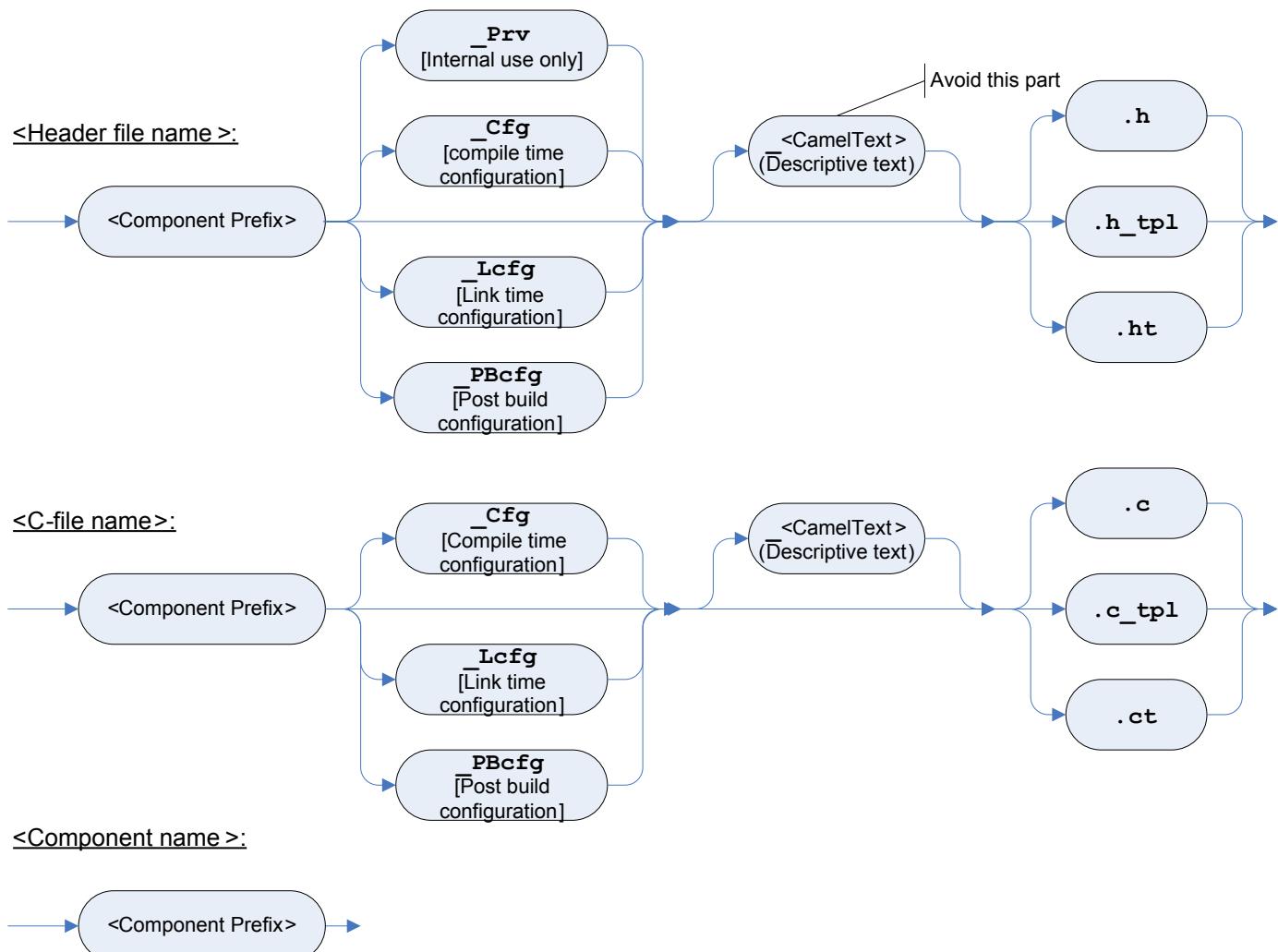
Instruction

Class: NamingConvention

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.

Rule CDGNaming_013: Usage of U Suffixes for Unsigned Integer Constants

Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 7.2

A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.

The type of an integer constant is a potential source of confusion, because it is dependent on a complex combination of factors including:

- ▶ The magnitude of the constant
- ▶ The implemented sizes of the integer types

05 ▶ The presence of any suffixes

▶ The number base in which the value is expressed (i.e. decimal, octal or hexadecimal)

10 For example, the integer constant 40000 is of type signed int in a 32-bit environment but of type signed long in a 16-bit environment. The value 0x8000 is of type unsigned int in a 16-bit environment, but of type signed int in a 32-bit environment.

15 Note:

- ▶ Any value with a "U" suffix is an unsigned type
- ▶ An unsuffixed decimal less than 2^{31} is of signed type

20 But:

- ▶ An unsuffixed hexadecimal value greater than or equal to 2^{15} may be of signed or unsigned type
- ▶ For C90, an unsuffixed decimal value greater than or equal to 2^{31} may be of signed or unsigned type

25 Signedness of constants should be explicit. If a constant is of an unsigned type, applying a "U" suffix makes it clear that the programmer understands that the constant is unsigned.

25 Note: This rule does not depend on the context in which a constant is used; promotion and other conversions that may be applied to the constant are not relevant in determining compliance with this rule.

30 Rule CDGNaming_014: Usage of L as Suffix for Constants

35 Instruction

Class: NamingConvention

DerivedFrom: MISRA C:2012 Rule 7.3

The lowercase character "l" shall not be used in a literal suffix.

40 Using the uppercase suffix "L" removes the potential ambiguity between "1" (digit 1) and "l" (letter "el") when declaring literals. The L (or LL) suffix has to be used for constants.

45 Example:

```
uint8 Val_u8 = 55u;          /* OK: No L suffix needed */
uint16 Val_u16 = 1234u;      /* OK: No L suffix needed */
uint32 Val_u32 = 0x3874214AuL; /* OK: L suffix is needed */
sint32 Val_s32 = 12345678l;  /* Not OK: No big L is used */
sint32 Val_s32 = 12345679L;  /* OK */
uint64 Val_u64 = 0x1234567812345678ull; /* Not OK: No big L is used */
uint64 Val_u64 = 0x1234567812345678uLL; /* OK */
```

50 Rule CDGNaming_015: Usage of f as Suffix for Float Constants

55 Instruction

Class: NamingConvention

Float constants of single precision shall be suffixed with a "f" character.

60 A suffix "f" classifies an float constant value as 32 bit single float value. 32 bit float calculations can (mostly) be done directly on hardware. If the f suffix is missing the float constant is interpreted as 64 bit float number. In this case calculations can only be done using compiler software libraries for emulation of double precision of float numbers. It is not guaranteed that such libraries are available. And if they are used the flash size and run time of the code is explicitly increased.

65 Example:

```

float val_f32;
val_f32 = val_f32 + 20.2;      /* Not OK: This addition is done in an 64 bit */
                               /* float context --> double promotion      */
                               /*                                         */
val_f32 = val_f32 + 20.2f;    /* OK: Single float precision used           */
                               /*                                         */

```

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 27](#) an overview of all possible files is given.

Table 27 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
<Module>_Cfg_<Sub>.c	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.
<Module>_Cfg.h *	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.
<Module>_Cfg_<Sub>.h	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.
<Module>_Lcfg.c *	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.
<Module>_Lcfg_<Sub>.c	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.
<Module>_Lcfg.h *	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.
<Module>_Lcfg_<Sub>.h		(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.
<Module>_PBcfg.c *	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.
<Module>_PBcfg_<Sub>.c	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.
<Module>_PBcfg.h *	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.
<Module>_PBcfg_<Sub>.h	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.
<Module>_Types.h *	0...1	Module specific types and symbols
<Module>_Prv.h	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).
<Module>_Prv_<Sub>.h	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
<code><Module>_Prv_<Sub>_Inl.h</code>	0...n	Additional private module header to separate inline functions from module header <code><Module>_Prv.h</code> for a better structure of the module. Is included in <code><Module>_Prv.h</code> 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 <code><Module>_Prv.h</code> header too. <code><Sub></code> 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 28](#) 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 28 Overview of ARXML Files

File Name	Incidence	Description
<code><Module>_BSWMD.arxml</code>	1	BSW Software Module Description
<code><Module>_SWCD.arxml</code>	0...1	Software Component Description
<code><Module>_EcucParamDef.arxml</code>	0...1	ECU Configuration Parameter Definition
<code><Module>_{<Sub>}EcucValues.arxml</code>	0...n	ECU Configuration Values
<code><Module>_Prot_EcucValues.arxml</code>	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 29](#) 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 30](#) 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 29 Overview Script Processor Based Files for Configuration

File Name	Incidence	Description
<code><Module>_{<Sub>}.mwe</code>	0...n	oAW Model Workflow Engine (replaces the .oaw file)
<code><Module>_{<Sub>}.chk</code>	0...n	oAW Check Script
<code><Module>_{<Sub>}.ext</code>	0...n	oAW Xtend Script
<code><Module>_{<Sub>}.xpt</code>	0...n	oAW Xpand Script
<code><Module>_{<Sub>}.pm</code>	0...n	Perl module
<code><Module>_{<Sub>}.ct</code>	0...n	c template for configuration processor
<code><Module>_{<Sub>}.ht</code>	0...n	h template for configuration processor
<code><Module>_{<Sub>}.xt</code>	0...n	ARXML template for configuration processor
<code><Module>_{<Sub>}.rt</code>	0...n	report template for configuration processor
<code><Module>_{<Sub>}.hxt</code>	0...n	hex template for configuration processor
<code><Module>.bamf</code>	0...1	Build Action Manifest (mandatory if script is called from BCT)

Table 30 Overview Non Script Processor Based Template Files for Configuration

File name	Incidence	Description
<Module>{<Sub>}.c_tpl	0...1	C template file for non script based configuration
<Module>{<Sub>}.h_tpl	0...1	Header template file for non script based configuration

In [Table 31](#) all other specific files are listed with their file extensions.

Table 31 Overview Other Specific Files

File Name	Incidence	Description
<Module>{<Sub>}.mcs	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.
<Module>{<Sub>}.inc	0...n	Include header for a .mcs file. This file is Infineon Tricore specific.
{<Module>}<Sub>.exe	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>_Cbk.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>_Cbk.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_Cbk.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 the 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 KTOFF  
#define TP_CFG_USE_NORMAL_FIXED_ADDRESSING KTOFF  
#define TP_CFG_USE_EXTENDED_ADDRESSING KTPON  
...  
  
in Tp.c:  
#include "Tp_Cfg.h"  
...  
#if (TP_CFG_USE_NORMAL_ADDRESSING == KTOFF)  
... 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.

05 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.

10 Rule BSW_Files_008: Private Module Header

15 **Instruction** Elements which shall not be exported via module header file "<Module>.h" shall be placed in private header "<Module>_Prv.h".

20 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.

25 3.5.2 Header Include Concept

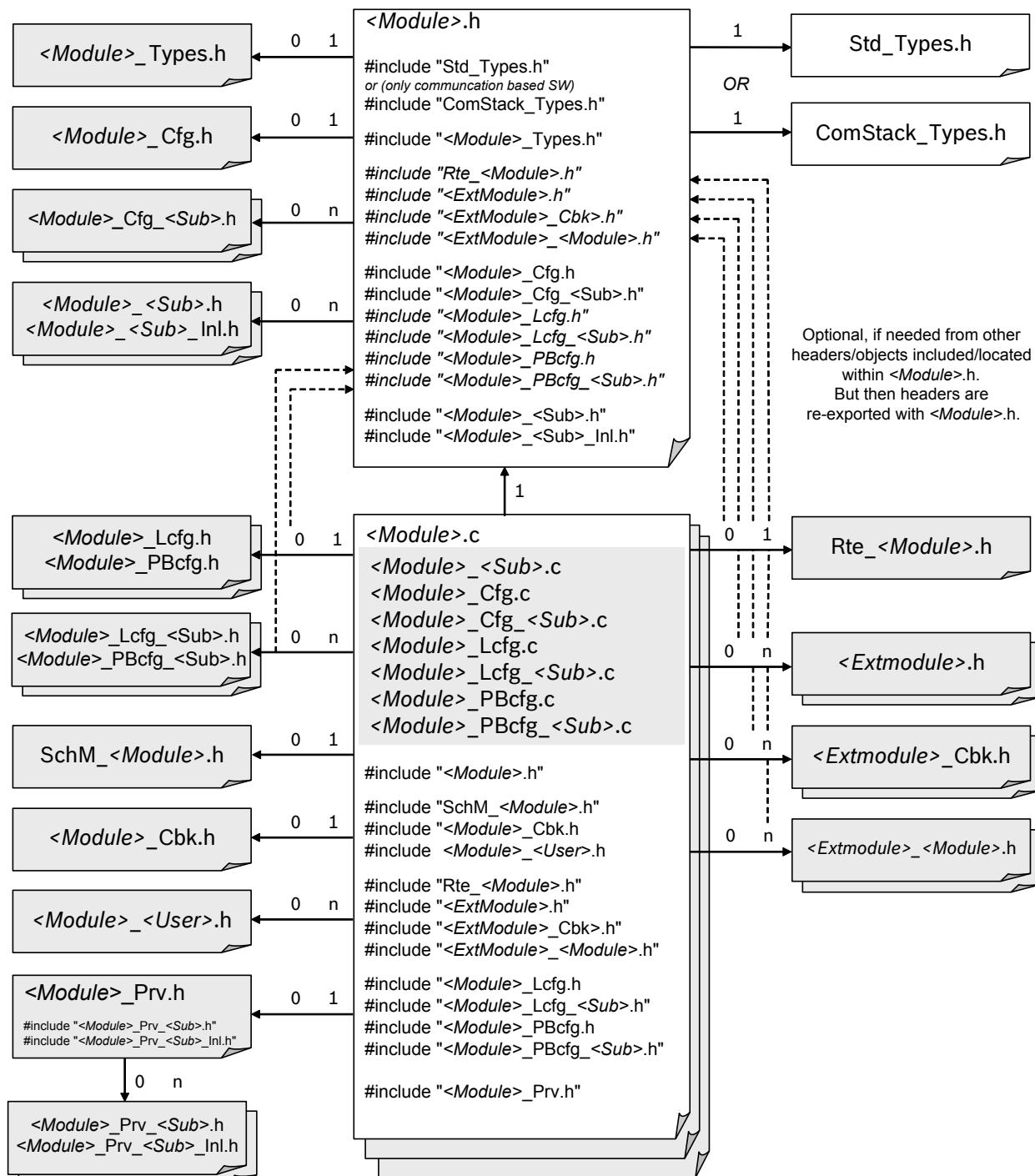
30 A module can contain a various set of header files. In addition headers from outside of the module are needed and have to be included. To provide a uniform visibility of header information a header include concept is needed. The following rules describe which headers have to be included in which files and in which order.

35 Rule BSW_HeaderInc_001: Definition of BSW Header Include Concept

40 **Instruction** The header include concept explained within the following rules shall be to be considered.

45 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>_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.

"SchM_<Module>.h" is the header with module specific functionalities provided by the BSW scheduler.

The module callback header "<Module>_Cbk.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>_Cbk.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.

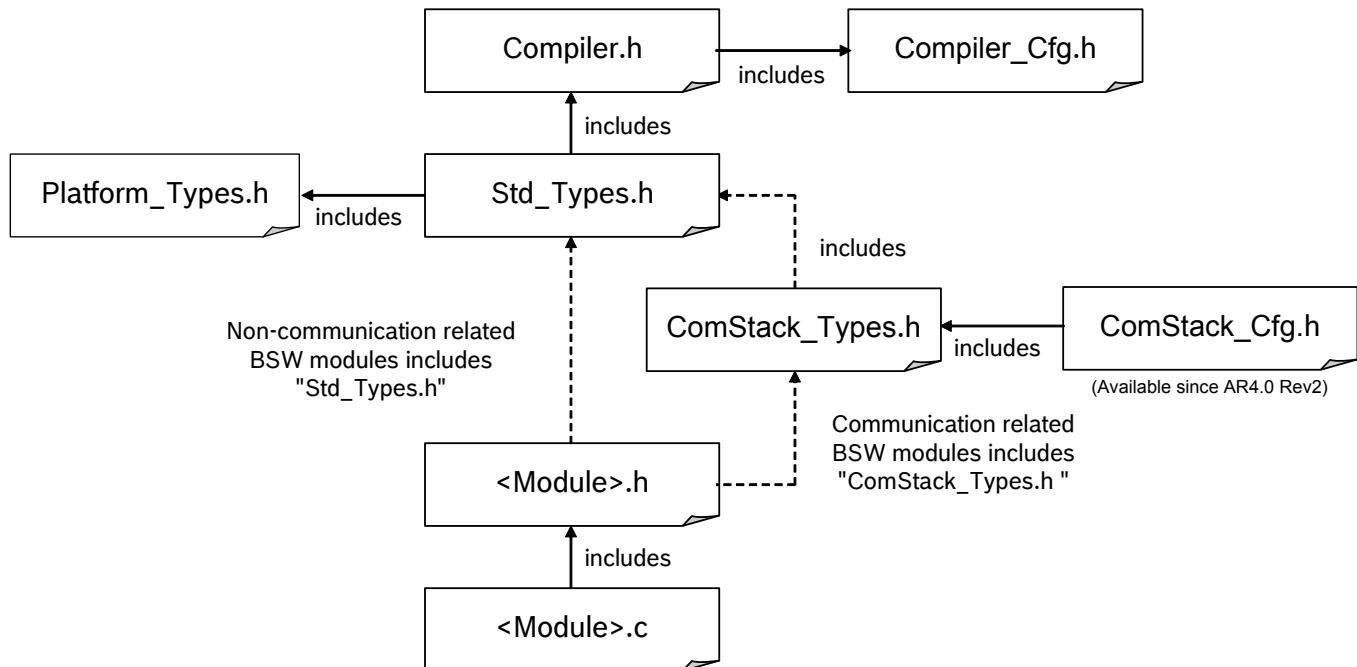
Rule BSW_HeaderInc_004: Include Order of Types Header inside Module Header

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 [\[BSW_HeaderInc_003\]](#) 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.

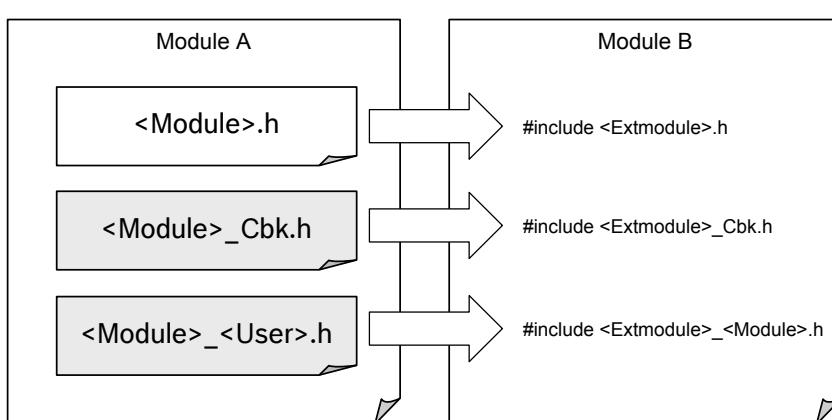
Declarations of callback functions are not a part of the module header, they have to be separated to <Module>_Cbk.h ([\[BSW_Files_003\]](#)). Every module that calls callback functions from other modules has to include the callback header from the other module. If no other APIs are needed an include 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

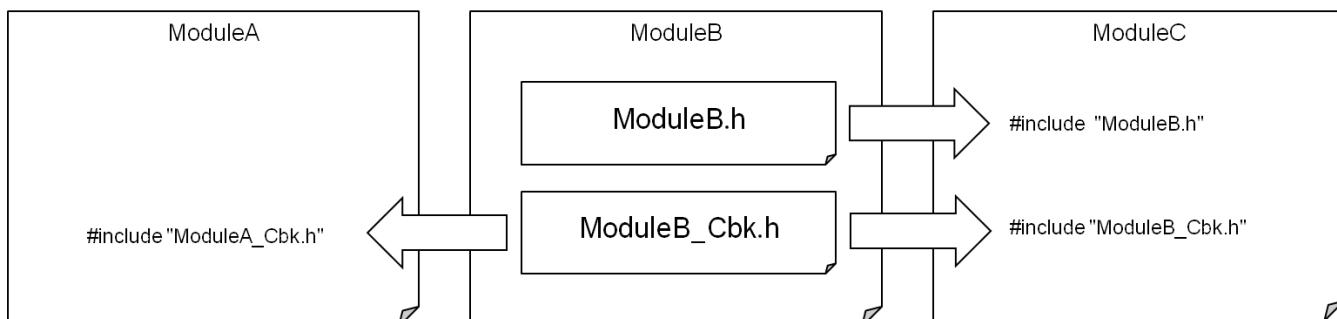


In the following two examples are given. There are three modules ModuleA, ModuleB and ModuleC. Interfaces from ModuleB are used from ModuleA and ModuleC. Two examples reflect the core of the rule exporting and importing only the relevant headers. Thus the two examples differ in usage and visibility of 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

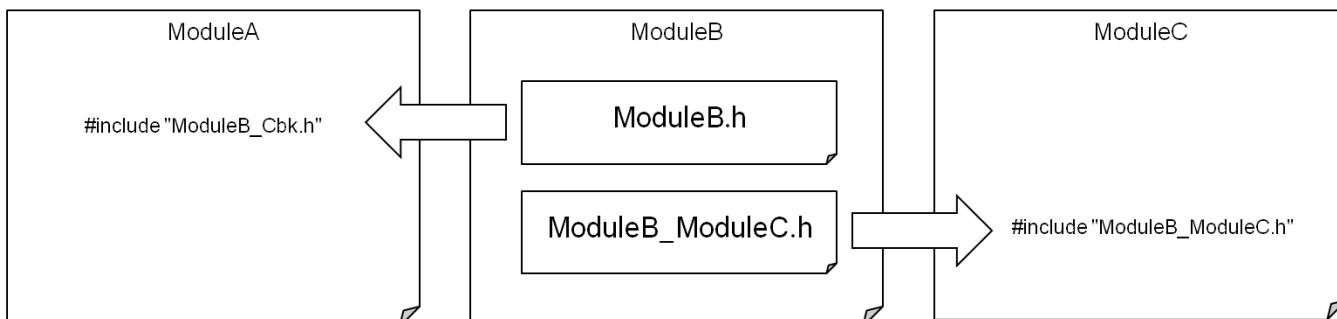
05
exported interfaces from ModuleB. Therefore ModuleC includes the common export header and the callback header from ModuleB.

Figure 12 Example 1



20
25
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

45
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>_Cbk.h" and/or "<Extmodule>_<Module>.h").

50
55
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>_Cbk.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

60
Scope: All headers except MemMap headers

DerivedFrom: MISRA C:2012 Rule 20.9

65
Precautions shall be taken in order to prevent the contents of a header file being included more than once.

05 When a translation unit contains a complex hierarchy of nested header files, it is possible for a particular header file to be included more than once. This can be, at best, a source of confusion. If this multiple inclusion leads to multiple or conflicting definitions, then this can result in undefined or erroneous behaviour.

10 The protection shall be done in the following form:

```
#ifndef HEADERNAME_H
#define HEADERNAME_H

...
/* HEADERNAME_H */
#endif
```

15 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.

20 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.

25

Rule BSW_HeaderInc_010: Preprocessor Check for AUTOSAR Headers

30 **Instruction** A pre-processor check shall be performed for all included AUTOSAR based header files (Inter Module Check).

35 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.

40 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 is based on stable information a check for valid versions is recommendable.

45 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
```

50

Rule BSW_HeaderInc_011: Correct Spelling of Headers within #include Directives

55 **Instruction** The name of a header within an #include directive shall be correctly spelled, i.e. mind upper and lower case. The name of an included header shall be syntactically identical to the file name of the header.

60 In some operating systems (typically systems based on Unix) file names are case-sensitive. For instance, if the file name of a header file starts with a capital letter, then the name of the included header shall start with a capital letter, too.

3.5.3 BSW Service Module with and without RTE

65 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

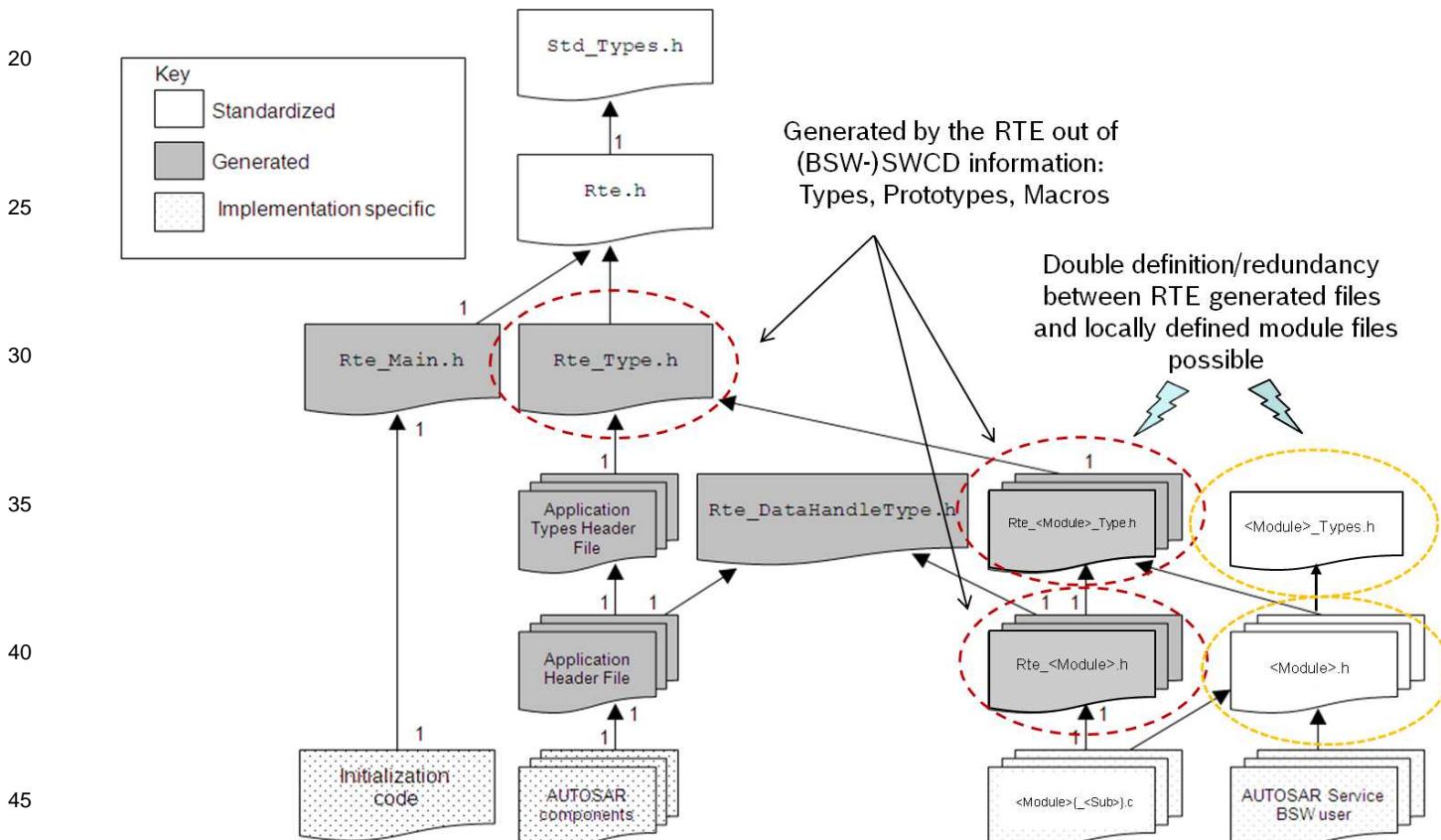
70

05 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 the following goals are reached:

- 10 ▶ Ensured avoidance of conflicts in project with a *RTE* generator caused of multiple declarations of types, macros, functions, ...
- 15 ▶ 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

20 The following *Figure 14* will illustrate the issue based on the relevant headers and the include order of the headers.

25 Figure 14 Overview Service Module and RTE



Rule BSW_ServiceRTE_001: BSW Service Module with and without RTE: Conditions for RTE headers

Instruction

55 **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 following *RTE* specific files shall be provided with the following content:

- 60 ▶ 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)
- 65 ▶ 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
- 70 ▶ The "*<Module>_SWCD.xml*" file shall be provided even if no *RTE* is available, but in case of an existing *RTE* generator this file is anyway mandatory

05 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.

10
15 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.

20
25 Hint It is obligatory that users of the mechanism described in this chapter document it in the product documentation (*_BswDoc.arxml file) in the integration hints section. 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:

30 "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."

30 Rule BSW_ServiceRTE_002: BSW Service Module with and without RTE: Conditions for Module Header

Instruction

35 **Scope:** BSW service modules which provide a SWCD file and have a close connection to the *RTE* generator

35 To support that a BSW service module can work with and without a *RTE* generator the module header / the module types header shall fulfill the following conditions:

- 40
- ▶ 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"
- 45

50 Example based on a NvM module:

NvM.h:

```
#include "Std_Types.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 */
```

65
...

Rte_NvM_Type.h: (Without RTE: This header is derived from a template file or is

```

05      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 */  */

10 Rte_Type.h: => Does not exist with systems without RTE

NvM.c:
#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 */          */

15 Nvm_SWCD.arxml: (Contains all relevant types and definitions. In case without RTE it
20      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: Return Type of Initialization Functions

Instruction

Status: removed

ReplacedBy: [\[BSW_ProcISR_001\]](#)

Rule BSW_APIDesign_002: Design of Main Processes

Instruction

Status: removed

ReplacedBy: [\[BSW_ProcISR_002\]](#)

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.
```

05 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.

10 ▶ 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).

15 Example:

```
Definition of an API: Can_ReturnType Can_Write(...)  
Return values: CAN_OK_E, CAN_E_BUSY_E, CAN_E_FAILED_E
```

```
15 Can_ReturnType is an enumeration type in can.h:  
16     typedef enum  
17     {  
18         CAN_OK_E = 0,  
19         CAN_E_BUSY_E,  
20         CAN_E_FAILED_E  
21     } Can_ReturnType;
```

25 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!

30 Rule BSW_APIDesign_009: Test of Error Information

35 Instruction

DerivedFrom: MISRA C:2012 Dir 4.7

If a function returns error information, then that error information shall be tested.

40 A function (whether it is part of The Standard Library, a third party library or a user defined function) may be deemed to 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.

45 Hint SW developer has to check that a function returns error information or not. A return value which represents no error information 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.

50 Rule BSW_APIDesign_010: Stable Number of Parameters

55 Instruction

DerivedFrom: MISRA C:2012 Rule 17.1

Functions shall not be defined with a variable number of arguments.

60 There are a lot of potential problems with such features. This also includes features of <stdarg.h> which shall not be used (va_list, va_arg, va_start, va_end, va_copy (only C99)).

65 Rule BSW_APIDesign_011: Prohibition of Recursive Function Calls

70 Instruction

DerivedFrom: MISRA C:2012 Rule 17.2

Functions shall not call themselves, either directly or indirectly. Recursive function calls are not allowed.

05 Recursion carries with it the danger of exceeding available stack space, which can lead to a serious failure. Unless recursion is very tightly controlled, it is not possible to determine before execution what the worst-case stack usage could be.

10 Rule BSW_APIDesign_012: Number of Arguments identical to Number of Parameters

Instruction

15 **DerivedFrom:** MISRA C:2012 Rule 17.3

The number of arguments passed to a function shall match the number of parameters.

20 Provided that a function call is made in the presence of a prototype, a constraint ensures that the number of arguments matches the number of parameters and that each argument can be assigned to its corresponding parameter. 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.

25 Rule BSW_APIDesign_013: Explicit Return Statement

Instruction

30 **DerivedFrom:** MISRA C:2012 Rule 17.4

All exit paths from a function with non-void return type shall have an explicit return statement with an expression.

35 The expression given to the return statement provides the value that the function returns. If a non-void function does not return a value but the calling function uses the returned value, the behaviour is undefined. This can be avoided by ensuring that, in a non-void function:

- ▶ Every return statement has an expression, and
- ▶ Control cannot reach the end of the function without encountering a return statement

40 Note: C99 constrains every return statement in a non-void function to return a value.

Examples:

```
45 uint32 MyModule_Func(uint16 Arg1_u16, uint16 Arg2_u16)
{
    uint32 result_u32;

    result_u32 = (uint32)Arg1_u16 + Arg2_u16;

    return (result_u32); /* OK: It is a failure if this line is not */
                         /* existing or there is only a 'return;' */
```

55 Rule BSW_APIDesign_019: Appropriate Number of Array Elements in Function Arguments

Instruction

60 **DerivedFrom:** MISRA C:2012 Rule 17.5

The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.

65 If a parameter is declared as an array with a specified size, the corresponding argument in each function call should point into an object that has at least as many elements as the array. The use of an array declarator for a function parameter specifies the function interface more clearly than using a pointer. The minimum number of elements expected by the function is explicitly stated, whereas this is not possible with a pointer. A function parameter array declarator which does

05 not specify a size is assumed to indicate that the function can handle an array of any size. In such cases, it is expected
 10 that the array size will be communicated by some other means, for example by being passed as another parameter, or
 by terminating the array with a sentinel value. The use of an array bound is recommended as it allows out-of-bounds
 15 checking to be implemented within the function body and extra checks on parameter passing. It is legal in C to pass an
 array of the incorrect size to a parameter with a specified size, which can lead to unexpected behaviour.

Examples:

```
/* Intent is that MyFunc1 does not access outside the range array1[ 0 ] .. array1[ 3 ] */
void MyFunc1(uint32 array1[4]);
```

```
/* Intent is that function handles arrays of any size */
void MyFunc2(uint32 array2[]);
```

```
void MyFunc(uint32_t *ptr)
{
    uint32 arr3[3] = {1, 2, 3};
    uint32 arr4[4] = {0, 1, 2, 3};

    MyFunc1(arr4);      /* OK: Size of array matches the prototype          */
    MyFunc1(arr3);      /* Not OK: Size of array does not match prototype */
    MyFunc1(ptr);       /* OK: Only if ptr points to at least 4 elements */
    MyFunc2(arr4);      /* OK */
    MyFunc2(ptr);       /* OK */
}
```

35 Rule BSW_APIDesign_020: No static Keyword between [] of an Array Parameter

Instruction

40 **Scope:** C99 based code

DerivedFrom: MISRA C:2012 Rule 17.6

The declaration of an array parameter shall not contain the static keyword between the [].

45 This is a C99 specific rule which bases on a mandatory MISRA C:2012 rule.

50 The C99 language standard provides a mechanism for the programmer to inform the compiler that an array parameter
 contains a specified minimum number of elements. Some compilers are able to take advantage of this information to
 generate more efficient code for some types of processor. If the guarantee made by the programmer is not honored, and
 the number of elements is less than the minimum specified, the behaviour is undefined. The processors used in typical
 55 embedded applications are unlikely to provide the facilities required to take advantage of the additional information
 provided by the programmer. The risk of the program failing to meet the guaranteed minimum number of elements
 outweighs any potential performance increase.

Example:

```
void MyFunc1(uint16 n, uint16 arr[static 20]);      /* Not OK: Static between [] */
```

60 Rule BSW_APIDesign_021: Usage of Non-Void Return Types

Instruction

65 **DerivedFrom:** MISRA C:2012 Rule 17.7

The value returned by a function having non-void return type shall be used.

05 It is possible to call a function without using the return value, which may be an error. If the return value of a function is intended not to be used explicitly, it should be cast to the void type. This is a possible exception of this rule and uses the same principle like for unused parameters (see [\[BSW_APIDesign_016\]](#)).

10 Example:

```
15 uint16 MyFunc1(uint16 para1)
{   return para1;
}

void MyFunc2(uint16 para2)
{
    MyFunc1(para2);      /* Not OK: Return value discarded */
    (void)MyFunc1(para2); /* OK: By exception */
    x = MyFunc1(para2); /* OK */
    ...
}
```

20 Rule BSW_APIDesign_022: No Modification of Function Parameters

25 Instruction

30 **DerivedFrom:** MISRA C:2012 Rule 17.8

35 A function parameter should not be modified.

40 A function parameter behaves in the same manner as an object that has automatic storage duration. While the C language permits parameters to be modified, such use can be confusing and conflict with programmer expectations. It may be less confusing to copy the parameter to an automatic object and modify that copy. With a modern compiler, this will not usually result in any storage or execution time penalty. Programmers who are unfamiliar with C, but who are used to other languages, may modify a parameter believing that the effects of the modification will be felt in the calling function.

45 Example:

```
50 uint16 MyFunc_Glob = 0;

void MyFunc1(uint16 para)
{
    para = MyFunc_Glob; /* Not OK: parameter is modified */
    ...
}

void MyFunc2(uint8 *p, uint8 *q)
{
    p = q;             /* Not OK: parameter is modified */
    *p = *q;           /* OK: parameter is not modified */
}
```

55 It is a good practice to mark constant pointer parameters with a const qualifier. There can be one or two const qualifier being used dependent if the pointer itself or the value the pointer points to is constant. [Table 32](#) shows the four possibilities.

60 Table 32 Overview of the Data Pointer Definition

Case	Example
Variable pointer to variable data	uint8 * ptrArg_pu8
Constant pointer to variable data	uint8 * const ptrArg_cpu8
Variable pointer to constant data	uint8 const * ptrArg_pcu8 or const uint8 * ptrArg_pcu8
Constant pointer to constant data	uint8 const * const ptrArg_cpcu8 or const uint8 * const ptrArg_cpcu8

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;
    ...

    ExtModule_Func;           /* Not OK: Parameter list is missing          */

30     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.        */
}

35     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;

60    /* 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

DerivedFrom: MISRA C:2012 Rule 2.7

Unused parameters shall be handled with a void cast.

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. Also MISRA C:2012 does not allow unused parameters. To avoid such warnings the corresponding unused parameter shall be handled with a void cast as shown in the example below. With this method the parameter is implicitly used and any compiler and MISRA related warnings are suppressed. There is no need to use special macros (like PARAM_UNUSED) to treat unused parameters. The void cast is a simple and immediate method of handling unused parameters.

Example:

```
void MyApi(uint32 Value1, uint32 Value2)
{
    uint32 var_u32 = Value1;

    /* "Value2" is defined because of compatibility aspects but is not used. */
    /* The 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 a changed interpretation of interface parameters shall always lead to a definition of a new interface.

It is possible that changes in software result 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.

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.

Rule BSW_ProcISR_007: No Dem/FIM Calls Within Interrupt Service Routines

Instruction Interrupt Service Routines shall not call Dem (Diagnostic Event Manager) or FIM (Function Inhibition Manager) services.

In general it should be avoided to call services within an Interrupt Service Routine.

The reasons for this rule are timing and priority aspects.

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 ([\[Abstr_Inline_001\]](#)).

Rule BSW_FuncObj_002: Explicit Functions and Objects

Instruction

DerivedFrom: MISRA C:2012 Rule 8.1

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 */
```

05
Rule BSW_FuncObj_003: Void Functions10
Instruction

DerivedFrom: MISRA C:2012 Rule 8.2

Function with no parameters shall be declared and defined with the parameter list void.

Example for a function with no parameter and no return value:

```
15    void MyFunc(void)
```

20
Rule BSW_FuncObj_004: Number of Definitions of Objects and Functions25
Instruction

DerivedFrom: MISRA C:2012 Rule 8.6

An identifier (object or function) with external linkage or global visibility shall have exactly one external definition in one file.

25
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 by this rule even if the definitions are the same. It is undefined behaviour if the declarations are different, or initialize the identifier to different values.

30
Rule BSW_FuncObj_005: Scope of Functions

Instruction Functions shall be declared at file scope and not on block scope.

35
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.

40
Rule BSW_FuncObj_006: Prototype Declarations45
Instruction

DerivedFrom: MISRA C:2012 Rule 8.2

DerivedFrom: MISRA C:2012 Rule 8.4

DerivedFrom: MISRA C:2012 Rule 8.5

DerivedFrom: MISRA C:2012 Rule 17.3

50
55
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.

60
65
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. This rule ensures the consistency between the declaration and the definition and declarations in different translation units.

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

05 file). The module header file is included in the `<Module>.c` file where the related function is implemented and a user has
 10 to include this header too, to be able to use the function in his module. Module global internal functions shall define
 15 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)
{
  ...
}
```

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;
  ...

  Hugo_Func(val1, val2);
  ...
}
```

50 **Hint** This rule cannot be maintained if a header file generation concept is used where specific header files are
 generated exactly focused on the demand of a module. Here the same external prototypes and identifiers are generated
 55 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

DerivedFrom: MISRA C:2012 Rule 8.2

DerivedFrom: MISRA C:2012 Rule 8.3

60 For each function parameter the type given in the declaration and definition shall be identical, and the return types shall
 65 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.

05 The types of the parameters and return values in the prototype and the definition shall match. Using types and qualifiers consistently across declarations of the same object or function encourages stronger typing. 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. Specifying parameter names in function prototypes allows the function definition to be checked for interface consistency with its declarations.

10
15
20
25

```

<Module>.h:
/* External prototype declaration: *****/
extern void Module_Func(uint16 param1, uint16 param2);
...
<Module>.c:
/* Includes *****/
#include "<Module>.h"
...
/* Implementation *****/
void Module_Func(uint16 param1, uint16 param2)
{
    ...
}

    \ Return type, data types
    / and names of parameters
    / have to be identical.
    /

```

An empty parameter list is not valid in a prototype. If a function type has no parameters its prototype form uses the keyword `void`, see rule [\[BSW_FuncObj_003\] p. 146](#).

30 Rule BSW_FuncObj_008: Static Objects and Functions

Instruction

35 **DerivedFrom:** MISRA C:2012 Rule 8.8

The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.

40 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.

45 The Standard states that if an object or function is declared with the *extern* storage class specifier and another declaration of the object or function is already visible, the linkage is that specified by the earlier declaration. This can be confusing because it might be expected that the *extern* storage class specifier creates external linkage. The static storage class specifier shall therefore be consistently applied to objects and functions with internal linkage.

50 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 [\[BSW_FuncObj_007\]](#).

55 **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.

60 Rule BSW_FuncObj_009: Constant Pointer Parameters

Instruction

65 **DerivedFrom:** MISRA C:2012 Rule 8.13

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.

70 A pointer should point to a `const-qualified` type unless either:

- 05 ▶ It is used to modify an object, or
 ▶ It is copied to another pointer that points to a type that is not const-qualified by means of either:
 – Assignment, or
 – Memory move or copying functions

10 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. This rule encourages best practice by ensuring that pointers are not inadvertently used to modify objects.

15 Example:

```
void MyModule_Func(uint16 * Param1_pu16, const uint16 * Param2_pcu16, uint16 * Param3_pu16)
    /* Param1_pu16: Addresses an object which is modified - no const           */
    /* Param2_pcu16: Addresses an object which is not modified - const required */
    /* Param3_pu16: Addresses an object which is not modified - const missing  */

{
    *Param1_pu16 = *Param2_pcu16 + *Param3_pu16;

/* Data at address Param3_pu16 has not been changed           */
/* -> This parameter should be declared as pointer to const data */
```

25 Rule BSW_FuncObj_010: Compatible Types of multiple declared Objects of Functions

30 **Instruction** If objects or functions are declared more than once their types shall be compatible.

35 The definition of compatible types is lengthy and complex. Two identical types are compatible but two compatible types need not be identical.

40 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.                           */

    /* ... */
}
```

55 Rule BSW_FuncObj_011: Global Constants

60 **Instruction** Global visible constant data shall be indicated with read-only purposes by explicitly assigning the const keyword.

65 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.

05 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 ([\[BSW_FuncObj_006\]](#)).

10 Example (simplified, without memory mapping concept):

```
const uint8 MaxPayload_cu8 = 0x18;
```

15 Rule BSW_FuncObj_012: Static Objects and Functions

Instruction

15 **DerivedFrom:** MISRA C:2012 Rule 8.7

19 Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.

20 Restricting the visibility of an object by giving it internal linkage or no linkage reduces the chance that it might be accessed inadvertently. Similarly, reducing the visibility of a function by giving it internal linkage reduces the chance of it being called inadvertently. Compliance with this rule also avoids any possibility of confusion between an identifier and an identical identifier in another translation unit or a library.

25 Rule BSW_FuncObj_013: Global Constants

Instruction

30 **DerivedFrom:** MISRA C:2012 Rule 7.4

34 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char".

35 Any attempt to modify a string literal results in undefined behaviour. For example, some implementations may store string literals in read-only memory in which case an attempt to modify the string literal will fail and may also result in an exception or crash. This rule, when applied in conjunction with others, prevents a string literal from being modified. It is explicitly unspecified in C99 whether string literals that share a common ending are stored in distinct memory locations. Therefore, even if an attempt to modify a string literal appears to succeed, it is possible that another string literal might be inadvertently altered.

40 Examples:

```
char *s = "string";           /* Not OK: s is not const-qualified */
const char *name2(void)
{
    return ("MISRA");        /* OK */
}
```

50 3.5.7 Code Re-entrancy

Rule BSW_Reentrancy_002: Services and APIs are Multicore Re-entrant

55 **Instruction** All services and APIs of a module shall be multicore re-entrant.

60 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.

65 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:

05 ▶ 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.

10 ▶ No assumptions about priorities and scheduling possible (e.g. for interrupt code)

15 ▶ No modification of its own code

20 ▶ No call of other non re-entrant functions

25 Examples:

30 **Non re-entrant code:**

```
sint32 a = 1;
```

```
sint32 foo(void)
```

```
{  
    a += 2;  
    return a;  
}
```

```
sint32 bar(void)
```

```
{  
    return (foo() * 10);  
}
```

35 **Same functionality re-entrant:**

```
sint32 foo(sint32 a)
```

```
{  
    return (a + 2);  
}
```

```
sint32 bar(sint32 a)
```

```
{  
    return (foo(a) * 10);  
}
```

40 **Non re-entrant code:**

```
sint32 a = 1;
```

```
sint32 foo(void)
```

```
{  
    SCHM_ENTER_<Module>_<Res_a>();  
    a += 2;  
    SCHM_EXIT_<Module>_<Res_a>();  
    /* -> value can be modified concurrently */  
    return a;  
}
```

45 **Same functionality re-entrant:**

```
sint32 a = 1;
```

```
sint32 foo(void)
```

```
{  
    sint32 l_a;  
  
    SCHM_ENTER_<Module>_<Res_a>();  
    a += 2;  
    l_a = a;  
    SCHM_EXIT_<Module>_<Res_a>();  
}
```

05 return l_a;
}

10 Rule BSW_Reentrancy_003: Scheduled Functions (Processes) are Multicore Re-entrant Instruction

15 **Status:** removed

ReplacedBy: Rules of chapter [3.5.5](#)

20 Rule BSW_Reentrancy_004: Atomic data access

25 **Instruction** Assumption about atomic data access shall be documented in the chapter for integration in the product documentation.

25 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.

25 Note: Read-Modify-Write C-Instructions (e.g. ++, --, +=, -=, ...) are not atomic and still have to be protected.

30 3.5.8 Providing Version Information

35 Rule BSW_VersionInfo_001: Provision Version Information

35 Instruction

35 **Scope:** Software based on an AUTOSAR SWS

35 Each AUTOSAR based BSW module shall provide information to identify vendor, module and SW version.

40 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.

45 Rule BSW_VersionInfo_002: Module Vendor Identifier

45 Instruction

45 **Scope:** Software based on an AUTOSAR SWS

50 A module vendor identifier "<MODULE>_VENDOR_ID" shall be provided in the module header file.

50 For Bosch the vendor identification number is "6". The vendor Id is defined by HIS.

55 Example:

55 In file Adc.h:
#define ADC_VENDOR_ID 6

60 Rule BSW_VersionInfo_003: Module Identifier

60 Instruction

60 **Scope:** Software based on an AUTOSAR SWS

65 A module identifier "<MODULE>_MODULE_ID" shall be provided in the module header file.

65 Valid module identifiers of AUTOSAR based BSW modules are listed in [Chapter C "List of Basic Software Modules"](#).

65 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
```

```
05 #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
```

10 **Hint** There are similar rules to provide version information also in the BSWMD file: Rule [\[BSWMDImpl_006\]](#) and [\[BSWMDImpl_007\]](#). Please consider that the version information given in the module header is consistent to the version information given in the BSWMD file.

15 **Rule BSW_VersionInfo_005: Provision of GetVersionInfo Interface**

20 **Instruction**

25 **Scope:** Software based on an AUTOSAR SWS

Each AUTOSAR based BSW module shall provide a function `<Module>_GetVersionInfo` to read out published parameters.

25 Each AUTOSAR BSW module shall provide a function to read out the version information of a dedicated module implementation. The provision of this interface is only relevant for modules listed in chapter "[List of Basic Software Modules](#)"

30 .
The prototype of this API is defined as followed:

```
30 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.

35 The following example shows the implementation for module "Adc" based on AUTOSAR R4.0:

```
35 #include "Adc.h"
...
40 #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;
}
45 #define ADC_STOP_SEC_CODE
#include "Adc_MemMap.h"
```

50 **Rule BSW_VersionInfo_006: Usage of GetVersionInfo Interface**

55 **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.

60 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: Suppressing and Resuming Interrupts

Instruction

DerivedFrom: MISRA C:2012 Dir 4.13

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.

Efficient Multicore Support

For efficient multicore support and migration the following lock APIs are available for integrators in form of macro interfaces or function APIs. It is important that respectively one of both kinds of services is called as a pair. With the macro interfaces the pairwise call is ensured by the kind of implementation of the macros.

That two different kinds of interfaces exists depends certainly on the usage of the interfaces. The macros, because of the internal by the implementation secured pairing, can only be used on the same calling level while the function API also can be used on different levels. This means that for example in front of an if clause the lock API rba_BswSrv_GetLockCommon can be called and within the if / else path the release API rba_BswSrv_ReleaseLockCommon can be set multiple times and on multiple positions. But it is in this case also important that depending to the program flow always a pairwise call of the lock and release API is ensured.

It is recommended to call the macros wherever it is possible.

There are different pairs of interfaces for different use cases:

▶ "Common" Resource + Interrupt Lock

Implements busy waiting for "common" resource + interrupt lock on multicore systems. For single core systems it implements an interrupt lock. Nesting is allowed.

Macro Interface:

```
#define RBA_BSWSRV_GET_LOCK_COMMON()
#define RBA_BSWSRV_RELEASE_LOCK_COMMON()
```

05

Function API:

```
extern void rba_BswSrv_GetLockCommon(void);
extern void rba_BswSrv_ReleaseLockCommon(void);
```

10

► Resource + Interrupt Lock on Multicore Systems

Implements busy waiting for specified resource + interrupt lock on multicore systems. For single core systems it implements an interrupt lock. Nesting is not allowed.

15

Macro Interface:

```
#define RBA_BSWSRV_GET_LOCK_NONEST(Lock_pst)
#define RBA_BSWSRV_RELEASE_LOCK_NONEST(Lock_pst)
```

20

Function API:

```
extern void rba_BswSrv_GetLockNoNest(rba_BswSrv_Lock_to* Lock_pst);
extern void rba_BswSrv_ReleaseLockNoNest(rba_BswSrv_Lock_to* Lock_pst);
```

25

► Single Core Interrupt Lock

Implements interrupt lock.

25

Macro Interface:

```
#define RBA_BSWSRV_SUSPEND_CORE_LOCAL_INT()
#define RBA_BSWSRV_RESUME_CORE_LOCAL_INT()
```

30

Function API:

```
extern void rba_BswSrv_SuspendCoreLocalInt(void);
extern void rba_BswSrv_ResumeCoreLocalInt(void);
```

35

This macro interfaces and function APIs shall only be used in <Module>_Cfg_SchM.h or for mapping other APIs.

35

3.6 Rule Set: Style Guide

40

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.

45

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.

50

3.6.1 Common File Style

55

Rule CCode_Style_001: Unique Style

Instruction Each C file, header and script source file shall follow common settings to get an unique style.

55

This rule will ensure that a common set of file-base style guidelines are fulfilled.

60

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.

65

In [Figure 15](#) the file header for C files and header files is shown.

70

Figure 15 Module Header for C Files and Header Files

```
/*<BASDKey>
*****
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2015. 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_u16, c_u16;
```

Allowed:

```
uint16 a_u16;
uint16 c_u16;
```

Rule CCode_Style_007: Initialization of Arrays and Structures

Instruction

DerivedFrom: MISRA C:2012 Rule 9.2

DerivedFrom: MISRA C:2012 Rule 9.3

Braces shall be used to indicate and match the structure in a non-zero initialization of arrays, unions and structures.

This rule applies to initializers for all objects and subobjects of an array, union or structure. 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_u16 = (a_u8 * b_u8) + c_u16;

if (var1 && var2)
...
if (!var1)
...

result = (value1 >= value2) ? value1: value2;
result ^= value1;

result++;

```

Rule CCode_Style_009: Nested Preprocessor Commands

Instruction Nested blocks of 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. The following example will show the principle.

```

#if (...)           <-- First level
...
# if (...)        <-- Second level (one whitespace)
...
# ifdef (...)     <-- Third level (two whitespaces)
...
# endif           <-- Third level (two whitespaces)
...
# else             <-- Second level (one whitespace)
...
# endif           <-- Second level (one whitespace)
#endif             <-- First level

^ # is on the same line

```

This rule is applicable only for blocks of nested preprocessor commands. It shall not be used for all of the preprocessor commands in a complete file. The benefit of the indentation is lost if additionally all existing #includes or #defines are indented in general. Therefore only "should" is mentioned in the rule but not "shall" that it is applied only for the relevant cases.

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. Then 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

DerivedFrom: MISRA C:2012 Rule 15.6

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 body of an iteration-statement or a selection-statement shall be a compound-statement.

The body of an iteration-statement (while, do ... while or for) or a selection -statement (if, else, switch) shall be a compound-statement. It is possible for a developer to mistakenly believe that a sequence of statements forms the body of an iteration-statement or selection-statement by virtue of their indentation. The accidental inclusion of a semi-colon after the controlling expression is a particular danger, leading to a null control statement. Using a compound-statement clearly defines which statements actually form the body. Additionally, it is possible that indentation may lead a developer to associate an else statement with the wrong if. An 'else if' needs no own braces but the following else path needs to have a compound statement (see [\[CCode_CntrFlow_007\]](#)).

Examples:

Function:	for loop:	switch-case:
void MyModule_Func(void)	for (I = 0; I < 10; I++)	switch (var)
{	{	{
...	...	case 1:
}	}	{
		...
		}
		break;
Typedef definition:	do-while:	case 2:
typedef struct	do	{
{	{	...
uint8 a_u8;	...	
uint8 b_u8;	} while (a > b);	
}		...
MyStruct_tst;		

```

05      if-else:
if (a > b)
{
    ...
}
else if (b > c)
{
    ...
}
else
{
    ...
}

10     while:
while (a > b)
{
    ...
}

15     default:
default:
{
    ...
}
break;
}
}

```

20 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.).

25 Example:

```

30 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

```

35 It is not allowed to write it in the following way:

```

40 void MyModule_Func(void) {
    /* Instructions */
}

45 
```

Rule CCode_BlockStyle_003: Horizontal Position of Brackets of Blocks

Instruction "{" and "}" shall be placed at the same horizontal position.

45 Example:

```

50 void MyModule_Func(void)
{
    /* Instructions */
}

55 ^ Same horizontal position

```

55 Rule CCode_BlockStyle_004: Instruction Block with One Instruction is a Block

Instruction

DerivedFrom: MISRA C:2012 Rule 15.6

60 If an instruction block only contains one instruction, the block shall nevertheless be embraced with "{" and "}".

Example:

```

65 if (a > b)
{
    a = b; <-- Only one instruction is still enclosed with curly brackets.
}

```

05
It is not allowed to write it in the following way:

```
if (a > b) { a = b }
```

10

3.6.3 Comments

15

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.

25

Rule CCode_Comments_002: Allowed Comment Marker

Instruction It is allowed to use "/* ... */" or "//" as comment marker styles to write a comment.

30 /* ... */ is the standard comment style defined in ISO C90 standard. // is an extension but it is supported by most compilers.

35

Rule CCode_Comments_003: Non-Nested Comments

Instruction

DerivedFrom: MISRA C:2012 Rule 1.3

DerivedFrom: MISRA C:2012 Rule 3.1

Comments shall not be nested and comment styles shall not be mixed.

45 C does not support the nesting of comments even though some compilers support this as a language extension. If a comment starting sequence, /* or //, occurs within a /* comment, is it quite likely to be caused by a missing */ comment ending sequence. The sequence // is permitted within a // comment.

50

Rule CCode_Comments_004: Sections of Code Should Not be Commented Out

Instruction

DerivedFrom: MISRA C:2012 Dir 4.4

55 Sections of code should not be "commented out".

This rule applies to both // and /* ... */ styles of comment.

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 constructs with a comment). Using start and end comment markers for this purpose is dangerous because C does not support nested comments (see [\[CCode_Comments_003\]](#), and any comments already existing in the section of code would change the effect).

65

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_009: No Line-Splicing within // Comments

Instruction

DerivedFrom: MISRA C:2012 Rule 3.2

Line-splicing shall not be used in // comments.

Line-splicing occurs when the \ character is immediately followed by a new-line character. If the source file contains multibyte characters, they are converted to the source character set before any splicing occurs. If the source line containing a // comment ends with a \ character in the source character set, the next line becomes part of the comment. This may result in unintentional removal of code.

Example:

```
void MyFunc(Boolean Arg_b)
{
    uint16 x = 0; // comment \
    if (Arg_b)
    {
        ++x;      /* Not OK because this line is always executed */
    }
}
```

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 the following suggested format shall be used: TRACE[<AUTOSAR SWS Trace ID>].

Example (Flash driver, AUTOSAR_SWS_FlashDriver):

05 /* TRACE[FLS165] Implementation of the Fls_GetVersionInfo interface. */

10 **Rule CCode_Comments_008:** Publishable Contents of Comments

15 **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"

20 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.

25

30

35

40

45

50

55

60

65

70

4 Rule Set: ECU Configuration

4.1 Introduction to ECUC

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 configurable¹. 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.

¹ Applying the ECUC methodology to application software or complex drivers is explicitly allowed by AUTOSAR, however.

05

4.1.1 Authoring of ECUC artifacts

10 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.

15

Rule ECUC_001: AUTOSAR conformance

20

Instruction

25 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).

30 All AUTOSAR ECUC artifacts shall be conforming to the corresponding AUTOSAR specifications.

35

Rule ECUC_002: AUTOSAR version

40

Instruction

45 On the head of the main development branch, all AUTOSAR ECUC artifacts shall be based on AUTOSAR release 4.0.2 or higher.

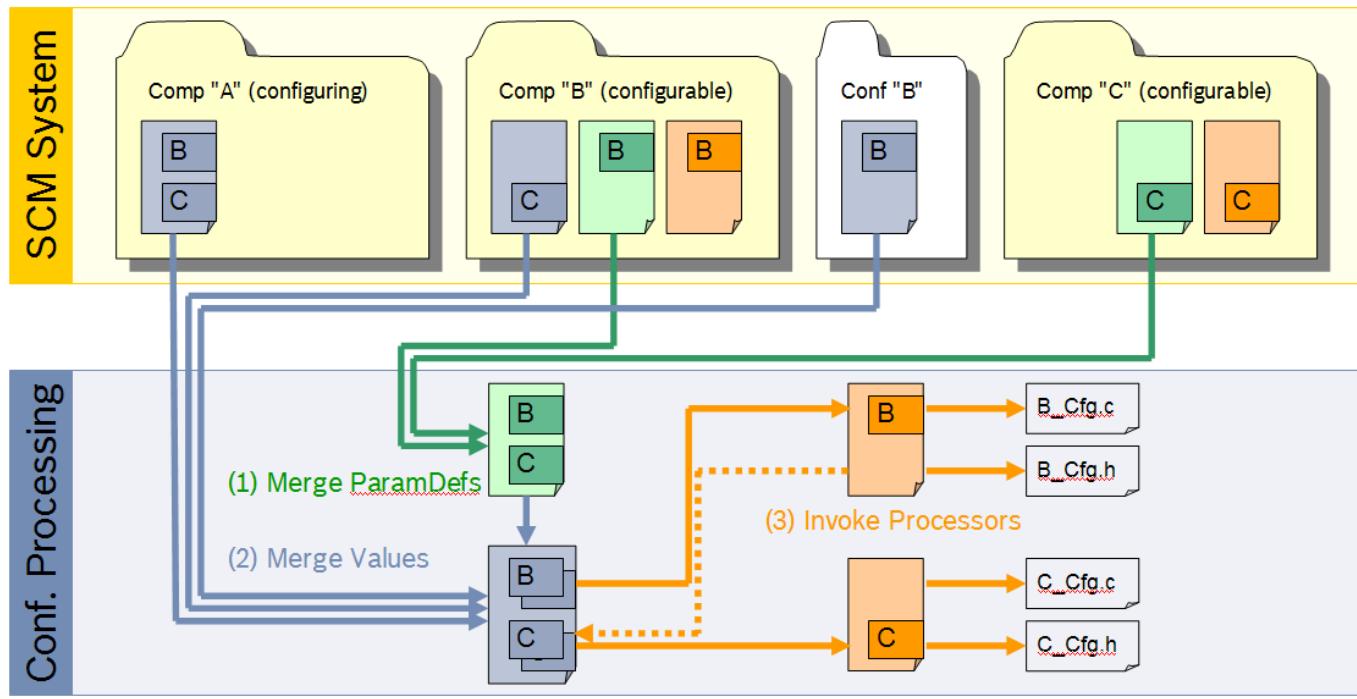
50

4.1.2 How ECUC Values Are Processed

55 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 processor² 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.

60
65
70
75
65
60
55
50
45
40
35
30
25
20
15
10
05
00
² In some AUTOSAR documents, ECUC processors are also called "BSW module generators" or simply "configuration tools".

Figure 19 Processing of ECUC Values



Symbols:  ECUC Values for X  ECUC ParamDefs for X  ECUC Processor for X

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

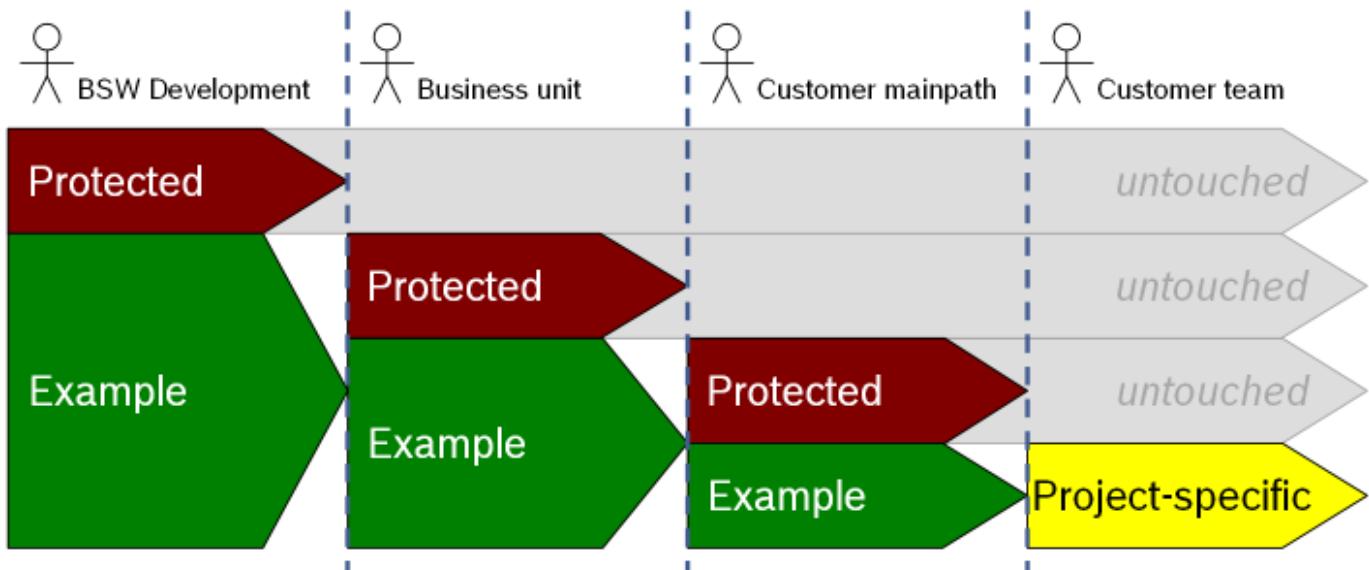
4.2.1 ECUC Values: Content Responsibility

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 responsibles.
- ▶ 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 responsible 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.arxml³.

Rule ECUC_V002: Protected configuration

Instruction

Protected ECUC values shall not be changed by any developer who receives these files as a deliverable⁴.

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.

³ See also the guideline on file naming rules for an overview of all file naming conventions.

⁴ 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.

05
Rule ECUC_V004: Release conditions10
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.

15
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).

20
Rule ECUC_V005: Editing25
Instruction

ECUC values shall be edited with BCT.

30
Rule ECUC_V006: Formal checks35
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.

40
Rule ECUC_V007: Reviewing45
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.

55
4.2.3 ECUC Values: Testing60
Rule ECUC_V008: Testing65
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.

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.arxml**.

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 [\[Document Excel-based ParamDef generator / URL: file:///bosch.com/dfsrb/Dfs-DE/DIV/CDG/Prj/CUBAS/70_EcuConfiguration/80_Tools/Converters/Xls2EcucParamDef\]](http://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.

05 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

- 10 ▶ no existing parameter definition may be removed and that
▶ new parameter definitions are made optional.

15 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.

15 Rule ECUC_PD003: Formal checks

Instruction

20 ECUC parameter definitions shall be formally checked for validity against the corresponding version of the AUTOSAR schema before delivery.

25 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:

- 30 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)
35 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)

40 Rule ECUC_PD004: Reviewing

Instruction

45 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

50 The reviewing of ECUC parameter definitions is currently regarded as being a sufficient quality measure. Hence, no dedicated tests are prescribed for these artifacts.

55 4.3.4 ECUC ParamDefs: Content-related Procedures

55 Rule ECUC_PD005: Vendor-specific derivation

Instruction

60 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.

65 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_IoSigDio_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

05

2. all references in alphabetical order
3. all sub-containers in alphabetical order.

10

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]."

20

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.

25

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.

30

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>
    </ECUC-ENUMERATION-LITERAL-DEF>
  </LITERALS>
</ECUC-ENUMERATION-PARAM-DEF>

```

35

40

45

50

55

60

65

70

```

05      </LONG-NAME>
<ORIGIN>RB:0.1.0:2012-01-16</ORIGIN>
</ECUC-ENUMERATION-LITERAL-DEF>
</LITERALS>
</ECUC-ENUMERATION-PARAM-DEF>
10

```

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. NvMR_bBlockUse-SetRamBlockStatus (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_ECU, 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.

05

4.4 ECUC Processors

10

4.4.1 ECUC Processors: Content Responsibility and Language

15

Rule ECUC_P001: Content responsibility

20

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.

25

Rule ECUC_P002: Language

30

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.

40

4.4.2 ECUC Processors: Editing, Checking, and Reviewing

45
No tools are prescribed for ECUC processor editing or checking.

Rule ECUC_P003: Reviewing

Instruction

50
Before delivery, ECUC processors shall be reviewed against the corresponding requirements and existing content-related procedures (see below).

55

4.4.3 ECUC Processors: Testing

Rule ECUC_P004: Testing

Instruction

60
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.

65
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.

05 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.

10 4.4.4 ECUC Processors: Documentation

15 Rule ECUC_P005: Documenting

Instruction

20 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.

25 This documentation is part of the module's concept/design documentation.

30 4.4.5 ECUC Processors: General Content-related Procedures

35 Rule ECUC_P006: Allowed output

Instruction

40 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 configuration⁵.

45 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.

50 For generated code and meta information, the same quality demands and coding guidelines apply as for manually created code.

55 Rule ECUC_P007: Stable output

Instruction

60 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:

- 65
- ▶ 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.

70 Rule ECUC_P008: Allowed input

Instruction

75 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.

65 ⁵ Within CDG, the only approved exception are the components rba_Wdg* which are allowed to generate artifacts of components Wdg_6_*.

05 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.

10 Due to maintenance issues, a global configuration values section (also known as GLOBDATA) is not allowed.

15 Rule ECUC_P009: Independence from input origin

Instruction

15 ECUC processors are not allowed to react differently in case of

- ▶ manually created input ECUC values resp.
- ▶ forwarded ECUC values.

20 Rule ECUC_P010: Independence from container short names in ECUC Values

Instruction

25 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.

30 Rule ECUC_P011: (Removed)

Instruction

35 Removed.

40 Rule ECUC_P012: Naming conventions for input files

Instruction

45 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.

50 Rule ECUC_P013: Naming conventions for output files

Instruction

55 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>_Lcfg[_<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**

65 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 OAW files and methods corresponding to processor actions shall be named according to the following scheme (see also rule ECUC_P014).

The <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders in a Perl processor.

A oAW processor for a component <Comp> shall be a set of oAW scripts with the following naming conventions:

Workflow Control (MWE, Model Workflow Environment) files:

- for configuration forwarding: <Comp>_Forward[<Suffix>].mwe
- for configuration validation: <Comp>_Validate[<Suffix>].mwe
- for code generator: <Comp>_Generate[<Suffix>].mwe
- for ID generation: <Comp>_Prepare[<Suffix>].mwe

05 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.

10 Forwarder scripts:

- configuration forwarder files: <Comp>_Forward[<Suffix>].ext

15 Generate ID scripts:

- ID generator files: <Comp>_Prepare<Suffix>.chk
- ID generator extension files: <Comp>_Prepare[<Suffix>].ext

20 Validator scripts:

- configuration validation files: <Comp>_Validate[<Suffix>].chk
- configuration validation extension scripts: <Comp>_Validate[<Suffix>].ext

25 Generator scripts:

- configuration generator template (Xpand) files: <Comp>_Generate[<Suffix>].xpt
- configuration generator extension (Xtend) files: <Comp>_Generate[<Suffix>].ext
- C code file generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_c.xpt
- C header file generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_h.xpt
- RTE configuration generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_Bswmd_<Suffix>.xpt and <Comp>_-[<Suffix>]_Cfg_Swcd_[<Suffix>].xpt

35 The file name part <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders or prepare scripts within the same component.

40 Examples: BswM_Generate.mwe, Dem_Events.chk, WgdM_Genutils.ext.

45 Caution Please also respect the naming conventions of input and output data and artifacts in the chapter ECUC Processors: Content-related Procedures.

50 Tip As rule of thumb Xpand scripts that generates a file shall be named exactly like the generated filename, where the point in its extention is replaced by _.

55 Example: Dem_Cfg_h.xpt generates the file Dem_Cfg.h

Rule ECUC_OP_002: Subdirectories

60 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 processing⁶ 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 WhoAmI or Register in AUTOSAR BSW

Instruction

The legacy Perl subroutines WhoAmI() and Register() shall not be used in AUTOSAR BSW.

⁶ 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.

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 PaColn 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
- [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'.

05

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).

10

Parameters: [0] Caller depth (normally 0).

15

Returns: Enriched caller list.

GetCDefine:

Description: This routine generates a pretty-printed list of C #defines from the provided parameters.

20

For example, by writing

25

```
conf_process::GetCDefine(
    "Input signals",
    [ "Name", "Id" ],
    [
        [ "CLUTCH", "1" ],
        [ "BRAKE", "12" ],
        [ "ACCELERATION", "123" ]
    ]);
```

30

you get

35

```
/* Input signals */
#define CLUTCH      1
#define BRAKE       12
#define ACCELERATION 123
```

30

Parameters: [0] Headline of the list (put into a C comment).

35

[1] Array of column headers (currently ignored): the first array element describes the symbol and second describes the semantics of its value.

40

[2] Array of definitions. Each definition is again an array where the first array element defines the symbol and the second defines its value.

45

Returns: Pretty-printed list of C #defines as a string.

GetCEnum:

Description: This routine generates a pretty-printed C enum from the provided parameters.

50

For example, by writing

45

```
conf_process::GetCEnum(
    "Input signals",
    [ "Name" ],
    [
        [ "Clutch" ],
        [ "Brake = 12" ],
        [ "Acceleration" ]
    ]);
```

55

you get

50

```
/* Input signals */
enum
{
    Clutch,
    Brake = 12,
    Acceleration
};
```

60

Parameters: [0] Headline of the enum (put into a C comment).

65

[1] Array of column headers (currently ignored): shall contain only one element describing the semantics of the enum values.

70

05

[2] Array of enumeration literals. Each literal definition is again an array with exactly one element defining the C literal name.

10

Returns: Pretty-printed C enum as a string.

GetCStruct:

Description: This routine generates a pretty-printed C typedef struct from the provided parameters.

15

For example, by writing

20

```
conf_process::GetCStruct(
    "signals",
    ["Type", "Property"],
    [
        ["uint8", "Clutch"],
        ["uint16", "Brake"],
        ["uint32", "Acceleration"]
    ]);
you get
typedef struct
{
    uint8 Clutch;
    uint16 Brake;
    uint32 Acceleration;
}signals;
```

25

Parameters: [0] Defines the name of the generated type.

30

[1] Array of column headers (currently ignored): the first array element describes the type and second describes the properties.

35

[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.

40

Returns: Pretty-printed C typedef struct as a string.

GetCStructInit:

45

Description: This routine generates a pretty-printed C struct (or array) initializer from the provided parameters.

50

For example, by writing

55

```
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
};
```

60

Parameters: [0] Defines the type and name of the struct or array to be initialized.

65

[1] Shall always be "".

[2] Array of column headers (currently ignored): shall contain only one element describing the semantics of the initializers.

70

[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.

GetInt32Bit:

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.

05

Returns: Current interface version of the Perl interface of the configuration framework.

IsModuleExistent:

Description: This routine queries whether the specified SW component exists.

10

Parameters: [0] Name of the SW component to be queried for existence.

Returns: 1 if the specified SW component exists, 0 otherwise.

15

Log:

Description: This routine creates an entry in the log file.

20

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.

25

Returns: -

30

LogEmpty:

Description: This routine creates an empty line in the log file.

35

Parameters: -

40

Returns: -

45

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).

50

Parameters: [0] Path to be made portable.

55

Returns: String containing a portable path, i.e. with forward slashes as path separators.

60

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").

65

Parameters: [0] Path to be made relative to the project root.

70

Returns: Relative path.

Pop:

75

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.

80

Parameters: [0] Name of the SW component whose configuration data shall be read.

85

Returns: Configuration data of the specified SW component if existent, undef otherwise.

90

Push:

95

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.

100

Parameters: [0] Name of the SW component which initiates the push operation.

105

[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.

110

[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 EcucParamDef is expected here, e.g. /AUTOSAR_Dio/EcucModuleDefs or /RB/RBA/rba_IoSigDio/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.

05

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

10
15
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).

20

4.5.1 BuildAction Declarations

25
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.

30

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.

35
40
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.

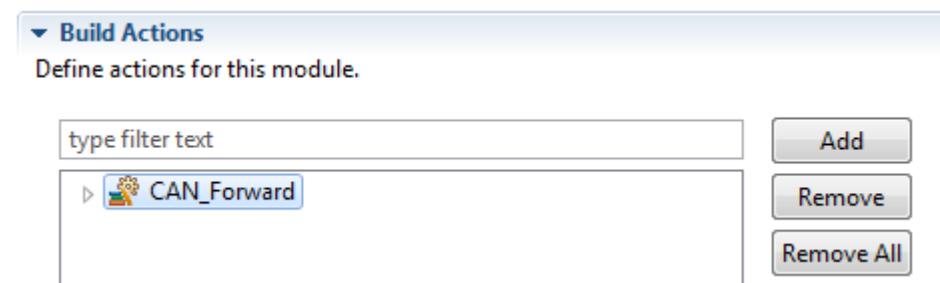
Rule ECUC_B002: Naming conventions for BuildActions

Instruction

45
50
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.

55
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



60
65
55
Define actions for this module.

type filter text

Add

Remove

Remove All

> CAN_Forward

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.

Dio_Generate

PROCESSOR: Dio_Process.pm [CUBAS:CC]

/AUTOSAR_Dio/EcucModuleDefs/Dio

Dio_Prepare

PROCESSOR: Dio_Process.nm [CUBAS:CC]

/AUTOSAR_Dio/EcucModuleDefs/Dio

Add
Remove
Remove All

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

05 build process, passed to other actions, etc. Artefact declarations are also used to determine a valid execution order of BuildActions present.

10 Rule ECUC_BIOA001: Artefact Declaration Composition

15 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.

20 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

25 **Data Details**
Define the selected input / output data element here.

Type*:	Output Data
Category*:	ARTIFACT
Port ID:	
Name*:	rba_Dma_PBcfg
Class*:	PJTSRC
Domain*:	CUBAS
Revision:	
FileType*:	c
Intended Filename:	
Parent Category:	
Parent Shortname:	

45 Rule ECUC_BIOA002: Artefact Declaration Characteristics

50 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.

55 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]_

60 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.

05 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 usefull to take care of the following list of commonly used Classes for Output file artifacts:

10 Table 33 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	PJTSWCD
Measurement/Calibration Support files	arxml	PJTMCSUPP
Report files	txt	PJTREP
BSW Documentation (FS files)	xml	PJTCOREFS

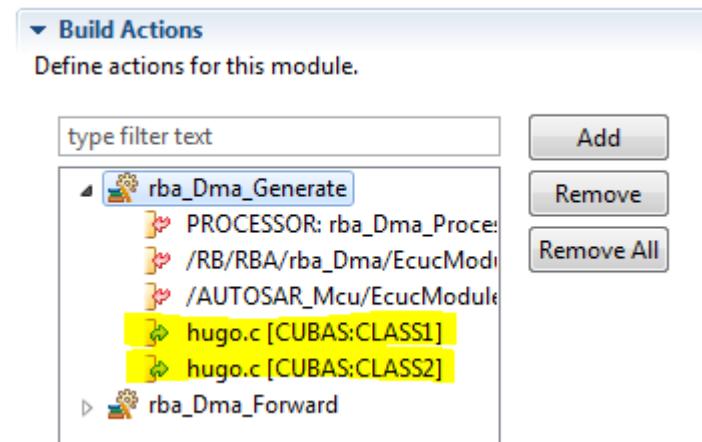
25 Rule ECUC_BIOA003: Uniqueness of Output Declarations

Instruction

30 One BuildAction instance shall never contain duplicate output element declarations which will be applied to one and the same artefact.

35 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:

40 Figure 24 Invalid Output Artefact Declaration



4.5.2.2 Model Reference Declarations

60 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

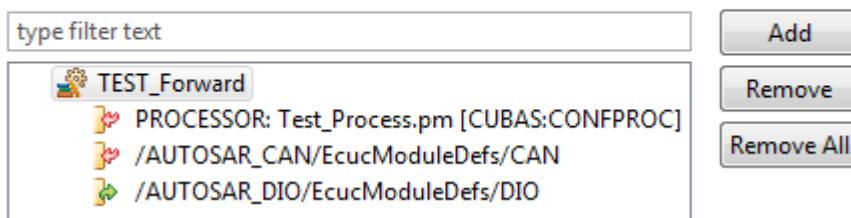
Build scripts of any technical nature (Perl or oAW) shall only access model elements they also explicitly declared in their corresponding Build Action. That means, a script is only allowed to read data from /AUTOSAR_CAN/EcucModuleDefs/CAN if it also declared this AR_OBJECT to be an input IO-element under 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

Build Actions

Define actions for this module.



A screenshot of a software interface showing build actions. On the left is a list box containing the following items:

- TEST_Forward
- PROCESSOR: Test_Process.pm [CUBAS:CONFPROC]
- /AUTOSAR_CAN/EcucModuleDefs/CAN
- /AUTOSAR_DIO/EcucModuleDefs/DIO

To the right of the list box are three buttons: "Add", "Remove", and "Remove All". Above the list box is a text input field labeled "type filter text".

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

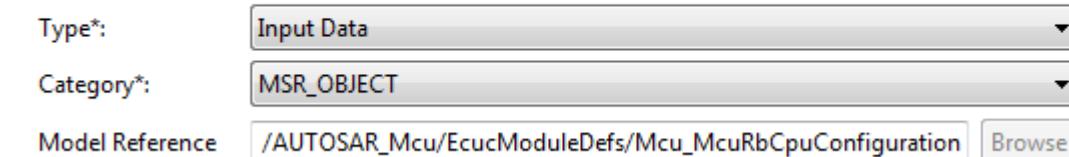
Whenever a model object is referenced by an IO-Element, the corresponding category needs to be used. That is in case of an MSR reference, the category shall be MSR_OBJECT, in case of an AUTOSAR reference, an AR_OBJECT shall be used. The model reference text needs to correspond to the rule mentioned above.

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

Data Details

Define the selected input / output data element here.



A screenshot of a software interface showing data details. The interface includes the following fields:

- Type*: Input Data
- Category*: MSR_OBJECT
- Model Reference: /AUTOSAR_Mcu/EcucModuleDefs/Mcu_McuRbCpuConfiguration
- Browse button (next to the Model Reference field)

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 34 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

Build Actions
Define actions for this module.

Type filter text:

-  rba_Dma_Generate
-  **PROCESSOR**: rba_Dma_Process.pm [CUBAS]
-  /RB/RBA/rba_Dma/EcucModuleDefs/rba_
-  /AUTOSAR_Mcu/EcucModuleDefs/Mcu_
-  rba_Dma_PBcfg.c [CUBAS:PJT SRC]
-  rba_Dma_PBcfg.h [CUBAS:PJT HDR]

Add **Remove** **Remove All**

Data Details
Define the selected input / output data element here.

Type*:	<input type="text" value="Input Data"/>
Category*:	<input type="text" value="ARTIFACT"/>
Port ID:	<input type="text" value="PROCESSOR"/>
Name*:	<input type="text" value="rba_Dma_Process"/>
Class*:	<input type="text" value="CONFPROC"/>
Domain*:	<input type="text" value="CUBAS"/>
Revision:	<input type="text"/>
FileType*:	<input type="text" value="pm"/>
Intended Filename:	<input type="text"/>
Parent Category:	<input type="text"/>
Parent Shortname:	<input type="text"/>

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 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 furtherly 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

Build Action Details

Set the properties of the selected action. Required fields are denoted by *****.

Short Name*	rba_Dma_Generate
Category*	GENERATOR
Required Environment*	/RB_Manifest/com_bosch_configfw_bfwact

Invocation parameters

For perl modules at least the parameters "MODULE_NAME" (whoAmI replacement) and "STEP_NAME" (name of the perl subroutine) have to be set.

Parameter	Value
1 STEP_NAME	Generate
2 MODULE_NAME	rba_Dma

Action Buttons

- Add
- Remove
- Remove All

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
- ▶ a file footer (with the history information from the SCM system)

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
#
#<BASDKey>
#####
# COPYRIGHT RESERVED, Robert Bosch GmbH, 2015. 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.

The file footer shall look like:

Figure 30 Perl Module Footer

```
# <BASDKey>
#####
# $History____:$
#####
#</BASDKey>
```

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:

```
#-----
#
```

```

05    # 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
#
#-----
10   sub FooFunction
15   {
20     ...
}

```

Rule Perl_004: Indentation and spacing style

25 **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

```

# open braces left
if ( $flag_s eq "h" )
{
  $headers_s = 0;
}

# 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;
$vertically_s = 4;

my $filename_s =    $args_h{PATHNAME};
my @names_a   = @{$args_h{FIELDNAMES}};
my $stab_s    =    $args_h{SEPARATOR};

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: $!";

```

05 \$rot13_s =~ tr[a-mn-z]
 [n-za-m];

Rule Perl_005: Line length

10 **Instruction** The line length of 120 characters shall not be exceeded.

5.2 Naming Conventions

Rule Perl_006: Packages and filenames

Instruction

20 **Class:** NamingConvention

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

35 **Class:** NamingConvention

40 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.

```
45    # 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!
```

50 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.

```
60    # 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;
```

05 printf "This is the number of 3*(A_CONST) [0]: %f\n", \$comp2_s;

Rule Perl_008: Naming of global and local variables

Instruction

Class: NamingConvention

- ▶ 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).

25 my \$verbose_s = 0;
my @stateTable_a = (1, 2, 3);

30 foreach my \$value_s (@stateTable_a)
{
 my \$currentLine_s;
 \$currentLine_s = &getLine(\$value_s);
 ...
}

35 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:

40
45 for (my \$index_s = 0; \$index_s < \$#table_a; \$index_s++)
{
 \$table_a[\$index_s] += 2;
}

50 when you should be writing this:

55 for (my \$i = 0; \$i < \$#table_s; \$i++)
{
 \$table_s[\$i] += 2;
}

- ▶ Please use prefixes from the table appendix of this document.

60 # 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)

```

05   {
10     if ($persons_h{$person_s}{"sex"} eq "female")
15       {
16         $cntrFemale_s++;
17     }
18     else
19       {
20         $cntrMale_s++;
21     }
22   }

25   # Prefix 'st'
26   $stRevers_s = GetReversibility();

30   if ($stRevers_s) # success case
31   {
32     ...
33   }

```

Rule Perl_009: References

25

Instruction

Class: NamingConvention

The names of reference variables are suffixed with

- 30 ▶ "_ph" for references to hash variables
- 35 ▶ "_pa" for references to array variables
- 35 ▶ "_ps" for references to scalar variables
- 35 ▶ "_pg" for references to typeglobs
- 35 ▶ "_pf" for references to functions

```

40   # Global variables
41   my @names_a = qw( John Jane);
42   my %address_s = ( "address" => "Park Ave",
43                     "city" => "Baltimore" );
44   # Global references (referencing)
45   my $names_pa = \@names_a;

50   sub Foo
51   {
52     ...
53   }

55   {
56     # Local variables
57     my $state      = "completed";

60     # Local references (referencing)
61     my $address_ph = \%Address;
62     my $state_ps   = \$state;
63     my $stdout_pg   = \*STDOUT;
64     my $foo_pf     = \&foo;

65     # Dereferencing
66     print &$foo_pf(1);
67     print $stdout_pg 'Hello world';
68     print $$state_ps;
69     print %$address_ph;
70     print @$Names_pa;
71   }

```

70

Rule Perl_010: Names of subroutines

Instruction

Class: NamingConvention

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 ist 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

Class: NamingConvention

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

Class: NamingConvention

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.

- 05 ▶ 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.

```
10 # Usual case
my @numbers_a = qw(4 5 6);
my @persons_a = <FILE>;
my %person_h = ( "name" => "Fritz", "age" => 19 );

15 # Exceptions
my %computers_h = ( "aix"      => "134.94.100.100",
                     "solaris"   => "134.94.100.24",
                     "www"       => "134.94.100.51",
                     );
20 my @time_a=localtime(time());
$hour_s  = $time_a[2];
$month_s = $time_a[4];
```

25 5.3 Coding Conventions

Rule Perl_012: Function Calls

30 **Instruction** The first thing to do in a function is to fetch its parameters

- 35 ▶ 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 );
```

40 ...

```
40 sub RaiseToPower
{
    # Formal parameters
    my ( $value_s, $power_s ) = @_;
45    return $value_s ** $power_s;
}
```

50 ...

```
50 sub GetLengthList
{
    # Formal parameters
    my $list1_pa = $_[0];
    my $list2_pa = $_[1];

    # Local parameters
    my $numList1_s;
    my $numList2_s;

60    $numList1_s = @$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 EtksEnable = YES
if($$ecucGeneral_ph{EtksEnable} eq 'YES')
{
    if( conf_process::IsModuleExistent('rba_CalWup') )
    {
        # Get the ECUC values for rba_CalWup
        my $EcucCalWup_ph = conf_process::Pop("rba_CalWup");
        ....
    }
    else
        ....
}
```

Rule Perl_021: Consistency of the Name of a Perl Module File and the Package Name within the Perl Module

Instruction

Class: NamingConvention

The specified package name within a perl module file (.pm file) shall be set identical to the file name of the perl module file if the perl module file is used as processor in a build action.

The perl processor has the demand that the file name of a perl module (name of the .pm file) is identical with the package name within the perl module specified with the 'package' keyword. If the file name is changed (by any reason) the package name specified by the 'package' keyword has to be changed too. The camel case notation has to be considered. More details about build actions can be found in [Chapter 4.5 "Build Action Manifests" p. 189](#)

Example:

```
File MyModule.pm:
package MyModule;
```

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)

- 05 ▶ 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.

10 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.

- 15 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.

20 # An else after a chain of elsif-s
 if (\$number_s == 1)
 {
 &DoSomething(\$number_s);
 }
 elsif (\$number_s > 1)
 {
 &DoSomethingElse(\$number_s);
 }
 else
 {
 #-- EMPTY
 }

25
 30
 35 # Comma at the end of an array or hash
 my %computers_h = ("aix" => "134.94.100.100",
 "solaris" => "134.94.100.24",
 "www" => "134.94.100.51",
);

40 **Rule Perl_015:** Make regular expressions readable

Instruction

- 45 ▶ 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).

50 # OK
 m/\w+:(\s+\w+)\s*\d+/;

BETTER
 m/\w+:(\s+ \w+) \s* \d+/x;

55 # 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

```

05 # Usage of quoted regular expressions
my $gerZipCode_s      = qr/[0-9]{5}/;
my $space_s           = qr/[ \t]+/;
my $city_s            = qr/\w+/;

10 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";
}

20 my $gerLocation_s    = qr/$gerZipCode_s$space_s$city_s/;
if ($example1_s =~ /$gerLocation_s/)
{
    print "--> 3: STRIKE!\n";
}

```

25 Rule Perl_016: Make dereferenciation to references readable

Instruction

- 30 ▶ 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.

35 Here are some examples of references of single arrays and hashes:

```

35 # References to single arrays and hashes
$hugo_pa = [ aa, bb, cc ];
$hugo_ph = {
    'Adam' => 'Eve',
    'Clyde' => 'Bonnie',
};

40 # Dereferencing of single array elements (ok)
$elem_s = $$hugo_pa[0];

45 # Dereferencing of single array elements (better)
$elem_s = ${$hugo_pa}[0];
$elem_s = $hugo_pa->[0];

50 # Dereferencing of an array (ok)
@hugo_a = @{$hugo_pa};

# Dereferencing of an array (better)
@hugo_a = @{$hugo_pa};

55 # Dereferencing of single hash elements (ok)
$elem_s = $$hugo_ph{"Clyde"};

# Dereferencing of single array elements (better)
$elem_s = ${$hugo_ph}{"Clyde"};
$elem_s = $hugo_ph->{"Clyde"};

60 # Dereferencing of a hash (ok)
%hugo_h = %{$hugo_ph};

# Dereferencing of a hash (better)
%hugo_h = %{$hugo_ph};

```

05 Here are some examples of references of more complicate structures:

```

# References to hashes of hashes
10 %hoh_h = (
    "John" => {
        "EMAIL" => "john@doe.com",
        "LOCATION" => "L.A.",
    },
    "Jane" => {
        "MAID_NAME" => "Mueller",
        "EMAIL" => "jane@doe.com",
    }
);

$hooh_ph = {
    "John" => {
        "EMAIL" => "john@doe.com",
        "LOCATION" => "L.A.",
    },
    "Jane" => {
        "MAID_NAME" => "Mueller",
        "EMAIL" => "jane@doe.com",
    }
};

# Access to single hash element (ok, the only way)
$elem_s = $hoh_h{Jane}{MAID_NAME}; # Mueller

30 # Dereferencing of a single sub-hash
%elem_h = %{ $hoh_h{Jane} };

# Access to single hash element (ok)
$elem_s = ${$hooh_ph{Jane}}{MAID_NAME}; # Mueller

35 # Access to single hash element (better)
$elem_s = $hooh_ph->{John}->{LOCATION}; # L.A.
$elem_s = $hooh_ph->{John}{LOCATION}; # L.A.

40 # Dereferencing of a single sub-hash (ok)
%elem_h = %{ $$hooh_ph{John} }; # %elem_h = ( EMAIL => "john@doe.com",
#                               LOCATION => "L.A.", )

45 # Dereferencing of a single sub-array (better)
%elem_h = %{ ${ $hooh_ph }{John} }; # ditto
%elem_h = %{ $hooh_ph->{John} }; # ditto

# References to arrays of arrays
50 @lol_a = (
    [1, 2],
    [3, 4]
);

$lol_pa = [
    [1, 2],
    [3, 4]
];

55 # Access to a single array element
$elem_s = $lol_a[1][0]; # '3'

60 # Dereferencing of a single sub-array
@elem_a = @{$lol_a[1]}; # @elem_a = (3, 4)

65 # Access to a single array element (ok)
$elem_s = ${ $$lol_pa[1] }[0]; # '3'

# Access to array hash element (better)

```

```

05 $elem_s = $lol_pa->[1]->[0]; # '3'
$elem_s = $lol_pa->[1][0]; # '3'

# Dereferencing of a single sub-array (ok)
@elem_a = @{$$lol_pa[1]}; # @elem_a = (3, 4)

10 # 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)

15 Rule Perl_017: Handling of multi-line strings

Instruction For the definition of multi-line strings should preferred here documents.

20 $price_s = '$100';

$here1_s = << "EOF";
Hey
    The price is $price_s.
Bye
EOF

25 $here2_s = << 'EOF';
Hey
    The price is $price_s.
Bye
EOF

30 print $here1_s;
#
# Hey
#     The price is $100.
# Bye
#

35 print $here2_s;
#
# Hey
#     The price is $price_s.
# Bye
#
40
45

```

Rule Perl_018: Local overwriting of global variables

Instruction Avoid the overwriting of global variables and furthermore of Perl system variables.

50 If the overwriting is necessary, use a local declaration within its own block:

```

# Enabling whole-file mode
{
  local $/;
  $file_s = <FH>;
}

```

60 5.5 Templates

65 **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 .perltidirc:

```
# This is a simple example of a .perltidirc configuration file
--indent-columns=4
--maximum-line-length=120
```

Some options which can be useful are the following:

```
# This is a simple of a .perltidirc 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 [\[Document The Perltidy Home Page / Name: Perltidy / URL: http://perltidy.sourceforge.net/\]](#)

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/method shall have a comment describing this function or method. Additional 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>

 10   <!-- 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>

 15   <!-- 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>

 20   <!-- 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>

 25   <!-- 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>

 30   <!-- 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 31 Module Header for Xtend Files

```
/*<BASDKey>
*****
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2015. 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 32 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

Class: NamingConvention

Whenever a function is using aspect orientation and may be overloaded by an aspect function, it shall have the string "Aspect" in it's 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!

Caution Usage of Aspect is not recommended. Instead please use validator to check for configuration which is mandatory for your component. See also rule OAW_Auxilliary_001.

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:

```

05  /*
10   * return a list of Mode Conditions with SchM Generic Mode Request Port.
15   */
20 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:

```

30 private String GetString() :
    "Hello World!"
;
```

Rule OAW_Xtend_006: Function Names

Instruction

Class: NamingConvention

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:

```

45 pDemConfigSet Dem_getDemConfigSet(Dem this) :
    this.demConfigSet
;
```

Example: A generator extension defined for non-Autosar component DemFifo:

```

50 String rba_DemFifoGetString() :
    "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

Rule OAW_Xtend_010: Usage of .set Methods are Forbidden for Forwarders

Instruction .set methods shall not be used in forward scripts.

The .set methods can unintentionally overwrite data, therefore forward scripts shall use the new oAW forwarding API defined by the tools department.

ID generator scripts historically use .set methods, but also there it has to be considered to rework them. E.g. no project should configure those IDs manually, then they can all be forwarded properly.

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 33 Module Header for Xpand Files

```
<REM>
<BASDKey>
*****
* COPYRIGHT RESERVED, Robert Bosch GmbH, 2015. 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 34 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::BswMModule»
```

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 – ENDAROUND, 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

Class: NamingConvention

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 ID generate 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 method 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.

```
30    /* isModuleExistent returns true if the specified module name is present in
       the configuration tree (both EcucParamdef and configuration available)
       Context: Autosar
*/
Boolean isModuleExistent(String moduleName)
```

Usage in ext:

```
if (False != isModuleExistent("FrTp")) then { // do some stuff }
```

Usage in xpt:

```
<IF (isModuleExistent("FrTp")) != false) ->
#define USE_FRTCP (1)
<ELSE ->
#define USE_FRTCP (0)
<ENDIF ->
```

Hint Since isModuleExistent is only applicable for the AUTOSAR context, make sure to call it accordingly.

E.g.: «IF (this.parent.parent.isModuleExistent("BswM")) != false -»

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.

05

7 Rule Set: Data Description

10
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.15
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. 122](#) the BSWMD file is one of the mandatory files of a BSW module.20
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 guideline. 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.25

7.1 Data Types

30

7.1.1 Interaction between DataPrototype and DataTypes

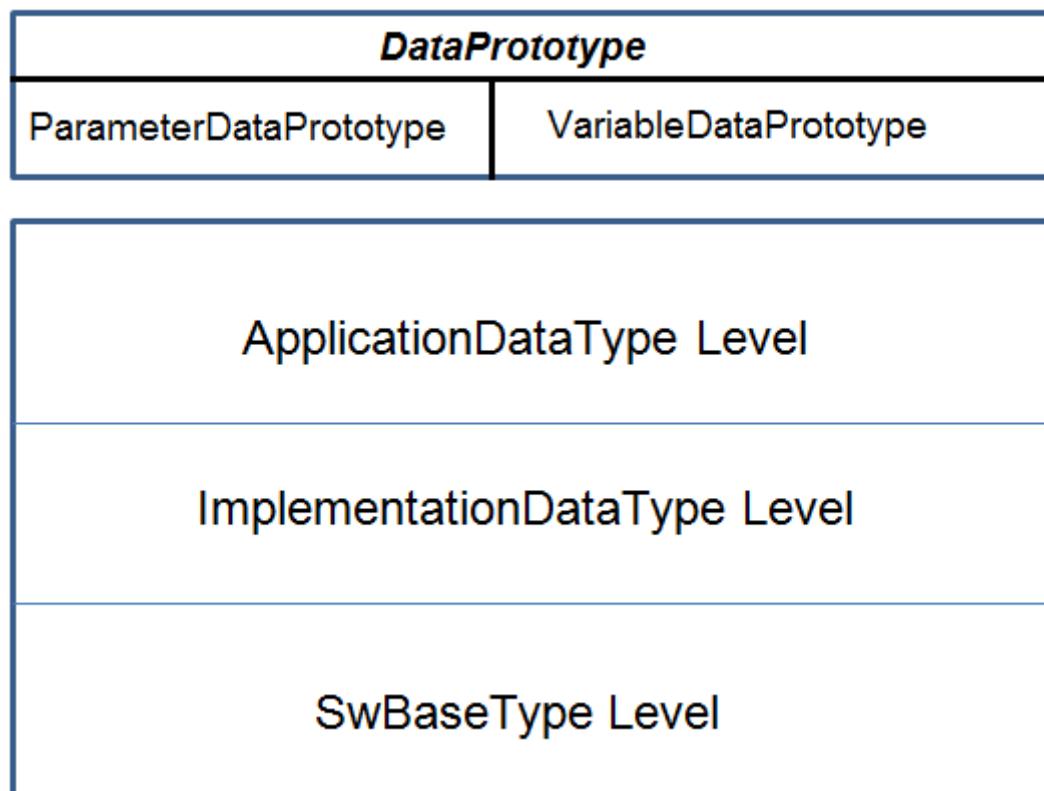
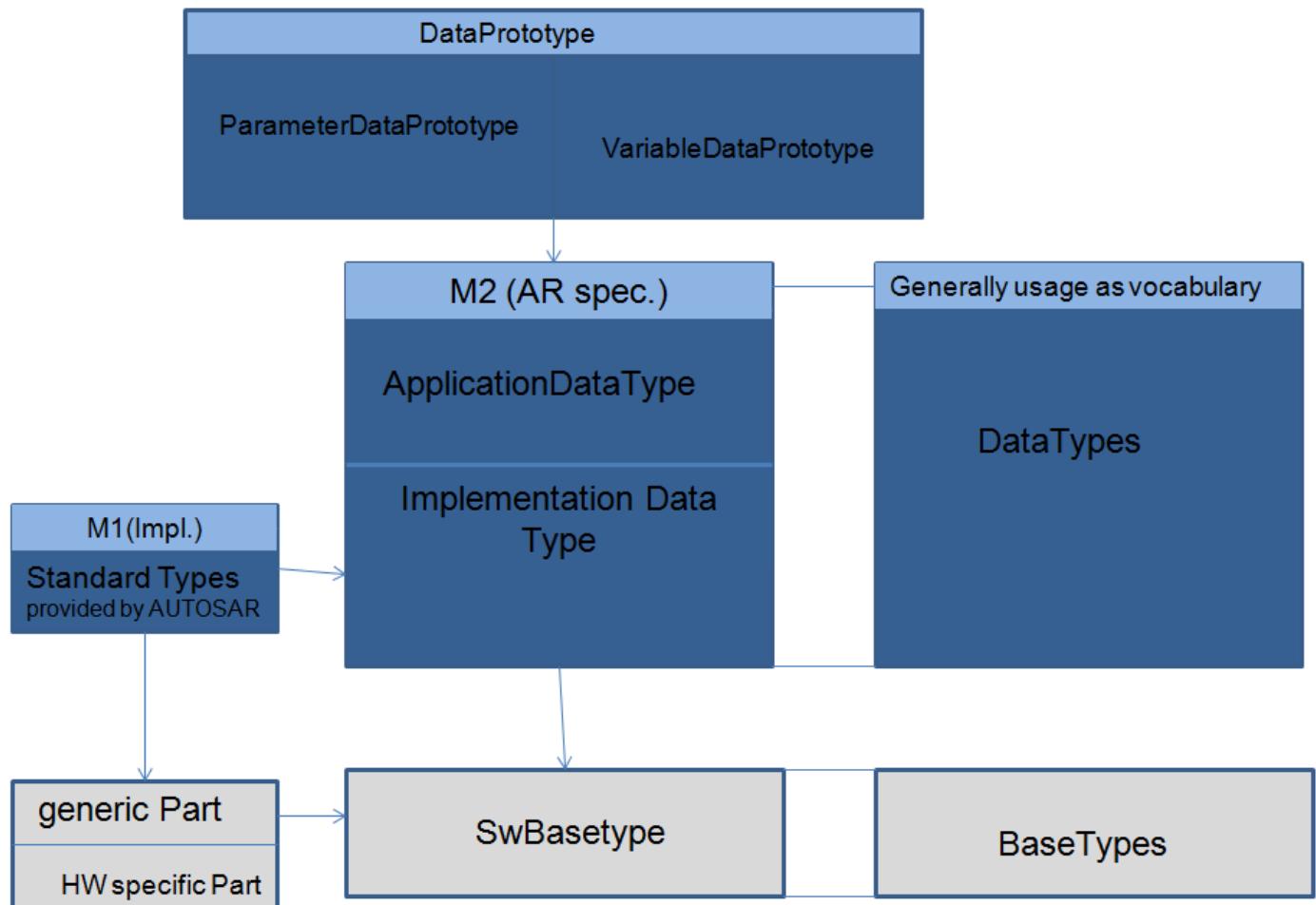
35
In AUTOSAR the meaning of data type is widely extended. It includes not only the C-data types but may also contain attributes which defines semantics. The (C-)language implementation properties shall be widely hidden from the functional developers. Therefore three layers of data type abstraction were introduced in AUTOSAR R4.0.40
Figure 35 Data Types 4.x70
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 36 Overview of AUTOSAR Types



7.1.2 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. 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 or calibration parameters. It does not consider implementation details such as bit-size, endianess, 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.

Hint If two or more data prototypes have the same set of values at their application model, then a reuse of respective *ApplicationDataType* is recommended.

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. Attributes of the SW data definition properties that are applicable for various categories as per AUTOSAR are described in "*Application Interfaces User Guide*".

ApplicationDataType can be standardised by AUTOSAR. If a standardised *ApplicationDataType* is available, then developer shall use this standardised *ApplicationDataType* only. *ApplicationDataType* shall be created only if standardised ones does not exist.

ApplicationDataTypes can be broadly classified as follows:

- ▶ Primitive *ApplicationDataType*
 - Primitive *ApplicationDataTypes* applicable for both BSW and ASW
 - 1. VALUE**
 - 2. BOOLEAN**
 - 3. VAL_BLK**
 - ASW specific Primitive *ApplicationDataTypes*
 - 1. CURVE**
 - 2. MAP**
 - 3. COM_AXIS**
- ▶ Complex *ApplicationDataTypes*
 - 1. ARRAY**
 - 2. STRUCTURE**

Hint The categories listed above are supported by AR_A2L Gen

The list of all the categories that are applicable to *ApplicationDataType* can be found at "[Overview of AUTOSAR Categories, Data types and related Elements](#)"

7.1.3 Primitive *ApplicationDataTypes* applicable for both BSW and ASW

The meta-class *ApplicationPrimitiveDataType* in combination with the attached data properties are used on the application level (M2 level) to specify the details on implementation level (M1 modeling level). The data properties specify the lower and upper ranges that constrain the applicable value interval for the data, the accessibility constraints of the data such as READ-ONLY, READ-WRITE, their physical units and so on. All this data properties are addressed by the attribute *SwDataDefProps*.

In addition to data properties, these primitive data types are categorized based on the nature of data on application level. Data may be integer, string and so on. This categorization is taken care with the help of the attribute *Category*.

The following section describes primitive *ApplicationDataType* that are applicable at both Application software level and Basic Software level

7.1.3.1 *ApplicationDataType_Value*

ApplicationPrimitiveDataTypes are used to specify integer data type. These *ApplicationPrimitiveDataType* shall have their *Category* set to "Value". *ApplicationPrimitiveDataTypes* are uniquely represented with their *ShortName*.

Data properties are defined as part of *SwDataDefProps*. An *ApplicationPrimitiveDataType* shall have the following properties defined.

- ▶ *SwCalibrationAccess*: *SwCalibrationAccess* shall define access privilege applicable for the calibration data. If a *SwCalibrationAccess* is defined at *SwDataDefProps* of *ApplicationPrimitiveDataType* and at *SwDataDefProps* of *DataPrototype* level, then *SwCalibrationAccess* defined at *DataPrototype* level is used.
- ▶ *CompuMethodRef*: A reference to *CompuMethod* shall be given using *CompuMethodRef* attribute. "DEST" is set to "COMPU-METHOD". *ApplicationPrimitiveDataType* shall have atmost one reference to *CompuMethod*
- ▶ *DataConstrRef*: Reference to *DataConstraint* shall be given using *DataConstrRef* attribute. "DEST" is set to "DATA--CONSTR". *ApplicationPrimitiveDataType* shall have atmost one reference to *DataConstraint*

- 05 ▶ *DisplayFormat*: *DisplayFormat* is used to control the resolution of the calibration data displayed in calibrating tool. An *ApplicationPrimitiveDataType* may contain a *DisplayFormat*. If *DisplayFormat* is defined at *SwDataDefProps* of *ApplicationPrimitiveDataType* and *SwDataDefProps* of data prototype then, *DisplayFormat* defined at data prototype is considered. If *DisplayFormat* is defined at *SwDataDefProps* of *ApplicationPrimitiveDataType* only then this *DisplayFormat* is considered. If it is not defined at *SwDataDefProps* of *ApplicationPrimitiveDataType* and *SwDataDefProps* of data prototype, then a default *DisplayFormat* is generated by ArA2L gen in A2L file.
- 10 ▶ *SwImplIPolicy*: *ApplicationPrimitiveDataType* may have *SwImplIPolicy*. If a *SwImplIPolicy* is defined at *SwDataDefProps* of *ApplicationPrimitiveDataType* and at *SwDataDefProps* of DataPrototype level, then *SwImplIPolicy* defined at DataPrototype level is used.
- 15 ▶ *SwRecordLayoutRef*: *ApplicationPrimitiveDataType* may have a reference to *SwRecordLayout*. If it is not defined then a default *SwRecordLayout* is generated by ArA2Lgen in A2L file.

20

25

30

35

40

45

50

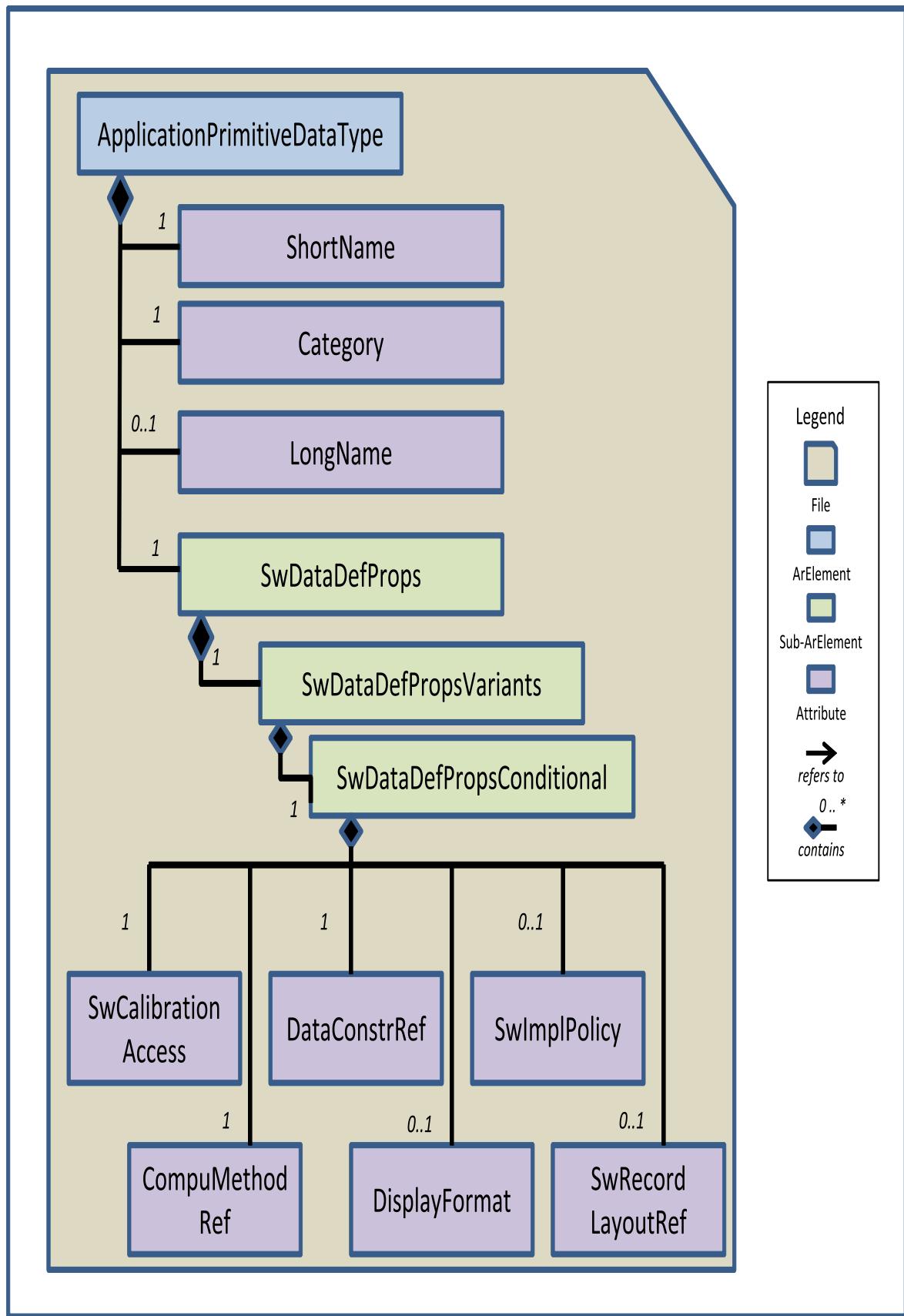
55

60

65

70

Figure 37 ApplicationDataType_Value

**Example:**

```

<AR-PACKAGE>
  <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
  <ELEMENTS>
  
```

```

05   <APPLICATION-PRIMITIVE-DATA-TYPE>
10     <SHORT-NAME>ApplicationDataType_Primitive</SHORT-NAME>
15     <CATEGORY>VALUE</CATEGORY>
20       <SW-DATA-DEF-PROPS>
25         <SW-DATA-DEF-PROPS-VARIANTS>
30           <SW-DATA-DEF-PROPS-CONDITIONAL>
35             <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
40               <COMPU-METHOD-REF DEST="COMPU-METHOD">
45                 <!-- Refer to a CompuMethod -->
50               </COMPU-METHOD-REF>
55             <DATA-CONSTR-REF DEST="DATA-CONSTR">
60               <!-- Refer to a DataConstraint -->
65             </DATA-CONSTR-REF>
70               <DISPLAY-FORMAT>10.5%</DISPLAY-FORMAT>
75             <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
80             <SW-RECORD-LAYOUT-REF DEST="SW-RECORD-LAYOUT">
85               <!-- Refer to the definition of a SwRecordLayout -->
90             </SW-RECORD-LAYOUT>
95           </SW-DATA-DEF-PROPS-CONDITIONAL>
100         </SW-DATA-DEF-PROPS-VARIANTS>
105       </SW-DATA-DEF-PROPS>
110     </APPLICATION-PRIMITIVE-DATA-TYPE>
115   </ELEMENTS>
120 </AR-PACKAGE>

```

Note:

SwCalibration are explained in [\[SwCalibrationAccess\]](#)

CompuMethods are explained in [\[CompuMethods\]](#)

DataConstraints are explained in [\[DataConstr\]](#)

SwImplPolicy are explained in [\[SwImplPolicy\]](#)

SwRecordLayout are defined centrally. Developers has to refer to a existing SwRecordLayout.

7.1.3.1.1 Rules for ApplicationDataType of Category Value

Rule Data_003: Removed

Instruction

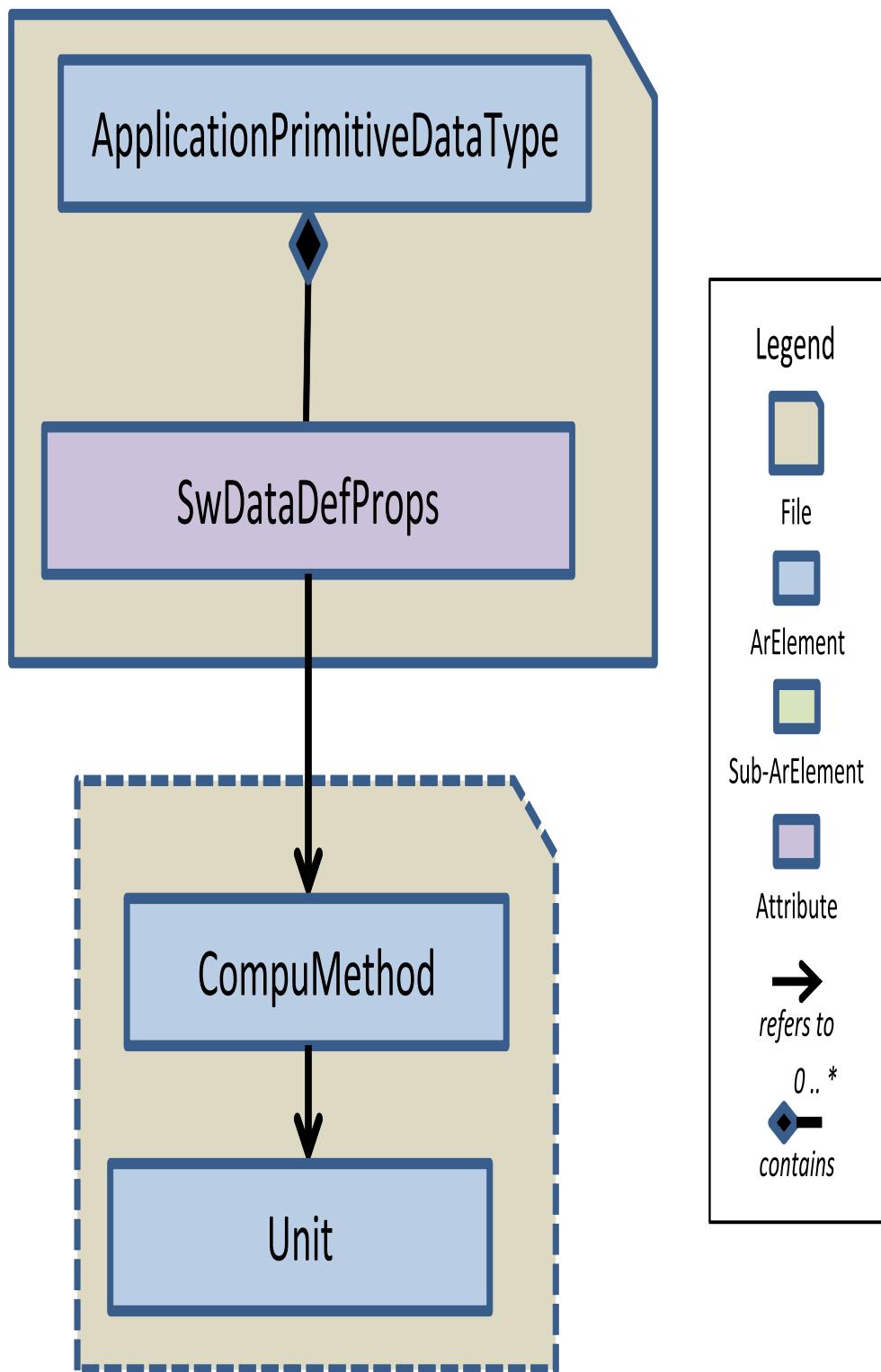
Status: removed

Rule Data_091: Referencing a CompuMethod from an ApplicationDataType

Instruction CompuMethod shall be referenced by an *ApplicationDataType* through *SwDataDefProps* using the attribute <COMPU-METHOD-REF>.

Hint *DataPrototype* will inherit the CompuMethod from *ApplicationDataType*. However referring to *CompuMethod* by *ImplementationDataType* is also possible under special conditions [\[Data_051a\]](#)

Figure 38 Connecting ApplicationDataType with CompuMethod



Rule Data_114: Data type for Application Primitive Data Type

Instruction

Scope: DGS

DerivedFrom: DP0050A-2

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 small number of ImplementationDataTypes as ImplementationDataTypes can be reused. It would be compatible with the data types used in Service-Libs routines.

Rule Data_004a: Where to Define *ApplicationDataType*

Instruction

DerivedFrom: ARPac_14

DerivedFrom: CEL_061

DerivedFrom: Blueprint_011

1. An *ApplicationDataType* that is standardized by AUTOSAR shall be used whenever applicable.
2. An *ApplicationDataType* that is normalized as central element shall be used whenever applicable.
3. An *ApplicationDataType* that is already defined by another *SwComponentType* may be reused. For details refer to Rule Static_0108 [\[A_StaticView\]](#).
4. Module/Component specific *ApplicationDataTypes* shall be defined in the context (name space) of the module / component (that means in component's / module's *ArPackage*, sub- *ArPackage* "ApplicationDataTypes").

Rule Data_119: *DataConstraint* are mandatory for *ApplicationDataType* that are mapped to Float

Instruction ApplicationDataType shall have a DataConstraint, if it is mapped to *ImplementationDataType* of type float.

DataConstraints are mandatory in A2L format. Default constraints cannot be derived from a float data type, in the way as it is possible to derive them from an integer data type

However referring a DataConstraint is optional in AUTOSAR.

Rule Data_142: Definition of *SwDataDefProps* for *ApplicationPrimitiveDataType*

Instruction *SwDataDefProps* for an *ApplicationPrimitiveDataType* shall be defined as follows:

- ▶ A *SwDataDefProps* shall contain a *SwCalibrationAccess*, *CompuMethod* and *DataConstraint*.
- ▶ A *SwDataDefProps* may contain *DisplayFormat*, *SwImplPolicy* and *SwRecordLayout*.
- ▶ Default value for *SwCalibrationAccess* shall be READ-WRITE and for *SwImplPolicy* shall be STANDARD.

7.1.3.2 ApplicationDataType_Boolean

Another category of *ApplicationPrimitiveDataType* that is applicable for both ASW and BSW includes Boolean values. Here the calibration data is allowed to take only 2 values i.e., 0 or 1. Identification of *ApplicationPrimitiveDataType* is done through *ShortName*. *Category* is set to "Boolean". *ShortName*, *Category* and *SwDataDefProps* are mandatory attributes. The definition of *SwDataDefProps* is as follows:

05 ▶ *CompuMethodRef*: A reference to *CompuMethod* of category *IDENTICAL* or *TEXTABLE* shall be given using *CompuMethodRef* attribute. "DEST" is set to "COMPU-METHOD". *ApplicationPrimitiveDataType* shall have atmost one reference to *CompuMethod*

10 ▶ *DataConstraint*: Since there are only 2 values possible, *DataConstraint* is not required.

15 ▶ *SwRecordLayoutRef*: *ApplicationPrimitiveDataType* may have a reference to *SwRecordLayout*. If it is not defined then a default *SwRecordLayout* is generated by ArA2Lgen in A2L file.

20 ▶ All the other attributes are applicable similar to *ApplicationPrimitiveDataType* of category "Value".

15

20

25

30

35

40

45

50

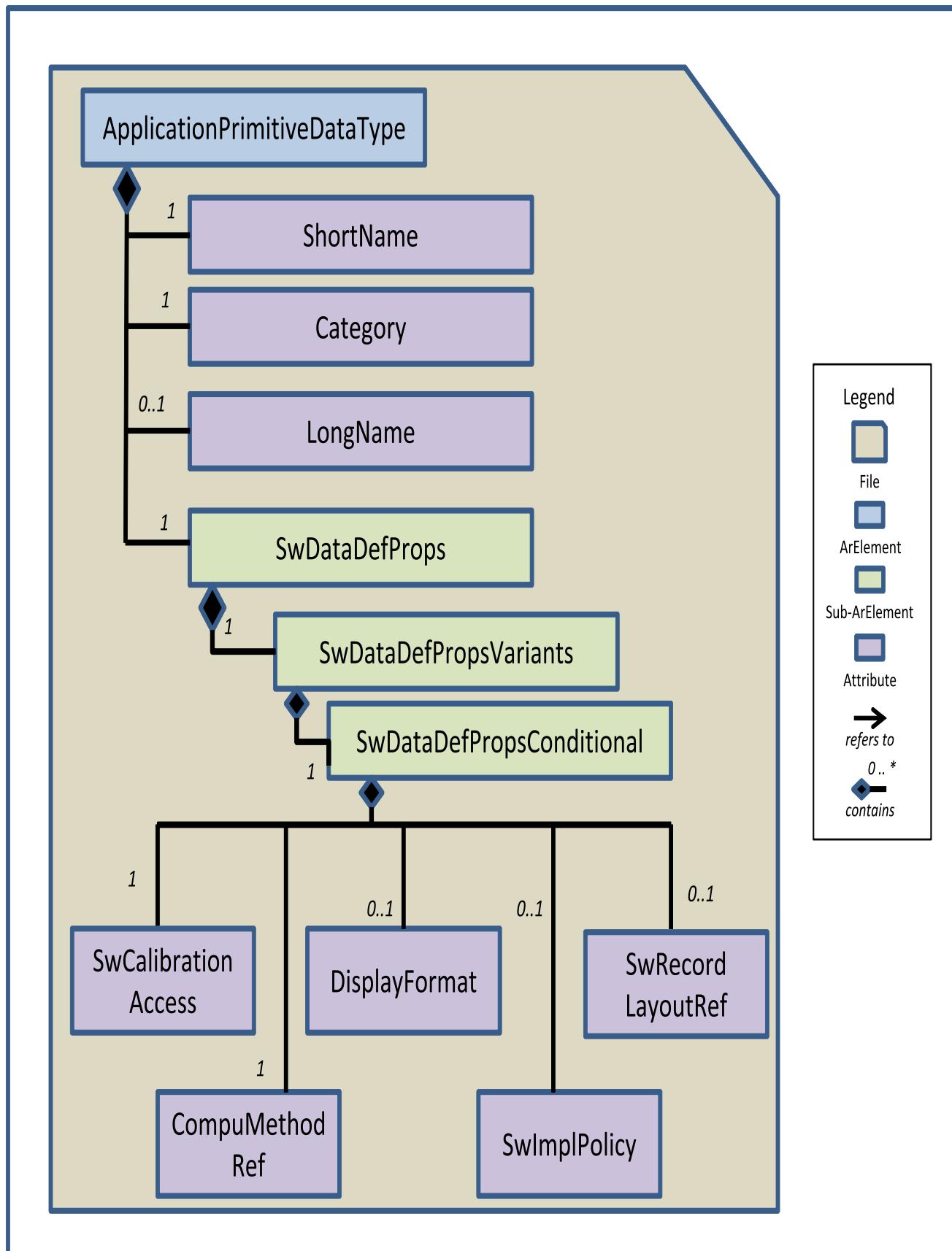
55

60

65

70

Figure 39 ApplicationDataType_Boolean

**Example:**

```

<AR-PACKAGE>
<SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
<ELEMENTS>
  
```

```

05   <APPLICATION-PRIMITIVE-DATA-TYPE>
10    <SHORT-NAME>ApplicationDataType_PrimitiveBool</SHORT-NAME>
15    <CATEGORY>BOOLEAN</CATEGORY>
20    <SW-DATA-DEF-PROPS>
25     <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 of Category IDENTICAL or TEXTTABLE -->
        </COMPU-METHOD-REF>
        <DISPLAY-FORMAT>10.5%</DISPLAY-FORMAT>
        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        <SW-RECORD-LAYOUT-REF DEST="SW-RECORD-LAYOUT">
          <!-- Refer to the definition of a SwRecordLayout -->
        </SW-RECORD-LAYOUT>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
</ELEMENTS>
</AR-PACKAGE>

```

Note:

CompuMethods are explained in [\[CompuMethods\]](#)

7.1.3.2.1 Rules for ApplicationDataType of Category Boolean

Rule Data_173: Definition of *SwDataDefProps* for *ApplicationPrimitiveDataType* of Category Boolean

Instruction *SwDataDefProps* shall be a mandatory element for *ApplicationPrimitiveDataType* of Category Boolean as mandatory element similar to *Short-Name* and *Category*.

SwDataDefProps for an *ApplicationPrimitiveDataType* of category Boolean shall be defined as follows:

- ▶ A *SwDataDefProps* shall contain a *SwCalibrationAccess*, *CompuMethod*.
- ▶ A *SwDataDefProps* may contain *DisplayFormat*, *SwImplPolicy* and *SwRecordLayout*.

Rule Data_155: CompuMethods referenced from ApplicationDataType of Category BOOLEAN

Instruction *ApplicationDataType* of Category BOOLEAN shall refer to *CompuMethods* of Category TEXTTABLE only

7.1.3.3 ApplicationDataType_ValueBlock

A value block defines values stored together within one calibration parameter object. It is similar to value array but it stores the values by means of an axis. Here the category of *ApplicationDataType* will be set to "VAL_BLK". *SwDataDefProps* aggregates *SwCalibrationAccess*, *SwValueBlockSize*, *CompuMethod*, *DisplayFormat* *SwImplPolicy*, *DataConstr* and *SwRecordLayout*. *SwValueBlockSize* defines the number of values each block can have.

Example: VAL_BLK

```

60   <APPLICATION-PRIMITIVE-DATA-TYPE>
65    <SHORT-NAME>ApplicationDataType_ValBlk</SHORT-NAME>
    <CATEGORY>VAL_BLK</CATEGORY>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
          <SW-VALUE-BLOCK-SIZE>10</SW-VALUE-BLOCK-SIZE>
          <COMPU-METHOD-REF DEST="COMPU-METHOD">

```

```

05      <!-- Refer to a CompuMethod -->
</COMPU-METHOD-REF>
10     <DATA-CONSTR-REF DEST="DATA-CONSTR">
      <!-- Refer to a DataConstraint -->
</DATA-CONSTR-REF>
15     <DISPLAY-FORMAT>10.5%</DISPLAY-FORMAT>
<SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
<SW-RECORD-LAYOUT-REF DEST="SW-RECORD-LAYOUT">
      <!-- Refer to the definition of a SwRecordLayout -->
</SW-RECORD-LAYOUT>
</SW-DATA-DEF-PROPS-CONDITIONAL>
</SW-DATA-DEF-PROPS-VARIANTS>
</SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Note:

SwCalibration are explained in [\[SwCalibrationAccess\]](#)

CompuMethods are explained in [\[CompuMethods\]](#)

DataConstraints are explained in [\[DataConstr\]](#)

SwImplPolicy are explained in [\[SwImplPolicy\]](#)

SwRecordLayout are defined centrally. Developers has to refer to a existing SwRecordLayout.

Hint Currently RTE-Generator does not handle category VAL_BLK. The category is reported as "Unkown" in McSupport file.

7.1.3.3.1 Rules for ApplicationDataType of CategoryValueBlock

tbd

7.1.3.4 ApplicationDataType_String

tbd

7.1.3.4.1 Rules for ApplicationDataType of Category String

Rule Data_130:Removed

Instruction

Status: removed

7.1.4 Complex Application DataType

The Primitive *ApplicationDataType* can be used as the basis for complex *ApplicationDataType*s.

7.1.4.1 ApplicationDataType of Category Array

For describing the type of an array you shall define an *ApplicationArrayType*. An *ApplicationArrayType* 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 *ARRAYS* or *ARRAY* of *STRUCTURES*.

05

Surprisingly the array size has to be described as property of the child, using *MaxNumberOfElements*.

10

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 *ApplicationArrayType* are applicable for whole Array

15

20

25

30

35

40

45

50

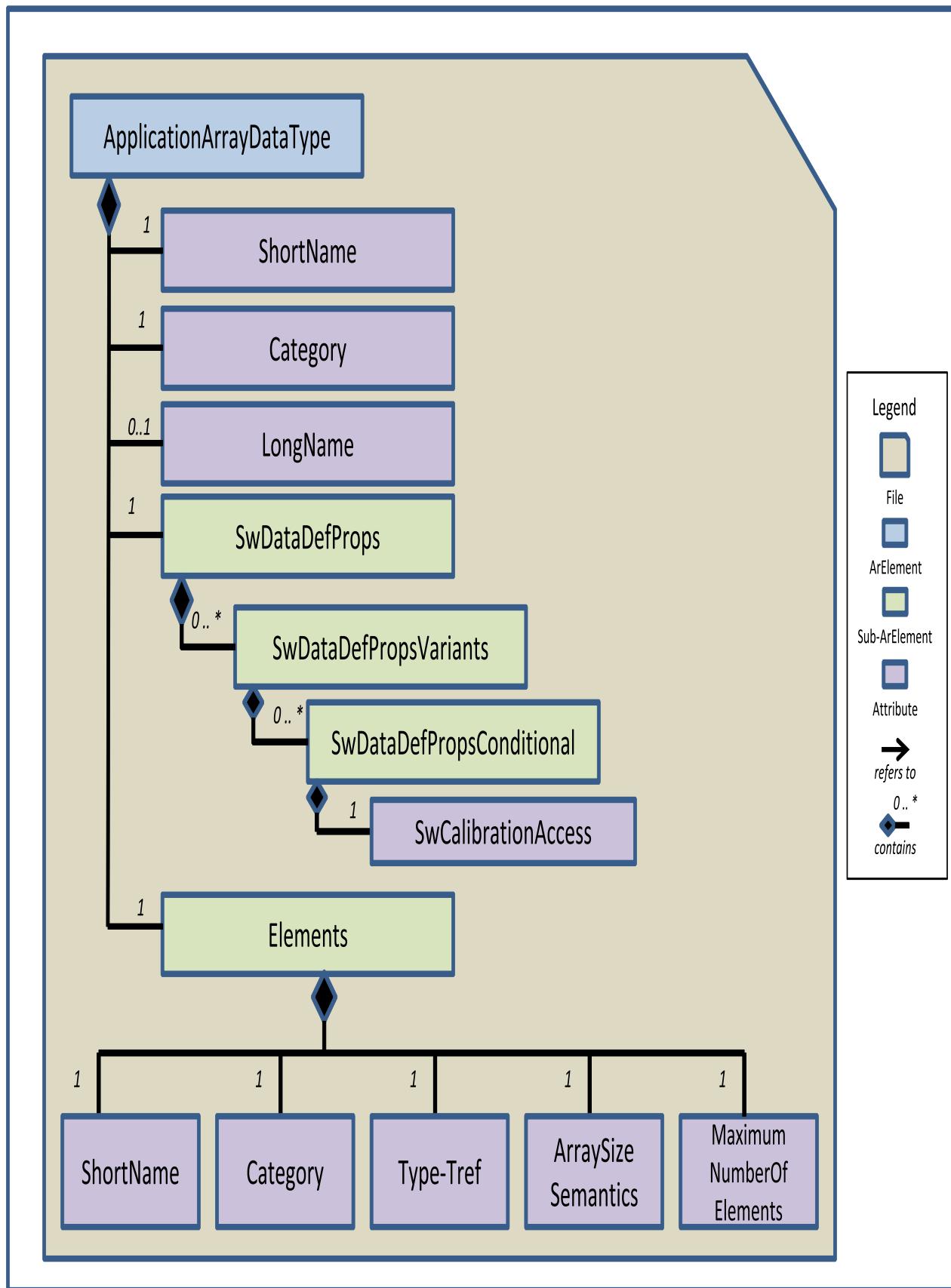
55

60

65

70

Figure 40 Schematic Representation of ApplicationArrayType

**Example:**

```

<APPLICATION-ARRAY-DATA-TYPE>
  <SHORT-NAME>ApplicationDataType_Array</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>

```

```

05 <SW-DATA-DEF-PROPS>
10   <SW-DATA-DEF-PROPS-VARIANTS>
15     <SW-DATA-DEF-PROPS-CONDITIONAL>
20       <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
25     </SW-DATA-DEF-PROPS-CONDITIONAL>
30   </SW-DATA-DEF-PROPS-VARIANTS>
35 </SW-DATA-DEF-PROPS>
40 <ELEMENT>
45   <SHORT-NAME>ApplicationDataType_Array_Element</SHORT-NAME>
50   <CATEGORY>VALUE</CATEGORY>
55     <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
60       <!--Refer to Application Primitive Data Type -->
65     </TREF>
70   <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
75   <MAX-NUMBER-OF-ELEMENTS>3</MAX-NUMBER-OF-ELEMENTS>
80 </ELEMENT>
85 </APPLICATION-ARRAY-DATA-TYPE>

```

Note:

CompuMethods are explained in [\[CompuMethods\]](#).

DataConstraints are explained in [\[DataConstr\]](#).

7.1.4.1.1 Rules for ApplicationDataType of Category Array

Rule Data_140: Defining Array Size

Instruction *MaxNumberOfElements* describes the size of ARRAY, as such the attribute *MaxNumberOfElements* shall be defined for *ApplicationArrayType*

ArraySizeSemantics should not be defined in the *ApplicationArrayType* (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.

Rule Data_176: Defining LongNames, Annotations and Description of an Array

Instruction *LongName*, *Annotations* and *Descr* shall be defined at the parent level in case of simple Array and nested Array.

In case of an Array & Structure combination, *LongNames*, shall be defined at Parent Element and may also be defined at *ArrayElement* level.

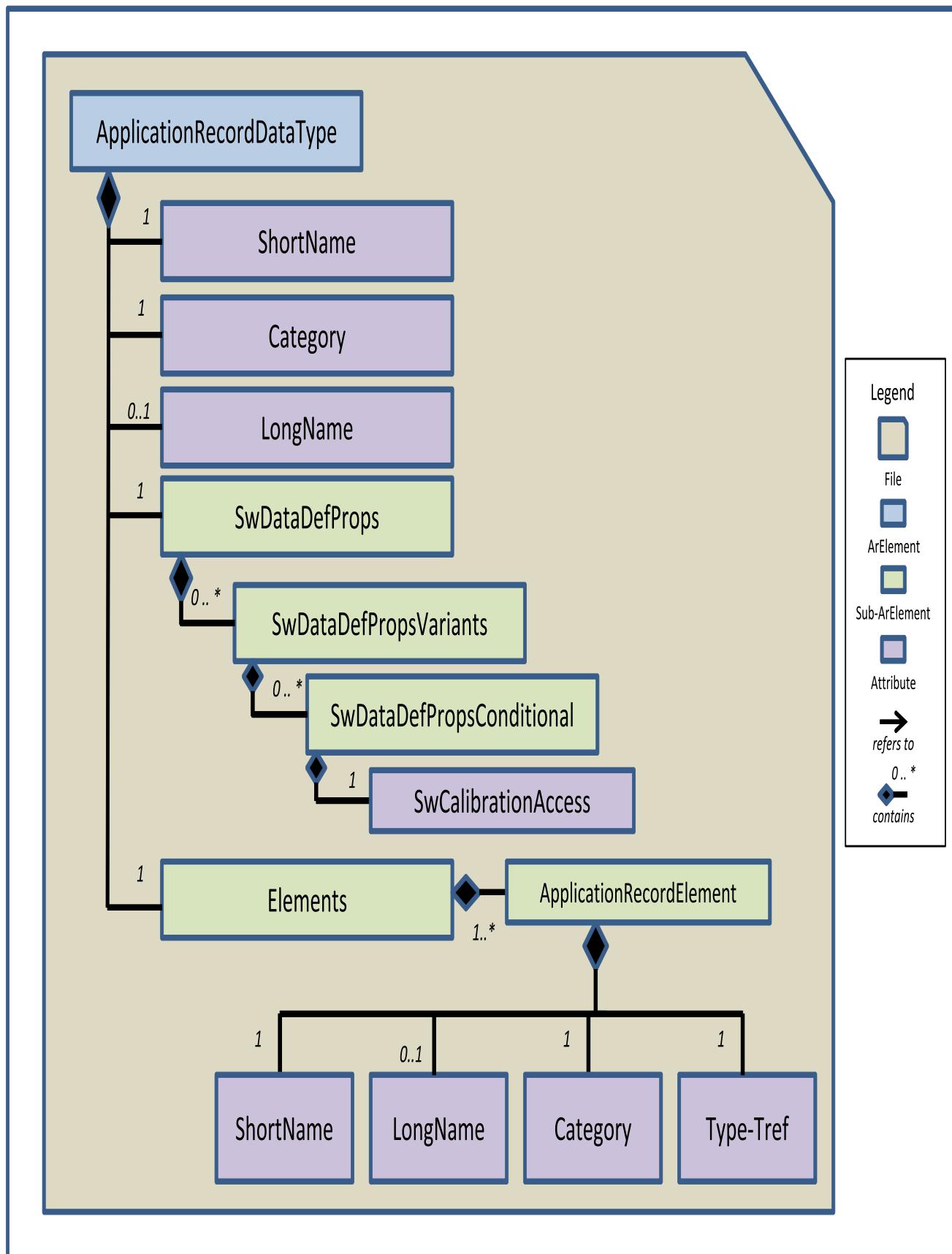
Hint During A2L Generation, for simple Array and nested Array, *LongNames*, *Annotations* and *Descr* are considered from Parent Element only. However for Array & Structure combination, *LongNames* of Parent element is appended with *LongNames* of child element.

7.1.4.2 ApplicationDataType of Category 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 41 Schematic Representation of ApplicationDataType_Struct

**Example:**

```
<APPLICATION-RECORD-DATA-TYPE>
<SHORT-NAME>ApplicationDataType_Struct</SHORT-NAME>
<CATEGORY>STRUCTURE</CATEGORY>
```

```

05   <SW-DATA-DEF-PROPS>
06     <SW-DATA-DEF-PROPS-VARIANTS>
07       <SW-DATA-DEF-PROPS-CONDITIONAL>
08         <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
09       </SW-DATA-DEF-PROPS-CONDITIONAL>
10     </SW-DATA-DEF-PROPS-VARIANTS>
11   </SW-DATA-DEF-PROPS>
12   <ELEMENTS>
13     <APPLICATION-RECORD-ELEMENT>
14       <SHORT-NAME>ApplicationRecordElement_01</SHORT-NAME>
15       <CATEGORY>VALUE</CATEGORY>
16       <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
17         <!-- Refer to the Snippet of an ApplicationPrimitiveDataType -->
18       </TYPE-TREF>
19     </APPLICATION-RECORD-ELEMENT>
20   </ELEMENTS>
21 </APPLICATION-RECORD-DATA-TYPE>

```

Note:

CompuMethods are explained in [\[CompuMethods\]](#)

25 **Hint** Structures can be nested. Nesting shall not only contain Primitive ApplicationDataTypes but also Complex ApplicationDataTypes. In the following example, an ApplicationDataType of Category Structure contains three Sub-Elements: a Value element , a Structure element and an Array element.

30 Value element refers to a primitive *ApplicationDataType*, where as Structure subelement and Array subelement refers to *ApplicationRecordDataType* and *ApplicationRecordDataType* respectively.

```

35 <APPLICATION-RECORD-DATA-TYPE>
36   <SHORT-NAME>ApplicationDataType_NestedStruct</SHORT-NAME>
37   <CATEGORY>STRUCTURE</CATEGORY>
38   <SW-DATA-DEF-PROPS>
39     <SW-DATA-DEF-PROPS-VARIANTS>
40       <SW-DATA-DEF-PROPS-CONDITIONAL>
41         <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
42       </SW-DATA-DEF-PROPS-CONDITIONAL>
43     </SW-DATA-DEF-PROPS-VARIANTS>
44   </SW-DATA-DEF-PROPS>
45   <ELEMENTS>
46     <APPLICATION-RECORD-ELEMENT>
47       <SHORT-NAME>ApplicationRecordElement_01</SHORT-NAME>
48       <CATEGORY>VALUE</CATEGORY>
49       <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
50         <!-- Refer to the Snippet of an ApplicationPrimitiveDataType -->
51       </TYPE-TREF>
52     </APPLICATION-RECORD-ELEMENT>
53     <APPLICATION-RECORD-ELEMENT>
54       <SHORT-NAME>ApplicationRecordElement_02</SHORT-NAME>
55       <CATEGORY>STRUCTURE</CATEGORY>
56       <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
57         <!-- Refer to the Snippet of an ApplicationRecordDataType -->
58       </TYPE-TREF>
59     </APPLICATION-RECORD-ELEMENT>
60     <APPLICATION-RECORD-ELEMENT>
61       <SHORT-NAME>rba_Runtime_Cores</SHORT-NAME>
62       <CATEGORY>ARRAY</CATEGORY>
63       <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
64         <!-- Refer to the snippet of an ApplicationArrayType-->
65       </TYPE-TREF>
66     </APPLICATION-RECORD-ELEMENT>
67   </ELEMENTS>
68 </APPLICATION-RECORD-DATA-TYPE>

```

7.1.4.2.1 Rules for ApplicationDataType of Category Structure

Rule Data_123: Removed

Instruction

Status: removed

Rule Data_141: Definition of SwDataDefProps at SubElement Level

Instruction The *SwDataDefProps* of the *Elements* of an *ApplicationArrayType* or *ApplicationRecordType* 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*.

7.1.5 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.

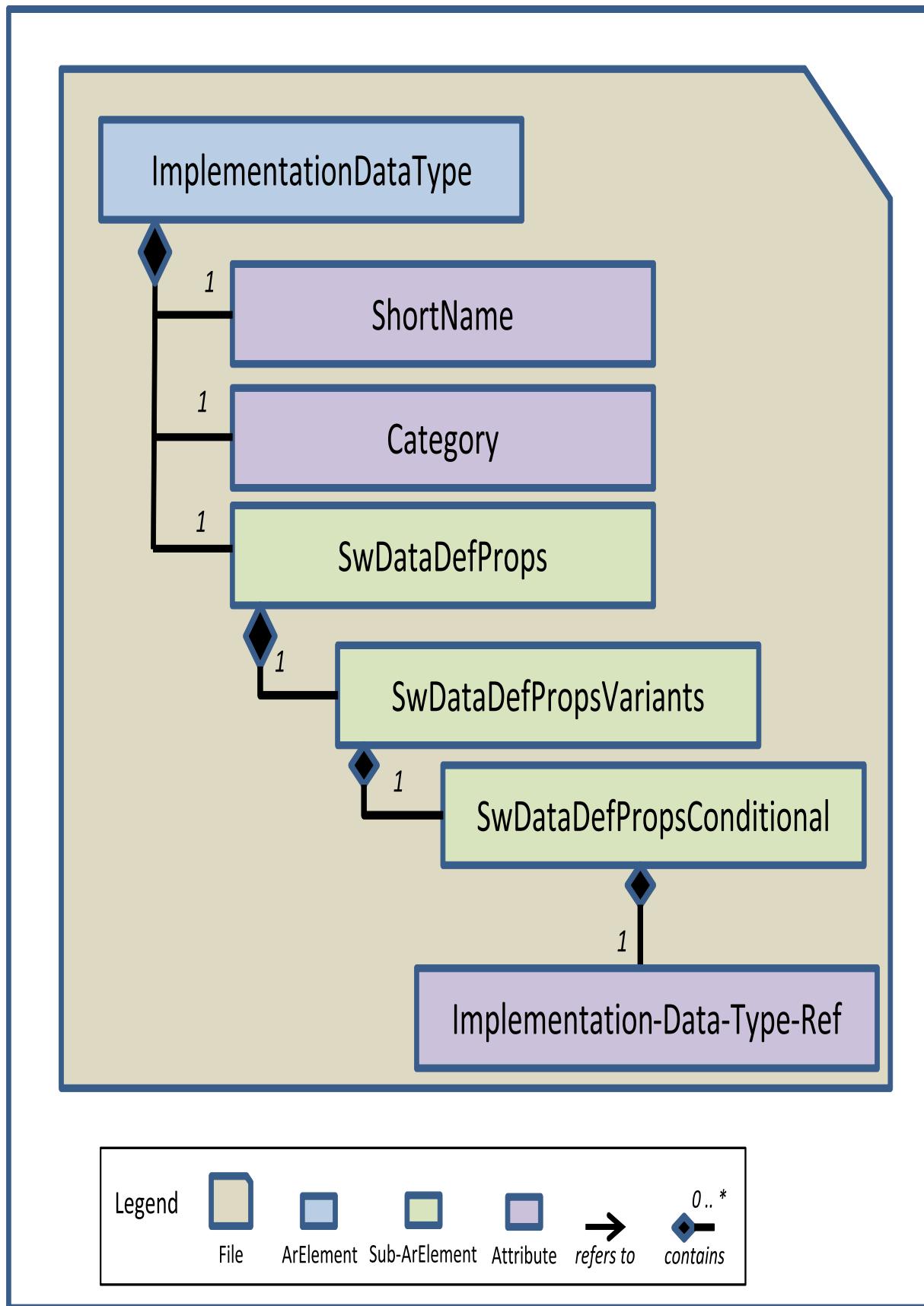
There are several use cases for this level in AUTOSAR:

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. Primitive *ImplementationDataTypes* applicable for both ASW and BSW softwares:
 - VALUE
2. Composite *ImplementationDataTypes*
 - ARRAY
 - STRUCTURE

Figure 42 Schematic Representation of ImplementationDataType



7.1.5.1 Rules for ImplementationDataType

Rule Data_004b: Where to Define *ImplementationDataType*

Instruction

1. Primitive Implementation data types (e.g. sint16) shall not be defined locally as they are always central and Standardized and delivered as Central Elements.
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 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 (for enums) shall be provided as central elements if they are product line specific, or they shall be defined module specific.

Rule Data_050: Reuse of *ImplementationDataTypes*

Instruction 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.

7.1.5.2 ImplementationDataType of Category Value

Example:

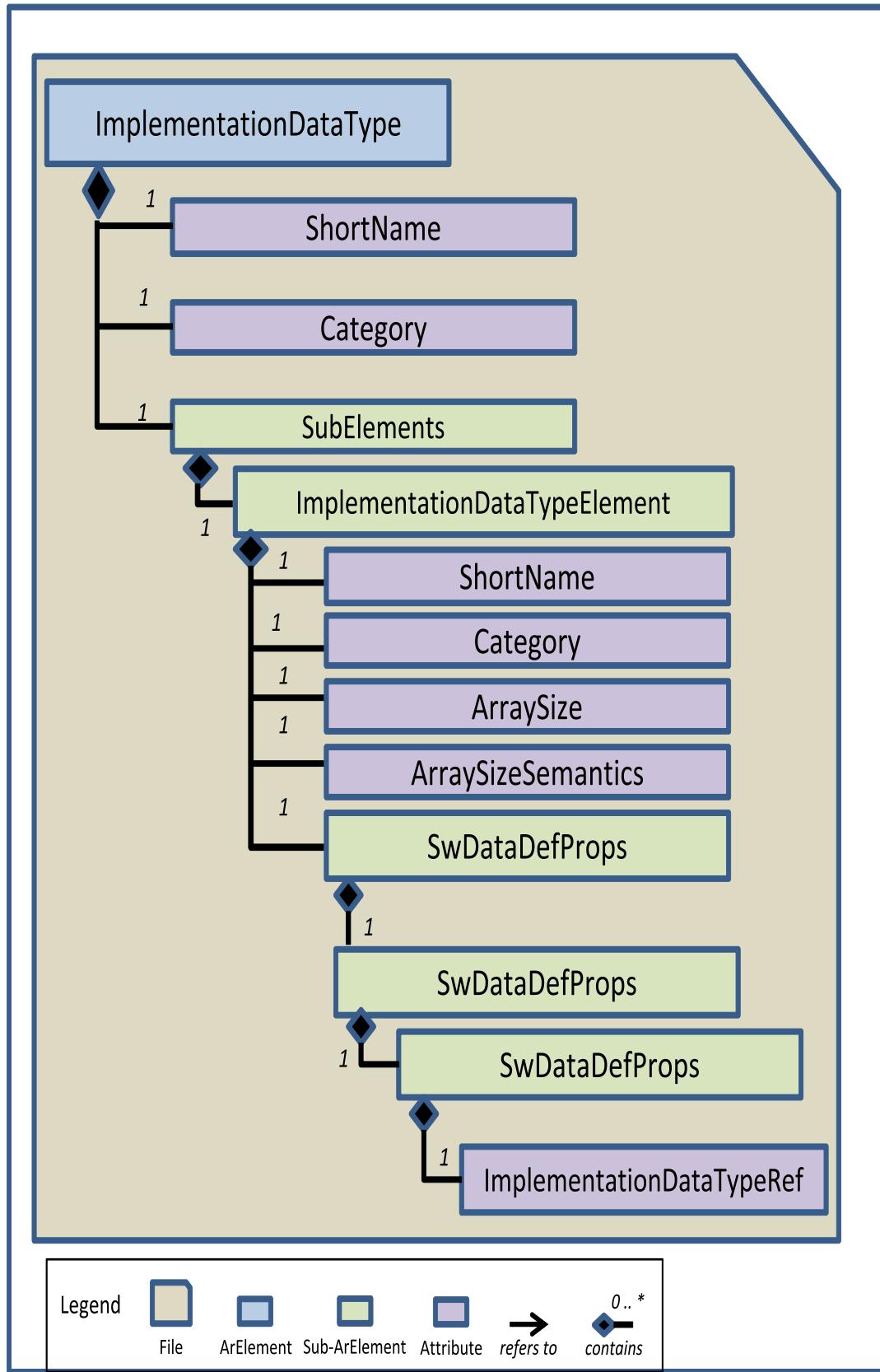
```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>ImplementationDataType_Value</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
          <!-- Refer 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>
```

7.1.5.2.1 Rules for ImplementationDataType of Category Value

tbd

7.1.5.3 ImplementationDataType of Category Array

Figure 43 ImplementationDataType_Array


Example:

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>ImplementationDataType_Array</SHORT-NAME>
```

```

05 <CATEGORY>ARRAY</CATEGORY>
06 <SUB-ELEMENTS>
07   <IMPLEMENTATION-DATA-TYPE-ELEMENT>
08     <SHORT-NAME>ImplementationDataType_Element</SHORT-NAME>
09     <CATEGORY>VALUE</CATEGORY>
10     <ARRAY-SIZE>50</ARRAY-SIZE>
11     <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
12     <SW-DATA-DEF-PROPS>
13       <SW-DATA-DEF-PROPS-VARIANTS>
14         <SW-DATA-DEF-PROPS-CONDITIONAL>
15           <BASE-TYPE-REF DEST="SW-BASE-TYPE">
16             <!-- Refer to a BaseType -->
17           </BASE-TYPE-REF>
18         </SW-DATA-DEF-PROPS-CONDITIONAL>
19       </SW-DATA-DEF-PROPS-VARIANTS>
20     </SW-DATA-DEF-PROPS>
21   </IMPLEMENTATION-DATA-TYPE-ELEMENT>
22 </SUB-ELEMENTS>
23 </IMPLEMENTATION-DATA-TYPE>

```

Note:

25 SwRecordLayouts are explained in [\[Rec_Layout\]](#)

7.1.5.3.1 Rules for ImplementationDataType of Category Array

30 Rule Data_139: Usage of ArraySize and ArraySizeSemantics

Instruction *ImplementationDataTypes* shall have *ArraySize* defined in *ImplementationDataTypeElement* which defines the size of Array.

35 *ArraySizeSemantics* shall be FIXED-SIZE.

40

45

50

55

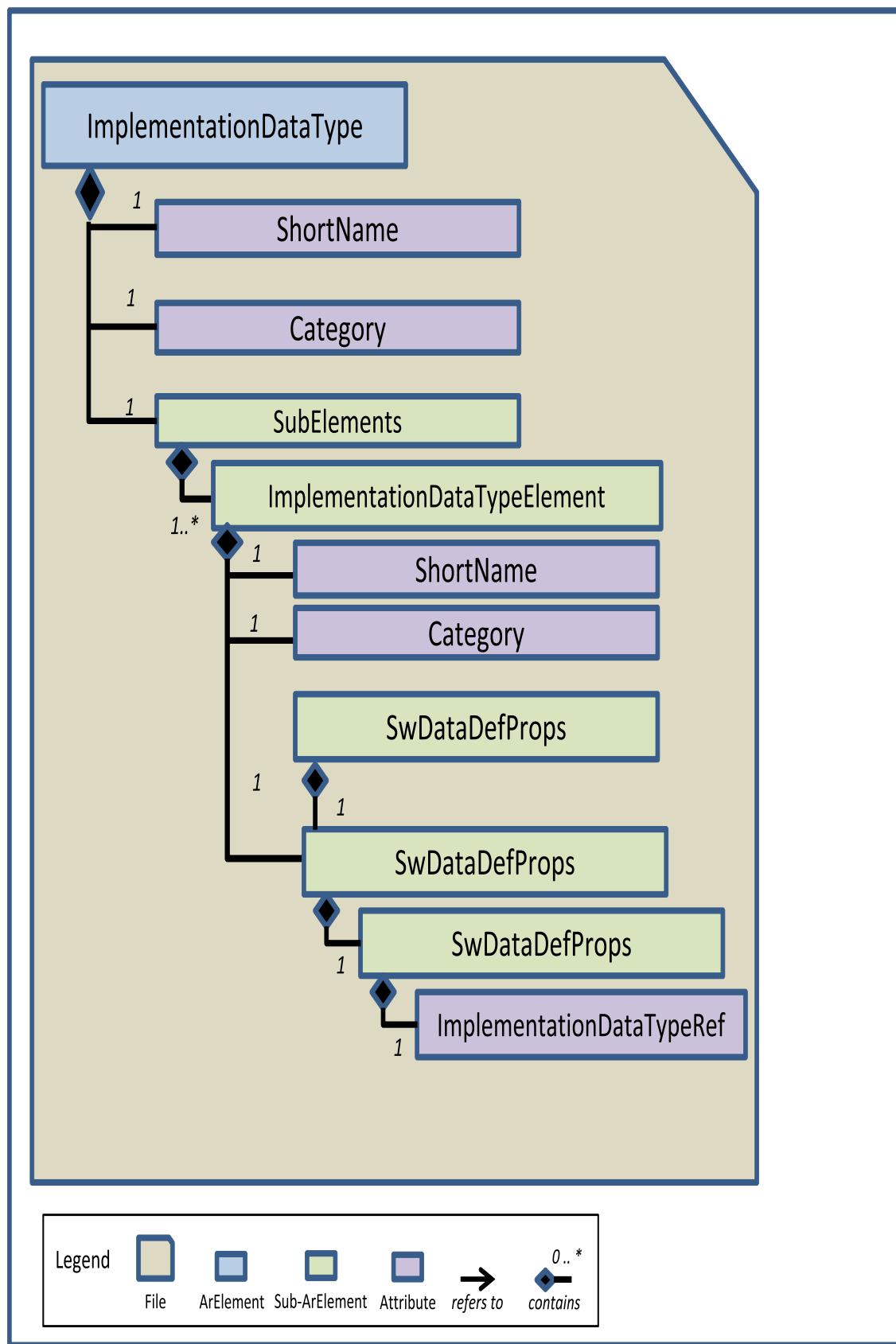
60

65

70

7.1.5.4 ImplementationDataType of Category Structure

Figure 44 ImplementationDataType_Struct



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 a ImplementationDataType of Category Value -->
              <!-- Snippet for ImplementationDataType of Category Value-->
              [ImplDataType_Val]
            </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>

```

Categories of ImplementationDataType

ImplementationDataType or the aggregated *ImplementationDataTypeElement* do not form closed set. They always have to refer to further type definitions. Depending upon the referenced type definition, *ImplementationDataType* will have the following categories:

- ▶ If the referenced type definition is *SwBaseType*, then the category corresponds to VALUE.
- ▶ If the referenced type definition is another *ImplementationDataType*, then the category corresponds to TYPE_REFERENCE.
- ▶ If the referenced type definition is *BswModuleEntry* in *SwPointerTargetProps*, then the category corresponds to FUNCTION_REFERENCE.
- ▶ If the referenced type definition is *SwDataDefProps* in *SwPointerTargetProps*, then the category corresponds to DATA_REFERENCE.

Hint This topic can be referred in [\[TPS_SWCT\] TPS_SWCT_01257](#).

7.1.5.4.1 Rules for ImplementationDataType of Category Structure

tbd

7.1.6 Data Type Mapping

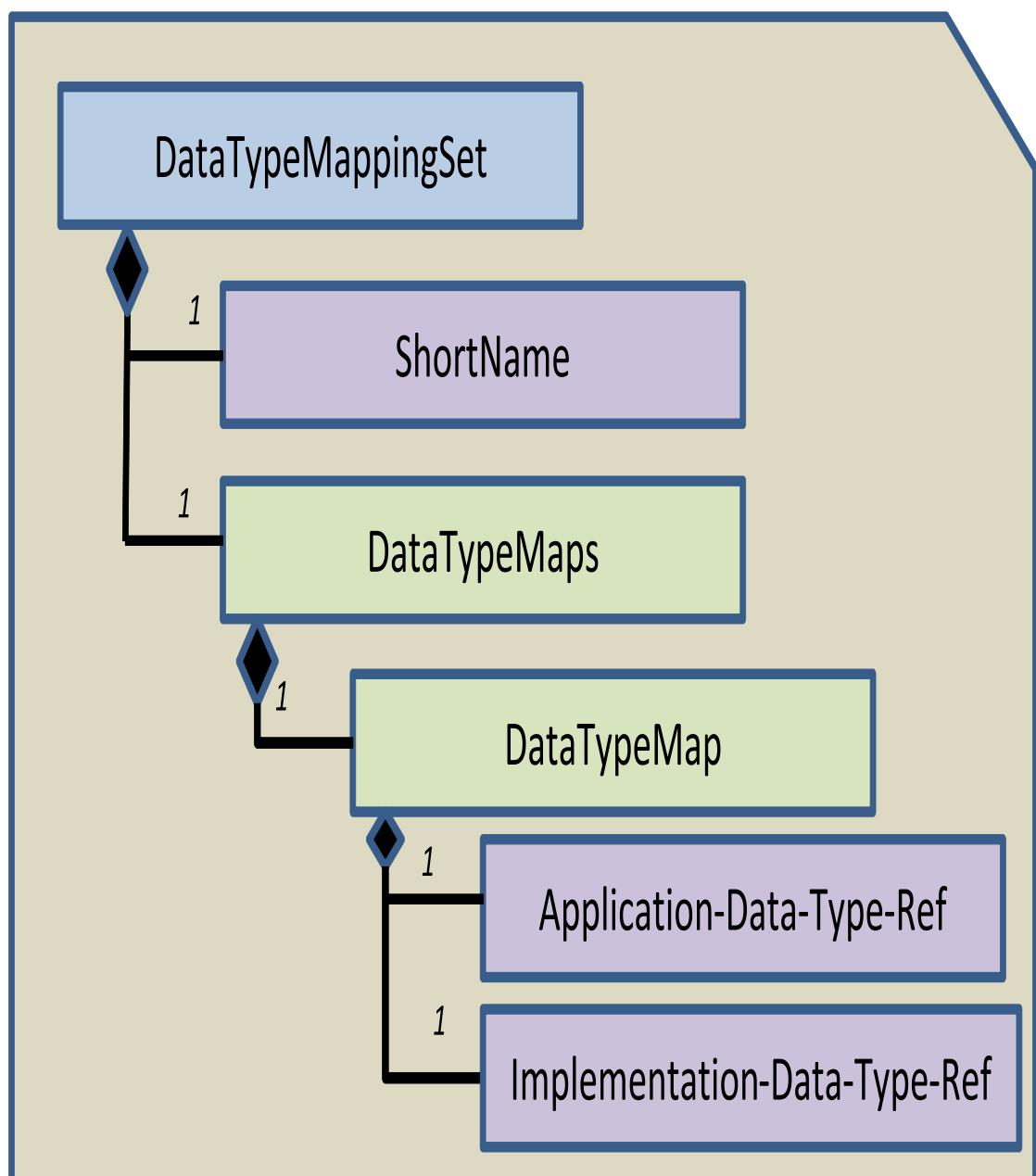
This class represents the relationship between *ApplicationDataType* and its implementing *ImplementationDataType*.

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 45 Schematic Representation of Data Type Mapping



DataTypeMappingSet

Example:

```
<DATA-TYPE-MAPPING-SET>
```

```

05 <SHORT-NAME>DataTypeMappingSet</SHORT-NAME>
<DATA-TYPE-MAPS>
10   <DATA-TYPE-MAP>
    <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      <!-- Refer to a ApplicationPrimitiveDataType -->
    </APPLICATION-DATA-TYPE-REF>
    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
      <!-- Refer to a ImplementationDataType -->
    </IMPLEMENTATION-DATA-TYPE-REF>
15   </DATA-TYPE-MAP>
   <DATA-TYPE-MAP>
    <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      <!-- Refer to a ApplicationPrimitiveDataType -->
    </APPLICATION-DATA-TYPE-REF>
    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
20      <!-- Refer to a ImplementationDataType -->
    </IMPLEMENTATION-DATA-TYPE-REF>
   </DATA-TYPE-MAP>
 </DATA-TYPE-MAPS>
</DATA-TYPE-MAPPING-SET>

```

25 **Note:** ApplicationPrimitiveDataTypes are explained in [\[AppI.DataTypeValue\]](#)

ImplementationDataTypes are explained in [\[ImplI.DataType\]](#)

30 7.1.6.1 Rules for DataTypeMappingSet

Rule Data_127: Where to Define DataTypeMappingSet

Instruction

35 **DerivedFrom:** CEL_064

DataTypeMappingSet may be defined centrally or decentral.

- 40 ▶ If the referenced ApplicationDataType and the referenced *ImplementationDataType* are defined centrally, then DataTypeMappingSet is also defined centrally.
- ▶ Mapping between ApplicationDataType and *ImplementationDataType* shall be defined locally (Component/ Module Specific Mapping) if the ApplicationDataType and *ImplementationDataType* are also defined locally.

45 Rule Data_129: Mapping Between ApplicationDataTypes and ImplementationDataTypes

Instruction Every ApplicationDataType shall be mapped to an *ImplementationDataType* through a *DataTypeMapping*.

50 For all ApplicationDataTypes defined by a component corresponding mapping to *ImplementationDataType* shall also be specified in the same component.

55 For ApplicationDataTypes defined as central elements, their corresponding mapping to an *ImplementationDataType* will also be defined as central elements.

7.2 DataPrototypes

7.2.1 Introduction

Data prototypes implement a role of data type. This means to say a data prototype is also a data type defined within 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 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 \[CalParam\]](#)

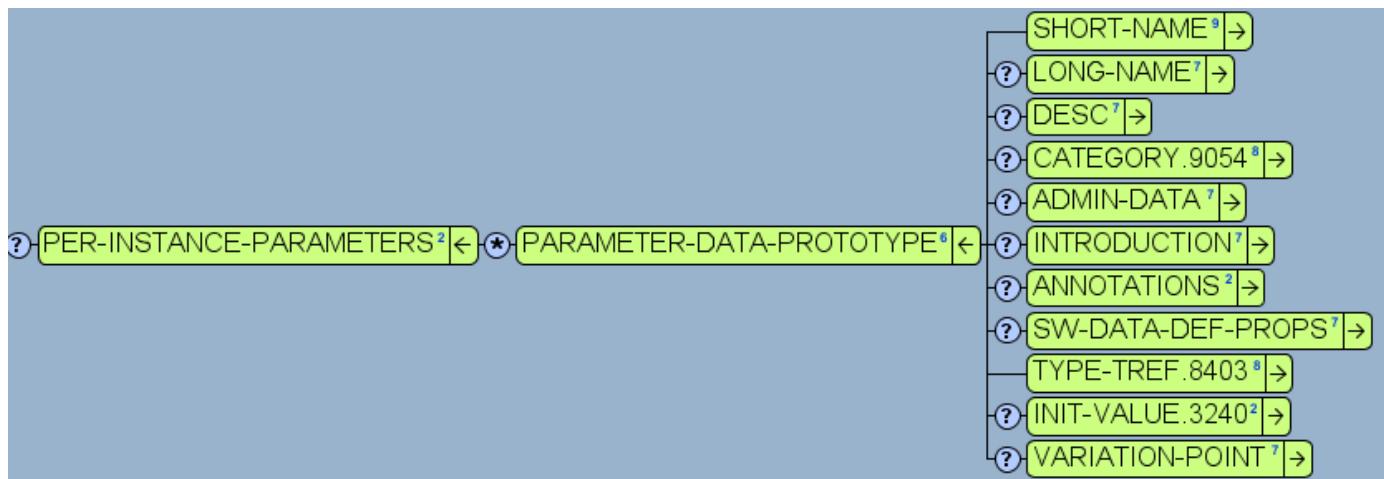
7.2.2 ParameterDataPrototype (Calibration Parameters)

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 46 Schematic Representation of Parameter Data Prototype



7.2.2.1 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

- 05
-
- ▶ NOT-ACCESSIBLE

Table 35 swCalibrationAccess Values

10

Literal	Description
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file
readOnly	The element will only appear as read-only in an ASAP file.
readWrite	The element will appear in ASAP file with both read and write access.

15

7.2.2.1.1 Rules for SwCalibAccess

20

Rule Data_040: Instantiating a ParameterDataPrototype with SwCalibrationAccess and SwImplPolicy

25

Instruction A calibration parameter is instantiated with a ParameterDataPrototype class that aggregates a SwDataDefProps with properties

- 30
-
- ▶ *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.*

35

Hint The DataSpec SubTWG have to define, where and with which attribute the SwCalibrationAccess should be defined

40

Rule Data_143: Calibration Access for Measurements

45

Instruction

Measurement object shall only have READ-ONLY as Calibration Access

50

7.2.2.2 Usage of SW-IMPL-POLICY

55

Hint This section is yet to be reviewed.

SwImplPolicy defines how a data element needs to be processed at the receiver's end. These attributes will take enum literals which are defined by AUTOSAR as their values.

60

SwImplPolicyEnum at ParameterDataPrototype:

The following table defines SwImplPolicy that are available for ParameterDataPrototype

65
Table 36 SwImplPolicy for ParameterDataPrototype

Enum Literal	Description
Const	The value of the data element is set once, but never changed after that. (e.g. Const Int x = 5;). This requires memory allocation in ECU
Fixed	The value of the data element is a pre-defined universal constant which can never be changed (e.g. pi = 3.14, gravity of earth "g" = 9.81). There is no memory allocation in ECU
Standard	This is applicable for all data elements

70

SwImplPolicyEnum at VariableDataPrototype:

The following table defines *SwImplPolicy* that are available for *VariableDataPrototype*

Table 37 SwImplPolicy for VariableDataPrototype

Enum Literal	Description
MeasurementPoint	The data element is created for measurement purposes only. The data element is never read directly within the ECU software. In contrast to a "standard" data element in an unconnected provide port is, this unconnection is guaranteed for Measurement Point data elements.
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

However developers are not supposed to use *SwImplPolicy* for data prototype as there are no proper usecases available.

7.2.2.2.1 Rules for *SwImplPolicy*

Rule Data_170: Handling of *SwImplPolicy* attribute at *ParameterDataPrototype*

Instruction

DerivedFrom: TPS_SWCT_02000

SwImplPolicy shall not be defined for *ParameterDataPrototype*

Hint If the attribute *swImplPolicy* is not explicitly set, then the default value "Standard" is applied.

Rule Data_171: Handling of *SwImplPolicy* attribute at *VariableDataPrototype*

Instruction

DerivedFrom: TPS_SWCT_02000

SwImplPolicy shall not be defined for *VariableDataPrototype*

Hint If the attribute *swImplPolicy* is not explicitly set, then the default value "Standard" is applied.

7.2.3 ParameterDataPrototype applicable for both ASW and BSW

7.2.3.1 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>
```

Note: ApplicationPrimitiveDataTypes are explained in [\[AppIDataTypeValue\]](#)

7.2.3.1.1 Rules for ParameterDataPrototype of Category Value

Instruction 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 ComponentPrototypes of the same ComponentType.

In case of PerInstance Parameters, calibration value in the ParameterDataPrototype will be specific for each instance of ComponentPrototypes of the same ComponentType.

Example:

```
<APPLICATION-SOFTWARE-COMPONENT-TYPE>
  <SHORT-NAME><!--Short Name for Appln Software Component --></SHORT-NAME>
  <PORTS>
    <P-PORT-PROTOTYPE>
    ...
    </P-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIOR>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME><!--Short Name for InternalBehavior --></SHORT-NAME>
      ...
      <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>
              <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>
        </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>
  <PORTS>
    <P-PORT-PROTOTYPE>
    ...
    </P-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIOR>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME><!--Short Name for InternalBehavior --></SHORT-NAME>
      ...
      <PER-INSTANCE-PARAMETERS>
        <PARAMETER-DATA-PROTOTYPE>
          <SHORT-NAME><!--Short Name for PDP --></SHORT-NAME>
          <CATEGORY>CURVE</CATEGORY>
          <SW-DATA-DEF-PROPS>
```

```

05   <SW-DATA-DEF-PROPS-VARIANTS>
10     <SW-DATA-DEF-PROPS-CONDITIONAL>
15       <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>
 </PER-INSTANCE-PARAMETERS>
</PARAMETER-DATA-PROTOTYPE>
...
</SWC-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIOR>
<APPLICATION-SOFTWARE-COMPONENT-TYPE>

```

Rule Data_010: Removed

Rule Data_011: Sharing Calibration Parameters between instances of different " SwComponentType "

Instruction

- ▶ For the Calibration parameters to be visible (shared) in other SwComponentTypes, a dedicated ParameterSw-ComponentType has to be used.
- ▶ 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>

```

```

05      </PARAMETERS>
</PARAMETER-INTERFACE>

10      <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>

```

20 Rule Data_031: Initial Value for ParameterDataPrototypes

Instruction Every ParameterDataPrototype shall have an initial value specified . This value shall be an implementation based one.

Hint Specification of Values shall refered in *Chapter "Assigning Initial Values for Calibration Parameter" [AssignInitVal-ForCalPrm]*

30 Rule Data_051: Reference to an *ImplementationDataType*

Instruction *ImplementationDataType* shall not to be referenced directly by a *DataProtoType*, but a *DataPrototype* shall refer to an *ApplicationDataType* which in turn is mapped to an *ImplementationDataType*

40 Rule Data_051a: Reference to an *ImplementationDataType*

Instruction

Scope:BSW

ImplementationDataType shall be referenced directly by a *DataProtoType*, if and only if the *DataProtoType* cannot be an A2L object.⁷

Hint If a *DataProtoType* directly refer to an *ImplementationDataType*, then such *ImplementationDataType* shall refer to a *CompuMethod* of Category *TextTable*.

50 Rule Data_174: Categories of *DataPrototype* and corresponding *ApplicationDataType* shall be same

Instruction A *DataPrototype* shall have same category as the *ApplicationDataType* to which it is referring.

55 Rule Data_178: Usage of *Constant Memorys* are not allowed for specifying *CalParams*

Instruction RTE-Generator shall not create any C-Code for *Constant Memorys*. As such neither in ASW nor in BSW *Constant Memorys* shall be used.

⁷ A *DataPrototype* that appear in an A2L file is termed as A2L object.

05 Though AUTOSAR allows the usage of *Constant Memorys* (TPS_SWCT_01483 [[TPS_SWCT](#)]), developers are not supposed to use it. Instead of *Constant Memorys*, AR-TYPED-PER-INSTANCE-MEMORYs or *sharedParameter* or *perInstanceParameter* may be used.

10 7.2.4 Complex ParameterDataPrototype

7.2.4.1 ParameterDataPrototype of Category Array

15 **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>
```

20 **Note:**

25 ApplicationArrayDataTypes are explained in [[ApplDataType_Array](#)]

30 35 40 45 50 55 60 65 70 7.2.4.1.1 Rules for ParameterDataPrototype of Category Array

tbd

7.2.4.2 ParameterDataPrototype of Category Structure

45 **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>
```

Note:

ApplicationRecordDataTypes are explained in [[ApplDataType_Structure](#)]

7.2.4.2.1 Rules for ParameterDataPrototype of Category Structure

tbd

7.2.5 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).

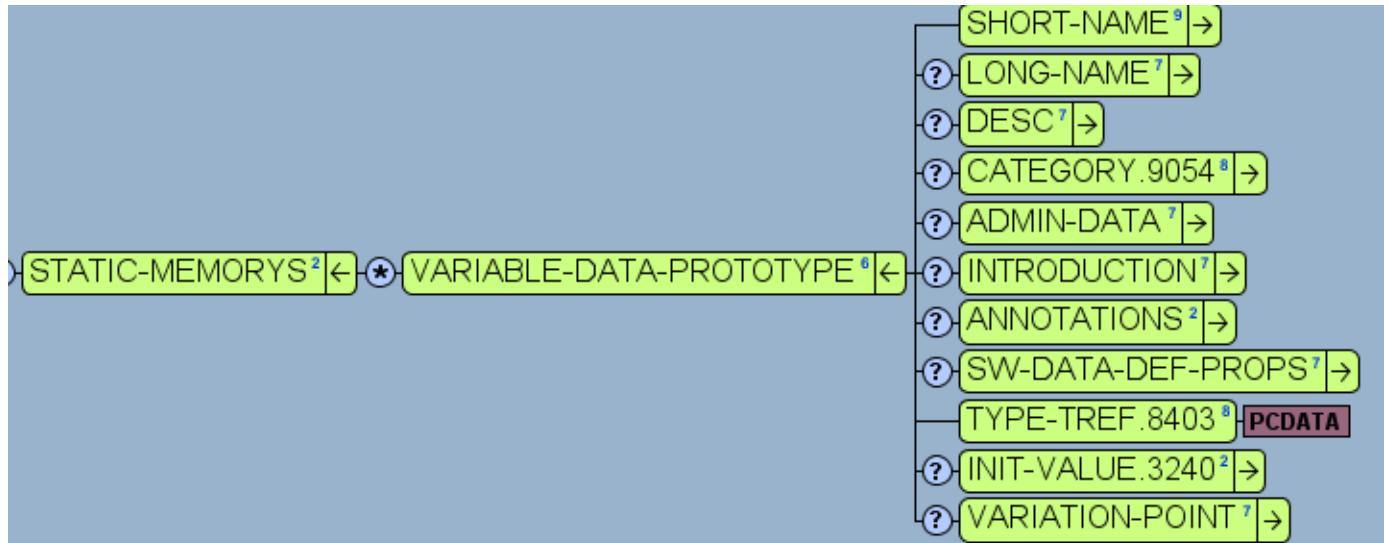
AUTOSAR Variable

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. nramData mapping)
2. As an element inside of a composite variable provided via Port and typed by ImplementationDatatype (e.g. inputValue for a curve)

Figure 47 Schematic Representation of Variable Data Prototype



7.2.6 VariableDataPrototype applicable for both ASW and BSW of Category Value

7.2.6.1 VariableDataPrototype of Category Value

Example:

```
<VARIABLE-DATA-PROTOTYPE>
  <SHORT-NAME>VariableDataPrototype_Value</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
```

```

05   <SW-DATA-DEF-PROPS-VARIANTS>
10     <SW-DATA-DEF-PROPS-CONDITIONAL>
15       <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>

```

Note: ApplicationPrimitiveDataTypes are explained in [\[ApplDataTypeValue\]](#)

7.2.6.1.1 Rules for VariableDataPrototype of Category Value

Rule Data_179: Possibility to define *VariableDataPrototypes* in ASW and BSW

Instruction

- ▶ In ASW, *VariableDataPrototypes* shall defined in the role of *EXPLICIT-INTER-RUNNABLE-VARIABLE* and *IMPLICIT--INTER-RUNNABLE-VARIABLE*.
- ▶ In BSW, *VariableDataPrototypes* shall be defined in the role of *Static Memorys*.

AUTOSAR allows to define *VariableDataPrototypes* in the role of *Static Memorys* (([\[TPS_SWCT\]](#) TPS_SWCT_01483) for ASW as well. However developers are not supposed to use it due to tooling constraint. Instead of *Static Memorys*, *EXPLICIT-INTER-RUNNABLE-VARIABLE*, *IMPLICIT-INTER-RUNNABLE-VARIABLE* is used. Snippet for defining *VariableDataPrototype* in ASW can be found in [\[G_ArCaDe\]](#)

7.2.6.2 VariableDataPrototype of Category Boolean

A *VariableDataPrototype* of Category Boolean can contain one boolean state. This boolean state shall occupy one complete byte or a word as direct addressing of single bits may not be available.

Example:

```

45   <VARIABLE-DATA-PROTOTYPE>
50     <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>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
        <!-- Refer to an ApplicationDataType of Category Boolean -->
      </TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>

```

Note: ApplicationDataTypes are explained in [\[ApplDataTypeBool\]](#)

7.2.6.2.1 Rules for VariableDataPrototype of Category Boolean

tbd

7.2.7 Complex VariableDataPrototype

7.2.7.1 VariableDataPrototype of Category Array

Example:

```

<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>
```

Note:

ApplicationArrayDataTypes are explained in [\[ApplDataType_Array\]](#)

7.2.7.1.1 Rules for VariableDataPrototype of Category Array

tbd

7.2.7.2 VariableDataPrototype of Category Structure

Example:

```

<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>
```

Note:

ApplicationRecordDataTypes are explained in [\[ApplDataType_Structure\]](#)

05

7.2.7.2.1 Rules for VariableDataPrototype of Category Structure

10
tbd15

7.3 Usage of Attributes of *SwDataDefProps*

20
SwDataDefProps are an important schema element in AUTOSAR. They are applicable for different levels:

- 25
-
- ▶ *ApplicationDataTypes*
 - ▶ *ImplementationDataTypes*
 - ▶ *ParameterDataPrototypes / VariableDataPrototypes*
 - ▶ *ParameterAccessss*
- 20
... and some more.

25
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*".30

7.3.1 Rules for *SwDataDefPropsUsage*

35

Rule Data_135: Usage of Attributes of *SwDataDefProps*

35
Instruction Properties of *SwDataDefProps* shall be defined according to table "*Usage of Attributes of SwDataDefProps*"40
Table 38 Usage of Attributes of *SwDataDefProps*45

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]

Attributes of SwDataDefProps	ApplicationDataType	ImplementationDataType	DataPrototype	ParameterAccess
swCalprmAxisSet.swCalprmAxis/ swAxisIndividual.inputVariableType	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.

Table 39 Not Used Attributes of *SwDataDefProps*

Attributes of SwDataDefProps	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/ SwAxisIndividual.unit	[na]	[na]	[na]	[na]
swCalprmAxisSet.swCalprmAxis/ swAxisGrouped.sharedAxisType	[na]	[na]	[na]	[na]

Attributes of SwDataDefProps	ApplicationDataType	ImplementationDataType	DataPrototype	ParameterAccess
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.

Rule Data_175: Semantics of *displayFormat*

Instruction *displayFormat* shall be of form " %d.pf ", where

- ▶ d represents minimum overall number of digits. This includes flag and decimal point.
- ▶ . represents decimal point
- ▶ p represents the number of digits after decimal point
- ▶ f represents type character for float representation.

Example:

%3.2f

Here the overall number of digits along with the decimal point shall be atleast 3 with 2 digits after decimal point.

So the value will be represented as 1.00

Hint Developer has to design display format so precisely that it do not lead to any rounding off errors.

7.4 Initial Values

Hint This chapter has to be reworked and jira has been created for the same. ARMETH-1448, ARMETH-1106, ARMETH-1017

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.

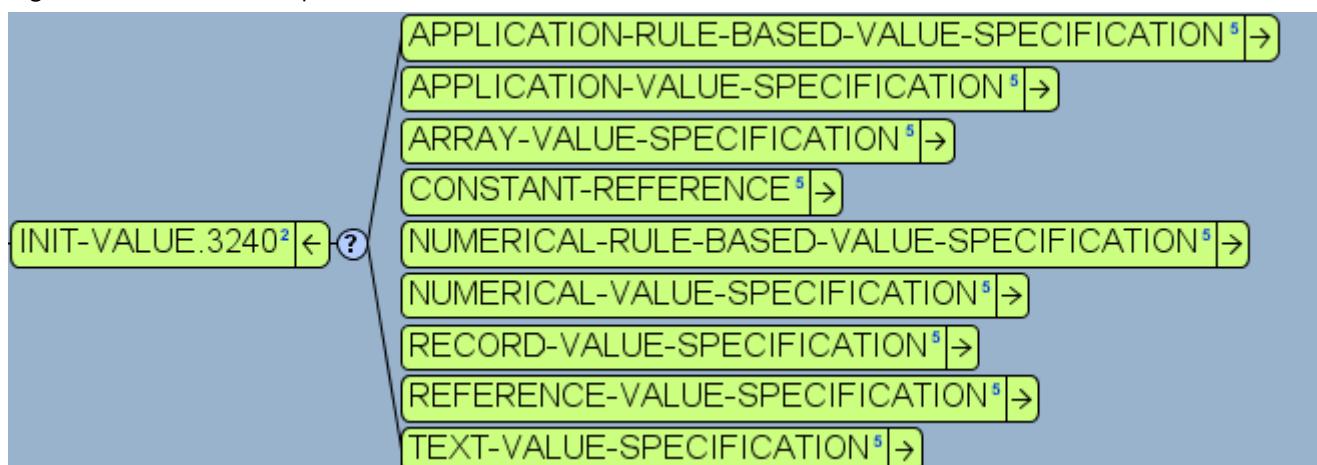
7.4.1.1 Initial Values for Calibration Parameter Overview

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 48 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.

```

10 <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 for InitValues

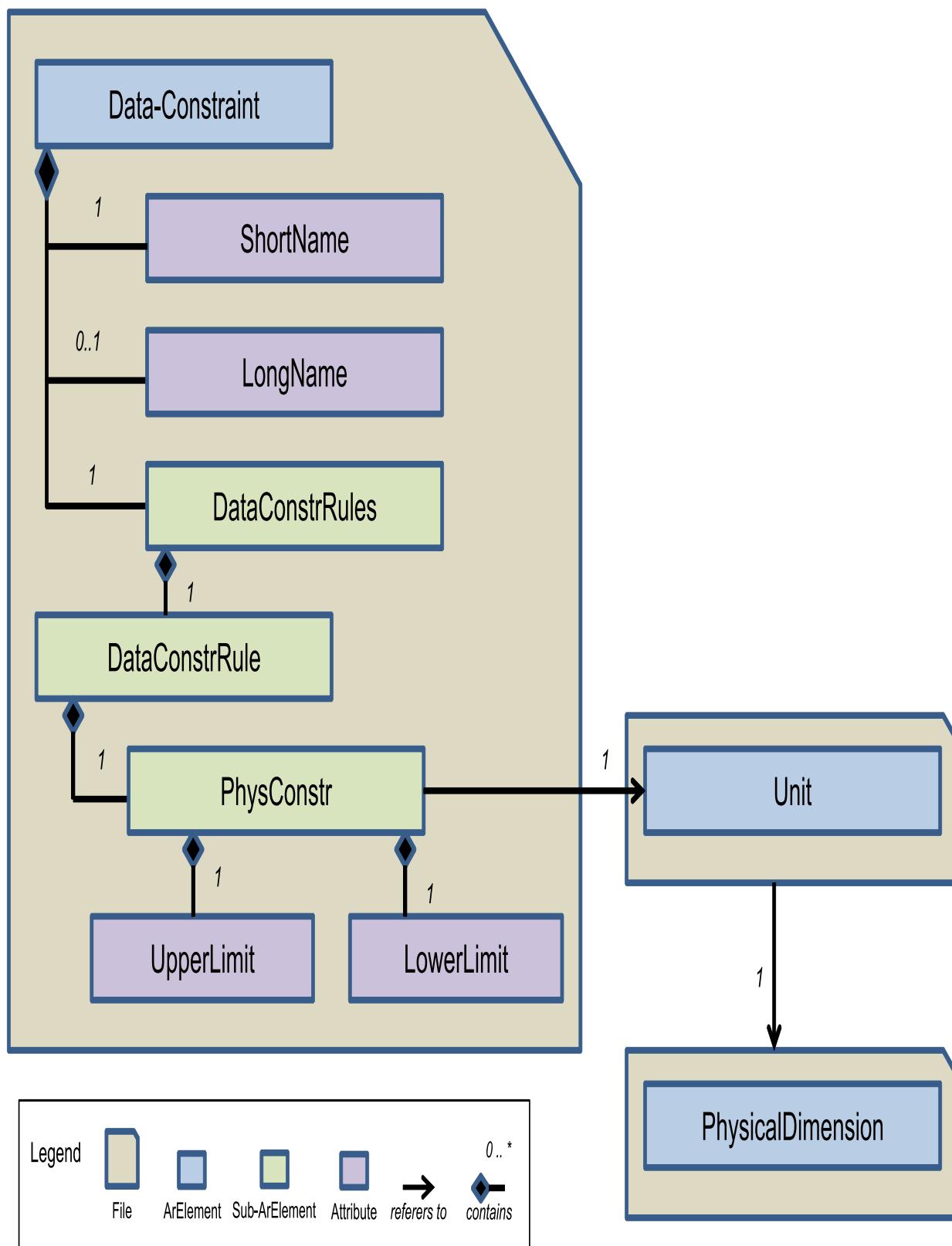
tbd

7.5 Data Constraints

DataConstraints are used to implement limits on the values for data element. These limits are necessary to give the application engineer a sensible range of possible values. The meta-class *DataConstr* is referenced via the attribute *SwDataDefProps* of an *ApplicationDataType*. *DataConstraints* are defined as part of *CentralElements* if the referring *ApplicationDataType* is also defined as *CentralElement* or the developer has to define an application specific *DataConstraint* (locally) if the referring *ApplicationDataType* is defined locally (specific for a concrete application).

A single *DataConstraint* can be reused by several *ApplicationDataTypes*.

Figure 49 Schematic Representation of Data Constraints

**Example:**

```
<AR-PACKAGE>
  <SHORT-NAME>DataConstrs</SHORT-NAME>
  <ELEMENTS>
```

```

05      <DATA-CONSTR>
<SHORT-NAME>DataConstraints</SHORT-NAME>
<LONG-NAME>Long Name for Data Constraint </LONG-NAME>
10     <DATA-CONSTR-RULES>
      <DATA-CONSTR-RULE>
        <PHYS-CONSTRS>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="OPEN">INF</UPPER-LIMIT>
          <UNIT-REF DEST="UNIT"><!-- Refer to an UNIT --></UNIT-REF>
        </PHYS-CONSTRS>
      </DATA-CONSTR-RULE>
    </DATA-CONSTR-RULES>
  </DATA-CONSTR>
</ELEMENTS>
</AR-PACKAGE>

```

20 Note:

Units are explained in [\[Unit_PhysDim\]](#)

25 7.5.1 Types of DataConstraint

A *DataConstraint* can be of following 2 types:

- 30 ▶ Physical Constraints (*physConstr*): This attribute describes the limitations applicable on the physical domain. *DataConstraint* referred by *ApplicationDataType* typically defines *physConstr*.
If a *physConstr* is missing in a *DataConstraint* then existing *CompuMethod* is used to calculate missing information by the tools (Referenced from [\[TPS_SWCT\]](#)).
- 35 ▶ Internal Constraints (*internalConstr*): This attribute describes the limitations applicable on the internal domain. *DataConstraint* referred by *ImplementationDataType* typically defines *internalConstr*. However, developers are not required to specify *internalConstr* for application specific *DataConstraint* as referring to a *DataConstraint* is restricted to *Application-DataType* only which requires *physConstr*.

40 7.5.2 Limits in DataConstraint

Limit specifies an interval boundary for the valid values in a given context (i.e. a data type). The boundary itself might or might not be part of the interval. This is specified using the attribute property *IntervalType* which is assigned with enum values.

The possible enum values that can be assigned to *IntervalType* are as follows.

- 50 ▶ **CLOSED** – value is part of the limit
- ▶ **OPEN** – value is not part of the limit

If the attribute property of the *IntervalType* is missing then the **CLOSED** type is will be taken as default value.

According to the AUTOSAR, the value obtained from a limit shall be a numerical value *constr_1191*. However it is possible to have either integers or float as value for a limit. Developer shall ensure whether the value assigned for a limit will make sense in the given context .

60 7.5.3 Rules for DataConstraint

Rule Data_157: Attributes which are allowed or mandated for DataConstraints

65 Instruction

- ▶ *DataConstraints* shall contain *ShortName* and *DataConstrRule* as mandatory attributes.

- ▶ *LongName* may be an optional attribute.
- ▶ *AdminData* should not be used.

The optionality or mandatoriness of the attribute described in the rule are BOSCH specific and it varies from the one provided by AUTOSAR. Details can be found in [\[TPS_SWCT\]](#)

Rule Data_158: Making a reference to *DataConstraint*

Instruction *DataConstraints* shall be referred from the *SwDataDefProps* of *ApplicationDataType* only.

Hint *DataPrototype* will inherit the *DataConstraints* from *ApplicationDataType*. However referring to *DataConstraint* by *ImplementationDataType* is forbidden due to tooling constraints.

Rule Data_159: Semantics of *DataConstrRule*

Instruction

DerivedFrom: *constr_2561*

A *DataConstrRule* shall contain *physConstr* and *constrLevel*. The *constrLevel* shall always set to "0", representing "hard limits"⁸

Hint Currently only hard limits are supported. This rule is derived from *constr_2561* [\[TPS_SWCT\]](#)

Rule Data_169: Allowed type of *DataConstraint*

Instruction *DataConstraint* of the type *PhysConstr* is the only allowed type. These constraint shall take value of type float.

Rule Data_160: Attributes that composes *PhysConstr*

Instruction A *PhysConstr* shall contain an *UpperLimit*, a *LowerLimit*.

A *PhysConstr* should make a reference to *Unit*. It shall only be omitted if consistent usage of the *PhysConstr* can be guaranteed, e.g. in case of generated code.

Hint AUTOSAR allows other attributes like *maxDiff*, *maxGradient*, *Monotony*, *scaleConstr* as part of *PhysConstr*. However developers are not supposed to use these attributes as there is no support by the tools currently.

Rule Data_161: Where to Define *DataConstraint*

Instruction

DerivedFrom: *CEL_063*

Developers shall use *DataConstraint* defined as part of central elements if it is available.

Hint Application specific *DataConstraint* are created by Developer if it is not available centrally.

⁸ Hard limits imply that calibration tools will not allow to violate these constraints.

Rule Data_162: Reuse of *DataConstraints*

Instruction Existing *DataConstraints* shall be reused if possible.

Rule Data_172: Naming convention for *DataConstraints*

Instruction

Class: NamingConvention

The naming convention for *DataConstraint* shall follow the following syntax:

Constr_[Referenced Unit]_Limits

where "Constr" is the prefix for *DataConstraint* followed by the short name of the unit referenced and finally limits in alphanumeric form.

Example:

Constr_DegC_LL1_UL1

7.6 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 ECU internal representation of data.

CompuMethods are used for the conversion of internal values into their physical representation and vice versa.

For example they are used

1. by the *RTE-Generator* to convert physically specified initial values to internal ones when generating C-Code.

2. by *calibration tools* when calibration parameters are read from the ECU's memory to display them in physical values.

3. by *calibration tools* when changed values of calibration parameters are to be written to ECU's memory.

AUTOSAR mandates mentioning of both conversion direction as per *constr_1021*. However *CompuMethods* are typically specified for only one conversion direction even though both the directions are defined. For *IdenticalCompuMethod* [Chapter \[CompuMethod_Identical\]](#) and *LinearCompuMethod* [Chapter \[CompuMethod_Linear\]](#) inverse functions can be derived from defined functions. However for other complex *CompuMethods* like *RatFuncCompuMethod* [Chapter \[CompuMethod_RatFunc\]](#) *compuPhysToInternal* is the only allowed direction and for *TextTableCompuMethod* [Chapter \[CompuMethod_TextTable\]](#) *compuInternalToPhys* is the only allowed direction.

CompuMethods can also be used to assign symbolic names to internal values (like an enumeration in C) [Chapter \[Enum Values\]](#)

AUTOSAR mandates *ShortName* for *CompuMethod*. But since *Category* also plays a major role in conversion process, *CompuMethods* shall also have a *Category* as mandatory attribute. However AUTOSAR does not provide any default value for *Category*.

AUTOSAR provides 10 categories for *CompuMethods*. But at UBK only few are supported (see [Table \[Compu_Method_Categories\]](#))

Table 40 Categories of CompuMethod

Category	Description	Allowed at UBK
<i>IDENTICAL</i>	This is the simplest of all numerical <i>CompuMethod</i> categories. It involves 1:1 conversion.	Yes
<i>LINEAR</i>	This category of <i>CompuMethod</i> is applicable for linear numerical conversion.	Yes

05

10

15

20

25

30

35

40

45

50

55

60

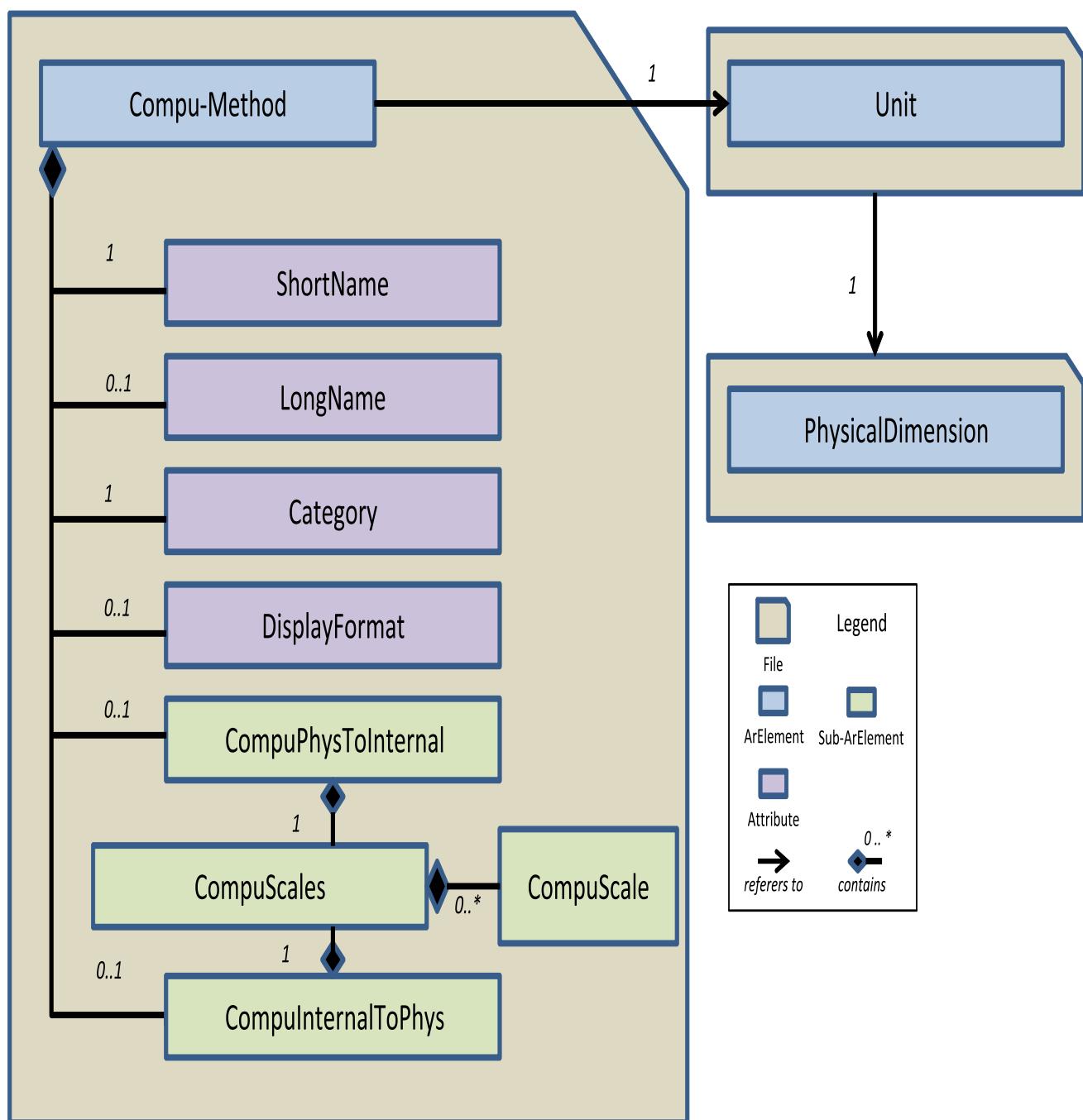
65

70

<i>RAT_FUNC</i>	This category of <i>CompuMethod</i> is applicable for non-linear numerical conversion.	Yes
<i>TEXTTABLE</i>	This category is used for transformations of the internal value into textual elements.	Yes
<i>TAB_NOINTP</i>	This category is used for transformations of the internal value into numerical values.	No
<i>BITFIELD_TEXTTABLE</i>	This category is used for transformations of the internal value into bit fields.	No
<i>SCALE_LINEAR</i>	Used for piece wise Linear conversion	No
<i>SCALE_LINEAR_AND_TEXTTABLE</i>	Used for piecewise definition of one linear and several texttable scales	No
<i>SCALE_RAT_FUNC</i>	Used for piecewise defined rational conversion.	No
<i>SCALE_RATIONAL_AND_TEXTTABLE</i>	Used for piecewise definition of one rational and several texttable scales.	No

All numerical *CompuMethod* shall refer to a *Unit* but *TEXTTABLE* typically will not.

Figure 50 Schematic Representation of CompuMethod

**Example For CompuMethod:**

```

<COMPU-METHOD>
  <SHORT-NAME><!--Construct the Name as per Naming Convention--></SHORT-NAME>
  <LONG-NAME><L-4 L="EN">LongName of CompuMethod</L-4></LONG-NAME>
  <CATEGORY><!--Category of CompuMethod--></CATEGORY>
  <UNIT-REF DEST="UNIT">
    Reference to Unit [Unit]
  </UNIT-REF>
  <COMPU-PHYS-TO-INTERNAL>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">99</UPPER-LIMIT>
    </COMPU-SCALE>
  </COMPU-SCALES>
</COMPU-PHYS-TO-INTERNAL>

```

```

05   <COMPU-RATIONAL-COEFFS>
06     <COMPU-NUMERATOR>
07       <V>0</V>
08       <V>5</V>
09     </COMPU-NUMERATOR>
10     <COMPU-DENOMINATOR>
11       <V>1</V>
12     </COMPU-DENOMINATOR>
13   </COMPU-RATIONAL-COEFFS>
14   </COMPU-SCALE>
15 </COMPU-SCALES>
16 </COMPU-PHYS-TO-INTERNAL>
17 </COMPU-METHOD>

```

ShortName are constructed as per the Naming Convention depending upon the category.

20 Details on *compuScale*

Hint This topic is yet to be reviewed.

25 *CompuMethods* are always defined in terms of "n" number of steps in AUTOSAR . This means conversion from physical values to internal values and vice versa takes place in terms of intervals. These "N" number of intervals in which a particular conversion takes place are called as *compuScales*. The interval is restricted by upper and lower limits. Details could be found in [\[TPS_SWCT\]](#) , [constr_1024](#).

30

35

40

45

50

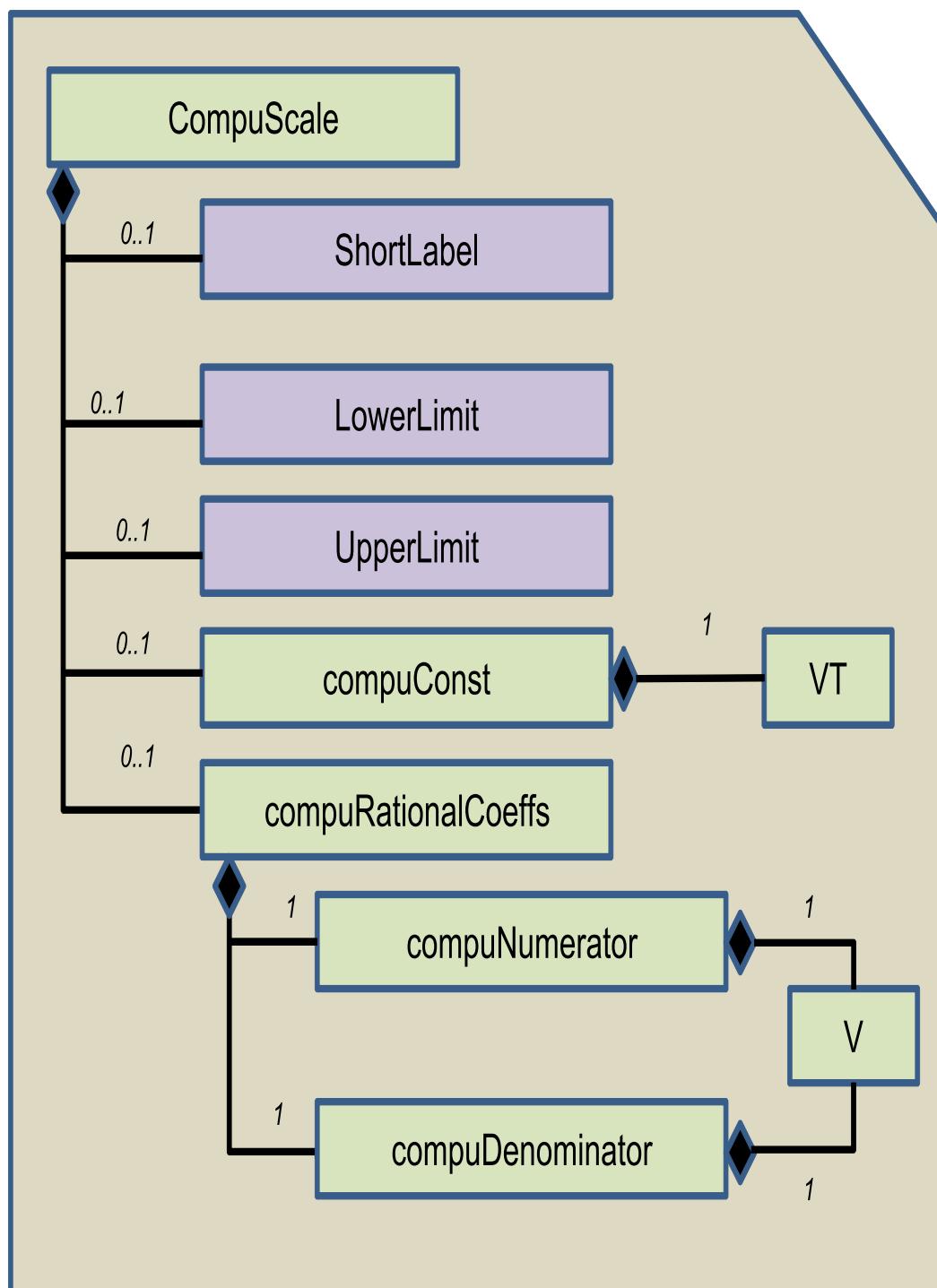
55

60

65

70

Figure 51 CompuScale



A *compuScale* can be specified in the following ways:

- 05 ▶ Involving **CompuConst**: This approach involves mapping of a literal to the *compuScale*. Here we make use of a meta class *CompuConst* which provides a literal. This literal will be mapped to *compuScale*. The literal is represented by a *VT* element.

```
10 <COMPU-METHOD>
..<COMPU-SCALES>
    <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0.0</LOWER-LIMIT> <!-- Lower limit of the interval -->
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0.0</UPPER-LIMIT> <!-- Upper limit of the interval -->
        <COMPU-CONST>
            <VT>CANIF_CS_OPENED</VT> <!-- Here "CANIF_CS_OPENED" is a constant -->
        </COMPU-CONST>
    </COMPU-SCALE>
</COMPU-SCALES>
...
</COMPU-METHOD>
```

- 15 20 25 ▶ Involving **CompuRationalCoeffs**: This approach involves rational functions to represent *compuScale*. Here we make use of a meta class *CompuRationalCoeffs*. A *CompuRationalCoeffs* contains numerator, known as *compuNumerator* and a denominator, known as *compuDenominator*. A Rational function is specified as rational coefficients for *compuNumerator* and *compuDenominator*.

30 A *compuNominatorDenominator* (a *compuNumerator* and a *compuDenominator*) will contain a sequence of *V* elements. Each of this *V* element carries information for exponents that means the first *V* element is the coefficient for x^0 and second *V* element is the coefficient for x^1 etc.

```
35 <COMPU-METHOD>
...
<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-METHOD>
```

50 55 60 65 70 7.6.1 Rules for CompuMethods

Rule Data_124: Where to Define CompuMethod

Instruction

DerivedFrom: CEL_051

DerivedFrom: CEL_052

CompuMethods should be coming from central elements. If a suitable *CompuMethod* exists as central element it shall be used instead of creating a local one. Arithmetic *CompuMethods* (These includes Categories IDENTICAL, LINEAR, RAT_FUNC) should be centralized. Textual *CompuMethods* (of Category TEXTTABLE) should be centralized if they are reusable.

Rule Data_147: Attributes which are allowed or mandated for all categories of CompuMethods

Instruction

RelatedTo: constr_1021

- ▶ Each CompuMethod shall have a *ShortName*.
- ▶ Each CompuMethod shall have a *Category*.
- ▶ Each CompuMethod should specify conversion directions (depending on *Category*):
 - compuPhysToInternal
 - compuInternalToPhys
- ▶ Each CompuMethod may specify *LongName*.
- ▶ Each CompuMethod shall specify *Unit* depending upon the *Category* of CompuMethod.
- ▶ Each CompuMethod may specify *Desc*, *AdminData*, *Introduction*, *Annotation*
- ▶ Each CompuMethod should not specify *DisplayFormat*.

DisplayFormat is usually specified by the *DataPrototypes*. So it is recommended not to specify it at CompuMethod.

Desc, *Admin-Data*, *Introduction*, *Annotation*, *LongName* are optional since they do not have any tooling impact.

All numerical CompuMethod shall refer to a *Unit* except of *TEXTTABLE* which typically does not.

The optionality or mandatoriness of the attribute described in the rule are BOSCH specific and it varies from the one provided by AUTOSAR. Details can be found in [\[TPS_SWCT\]](#)

Hint Direction of the defined *compuScale* depends on the *Category* of the CompuMethod. The rules [\[Data_115\]](#), [\[Data_117\]](#), [\[Data_118\]](#) will give details of the conversion direction for different categories of CompuMethod

Rule Data_110: Unit reference by different categories of CompuMethods

Instruction

RelatedTo: constr_1175 in TPS_SWCT "Software Component Template"

1. CompuMethod of Category *IDENTICAL*, *LINEAR*, *RAT_FUNC* shall refer to a *Unit*. If there is no *Unit* available, then the *Unit* with *ShortName* "NoUnit" shall be referenced.
2. CompuMethod of Category *TEXTTABLE* shall also refer to a *Unit* but it will be typically the *Unit* named "NoUnit".

Hint *Unit* reference by CompuMethod is illustrated at [\[CompuMethods\]](#)

The name "NoUnit" is not approved by ASAM⁹ and the name might be changed in future.

The *Unit* named "NoUnit" will be defined as central element.

Example for *ShortName* named NoUnit may be referred at [\[Unit\]](#)

For description on *Units* refer to [\[Unit_PhysDim\]](#)

Rule Data_132: Allowed Categories for CompuMethod

Instruction Not all the categories defined by AUTOSAR are supported in UBK yet. Developer shall use the categories *IDENTICAL*, *LINEAR*, *RAT_FUNC*, *TEXTTABLE* for their developmental activities.

The optionality or mandatoriness of the attribute described in the rule are BOSCH specific and it varies from the one provided by AUTOSAR

⁹ Association for Standardisation of Automation and Measuring System

Details on different categories can be found in [\[TPS_SWCT\]](#)

Rule Data_090: When to use which CompuMethod Category

Instruction Developer shall use:

- ▶ *CompuMethod of Category IDENTICAL* if identical (one to one) relation exist between physical and internal values.
- ▶ *CompuMethod of Category LINEAR* if linear relation exists between physical and internal values and if identical relation is not applicable.
- ▶ *CompuMethod of Category RAT_FUNC* if numerical relation exists between physical and internal values and linear relation is not applicable.
- ▶ *CompuMethod of Category TEXTTABLE* if text values are to be assigned to the internal values.

Rule Data_167: Using a fractional number as part of a ShortName

Instruction

Class: NamingConvention

Whenever a fractional number is to be used as part of a ShortName it shall be done using the following format:

m?{number} (p{number}) ? (Em?{number}) ?

where

- ▶ **number:** [1-9] ([0-9]) *
- ▶ "p" represents decimal point (".")
- ▶ "E" represents exponent
- ▶ "m" represents a minus "-"

Example:

The number 2.23exp-10 shall be represented as 2p23Em10

Hint This is a sub rule used in rules for naming convention of CompuMethod names. Naming Convention for the CompuMethods themselves are specified by rule for the different categories of CompuMethods.

Rule Data_113: Removed

Instruction

Status: removed

Rule Data_116: Removed

Instruction

Status: removed

Rule Data_111: Removed

Instruction

Status: removed

Rule Data_133: Removed

Instruction

Status: removed

7.6.2 IDENTICAL CompuMethod

This is the simplest type among all numerical categories of *CompuMethod*. This *CompuMethod* just hands over the internal value as physical value or vice versa.

For a *CompuMethod* of Category *IDENTICAL*, *ShortName* , *Category*, reference to *Unit* are mandatory elements. The data defined for *IDENTICAL CompuMethod* ranges from *-INF* to *+INF*.

CompuMethod_Identical**Example:**

```
<COMPU-METHOD>
  <SHORT-NAME>KelvinIdentcl</SHORT-NAME><!--Assuming reference uint is Kelvin-->
  <CATEGORY>IDENTICAL</CATEGORY>
  <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>
  <UNIT-REF DEST="UNIT"><!--Add a reference to Unit here --></UNIT-REF>
</COMPU-METHOD>
```

Note:

Units are explained in [\[Unit_PhysDim\]](#) .

7.6.2.1 Rules for Identical CompuMethod**Rule Data_146:** Attributes which are allowed or mandated for *CompuMethod* of Category *IDENTICAL*

Instruction

- ▶ A *CompuMethod* of Category *IDENTICAL* shall have a *ShortName*
- ▶ *Category* shall be set to *IDENTICAL*
- ▶ A *CompuMethod* of Category *IDENTICAL* shall reference a *Unit*.
- ▶ A *CompuMethod* of Category *IDENTICAL* should not specify *physConstr* or *internalConstr*
- ▶ A *CompuMethod* of Category *IDENTICAL* shall not contain *compuScales*

Since in an Identical *CompuMethod* no conversions are required, *compuScales* are not necessary.

Rule Data_163: Naming Convention for *CompuMethod* of Category *IDENTICAL*

Instruction

Class: NamingConvention

The name of a *CompuMethod* of category *IDENTICAL* shall follow the syntax:

<shortName of the Unit>+Identcl

where

- ▶ <shortName of the Unit>: is the short name of the referenced *Unit*
- ▶ Identcl: Keyword to represent Identical CompuMethod.

7.6.3 LINEAR CompuMethod

The Category *LINEAR* is applicable for a *CompuMethod* if linear relation exists between physical and internal values.

A linear conversion is performed in two steps:

- ▶ The internal value is multiplied with a linear conversion factor
- ▶ Then an offset is added.

ShortName, *Category* and *Unit* are mandatory attributes.

LongName is optional.

AUTOSAR allows to define the limits on range of *physicalValues* to be used. However for a linear conversion, a *physicalValue* can take any value without any limitations. As such defining a limit is not required.

A linear *CompuMethod* requires exactly one *compuScale* with 2 Compu Numerator and 1 Compu Denominator.

The following examples illustrates how a linear conversion is specified using *CompuMethod*.

$$F_{[\text{ms}]} = 5.0_{[\text{ms}]} + 2.0_{[\text{ms}]} * X$$

Example:

```
<COMPU-METHOD>
  <SHORT-NAME>VoltLnr1</SHORT-NAME><!-- Assuming referenced unit is Voltage -->
  <CATEGORY>LINEAR</CATEGORY>
  <UNIT-REF DEST="UNIT"><!--Add a reference to Unit here --></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>
```

Note:

Units are explained in [\[Unit_PhysDim\]](#).

7.6.3.1 Rules for *LINEAR* CompuMethod

Rule Data_115: Conversion Direction for CompuMethod of Category *LINEAR*

Instruction *CompuMethod* of category *LINEAR* shall be defined using *compuPhysToInternal*.

Inverse function can be derived quite easily from the defined function.

Rule Data_148: Restrictions on *compuScale* in LinearCompuMethod

Instruction A LINEAR CompuMethod shall specify one compuScale. Its compuNumerator shall contain two Vs, the first one specifying the offset and the second one specifying the linear conversion factor. Its compuDenominator shall be set to 1.0.

Hint This rule needs to be reviewed for the value a compuDenominator can have which is other than 1.0.

Rule Data_149: Attributes which are allowed or mandated for LinearCompuMethod

Instruction Attributes for a LinearCompuMethod:

- ▶ A LinearCompuMethod shall have *ShortName* and a reference to an *Unit*.
- ▶ *Category* shall be set to LINEAR
- ▶ A Linear CompuMethod may specify a *LongName*.
- ▶ Attributes *physConstr* and *internalConstr* shall not be specified.

Hint *LinearCompuMethod* are always defined for the whole range of -INF to +INF.

Rule Data_092: Floating numbers to be used for Numerical CompuMethod

Instruction The values within *compuScales* of *CompuMethods* shall be entered as float. Float numbers shall always be specified with decimal point ".".

e.g. "5.0" instead of "5"

Hint Float numbers shall follow the syntax of primitive numerical datatype as specified in Generic Structure Template [TPS_GST]

Rule Data_164: Naming Convention for LinearCompuMethod

Instruction

Class: NamingConvention

The name of a CompuMethod of category LINEAR shall follow the syntax:

<shortName of Unit>+Lnr+<sequenceNumber>

where

- ▶ <shortName of Unit>: is the short name of the referenced *Unit*
- ▶ Lnr: is the keyword used to represent Linear CompuMethod
- ▶ {sequenceNumber}: A positive integer to make the name unique in the given name space.

Hint This naming convention was originally coming from AUTOSAR. Review by BUs is open, scope of the rule might shrink after review.

7.6.4 RAT_FUNC CompuMethod

The rational function type is similar to the linear type without the restrictions for the *compuNumerators* and *compuDenominators*. AUTOSAR allows a *CompuMethod* of Category RAT_FUNC to have as many **V** elements as needed for the rational function. However at UBK the number of **V** elements that a *CompuNumerator* can have, is restricted to 1 and *compuDenominator* can have, is restricted to 2. The sequence of the values in *compuDenominator*, **V** carries the information for the exponents, that means the first **V** is the coefficient for x^0 , the second **V** is the coefficient for x^1 .

$$\text{Internal} = (v_{[0]} * x^0) / (w_{[0]} * x^0 + w_{[1]} * x^1)$$

Hint The conversion formula can be found at "Software Component Template" [TPS_SWCT]

ShortName, *Category* and *Unit* reference are mandatory attributes.

LongName is optional.

RAT_FUNC CompuMethod will specify the range for *compuNumerator* and *compuDenominator* through Lower-Limit and Upper-Limit attributes. For details on range specification refer [DataConstr]

The following examples illustrates how a Rat_Fun conversion is specified using *CompuMethod*.

At UBK the *RAT_FUNC CompuMethods* are only in use to express hyperbolic dependencies. That is that $v_{[1]}$ will not be used.

$$\text{Int} = 1000.0 / (60.0 + 2.0 * \text{Phys})$$

Example:

```
<COMPU-METHOD>
  <SHORT-NAME>Rat_n1000p0_d60p0_d2p0_DegC</SHORT-NAME>
  <CATEGORY>RAT_FUNC</CATEGORY>
  <UNIT-REF DEST="UNIT"><!-- Refer to an UNIT --></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>1000.0</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>60.0</V>
            <V>2.0</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>
```

Note:

Units are explained in [Unit_PhysDim]

7.6.4.1 Rules for RatFunc CompuMethod

Hint rule set for RAT_FUNC is not yet as complete as for Linear!

Rule Data_156: Attributes which are allowed or mandated for RatFunc CompuMethod

Instruction Attributes for RatFunc CompuMethod are:

- 05 ▶ A *RAT_FUNC CompuMethod* shall specify a *ShortName* and a reference to *Unit*.
- 10 ▶ *Category* shall be set to *RAT_FUNC*
- 15 ▶ *compuPhysToInternal* is the only conversion direction allowed.
- 20 ▶ A *RAT_FUNC CompuMethod* shall specify interval of the values (Lower-Limit Interval and Upper-Limit Interval) that are specified for *compuNumerator* and *compuDenominator*.

Rule Data_117: Removed

15 Instruction

Status: removed

Rule Data_150: Restriction on having the number of V elements

20 Instruction A *RAT_FUNC CompuMethod* shall specify one *compuScale*. Its *compuNumerator* shall contain one *V* element and its *compuDenominator* shall contain two *V* elements.

25 Hint Restriction on the usage of number of *V* elements at *compuNumerator* is due to domain constraints.

30 Int = 1000.0 / (60.0 + 2.0 * Phys)

Rule Data_165: Naming Convention for CompuMethod of category RAT_FUNC

35 Instruction

Class: NamingConvention

The name of a *CompuMethod* of category *RAT_FUNC* shall follow the syntax:

40 Rat({_n{Nominator}}* {_d{Denominator}}* {_U{Unit}} {_C{Counter}})?

45 where

- 50 ▶ *{Nominator}*: A fractional number with decimal point which correlates to the nominator, according to their order.
It shall be converted to a string as defined in [\[Data_167\]](#).
- 55 ▶ *{Denominator}*: A fractional number with decimal point which correlates to the denominator, according to their order.
It shall be converted to a string as defined in [\[Data_167\]](#).
- 60 ▶ *{Unit}*: is the short name of the referenced *Unit*
- 65 ▶ *{Counter}*: A positive integer to make the name unique in the given name space.

For the representation of a fractional number refer to rule [\[Data_167\]](#)

70 **Example to derive name of the CompuMethod** If nominators are 1.23, 2.34 and denominators are 3.45, 4.56, Unit is mgh/kg/s then *CompuMethod* shall be named as:

75 Rat_n1p23_n2p34_d3p45_d4p56_mg_h_Per_kg_s

7.6.5 TEXTTABLE CompuMethod

80 The type TEXTTABLE is used for transformations of the internal value into textual elements.

85 Hint TEXTTABLE *CompuMethod* do not support physToInternalConstr.

05 These *CompuMethods* of Category *TextTable* contains *CompuScales* with point ranges (i.e. lower and upper limit of a *CompuScale* are identical.)

10 Table 41 Tabular representation of Literal – Value

Literal	Value
false	0
true	1

15 **Example:**

```

<COMPU-METHOD>
  <SHORT-NAME>Txt_false_true</SHORT-NAME>
  <CATEGORY>TEXTTABLE</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>false</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>true</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

40 7.6.5.1 Handling of Enumeration Data Types

45 Enumeration is not a plain primitive *ImplementationDataType*, but a range of integers that are mapped on "labels". This mapping is realised using *CompuMethod* of Category *TextTable*.

50 For generating enum literals in the Rte generated code, developer shall refer to a *CompuMethod* of Category *TextTable* at the *ApplicationDataType* level in case of ASW or at the *ImplementationDataType* level in case of BSW.

55 The *Application Types Header File* (*.Type.h file) will contain the definitions of all enumeration constants for each *ApplicationDataType* referenced by a *SoftwareComponent*.

60 Each *CompuScale* which has a point range and is located in the *CompuInternalToPhys* container of a *CompuMethod* with *CategoryTextTable* and referenced by an *ApplicationPrimitiveDataType* has an entry in *Application Types Header File*.

65 The definition of the Enumeration data type in *Application Types Header File* is as follows :

```

Definition of the Enum Constant in .h File
#ifndef <prefix><EnumLiteral>
#define <prefix><EnumLiteral>    <value><suffix>
#endif /*<prefix><EnumLiteral> */
```

70 where **<prefix>** is the optional attribute which refers to *AUTOSAR.DataType* using *CompuMethod*, **<value>** is the value representing the *CompuScale*'s point range, **<suffix>** shall be "U" for unsigned data types and empty for signed data types and **<EnumLiteral>** is the name of the enumeration literal which is derived according to the following rule:

75 **<EnumLiteral>** is assigned with the string specified in *VT* element of *compuConst* of *CompuScale*.

05 The following example illustrates how *TextTableCompuMethod* can be used to handle Enumerated values.

Example:

```

<COMPU-METHOD>
  <SHORT-NAME>Txt_SensorError_win</SHORT-NAME>
  <CATEGORY>TEXTTABLE</CATEGORY>
  <UNIT-REF DEST="UNIT"><!-- Refer to a Unit with short name "No Unit" --></UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">352</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">352</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>SensorError_win</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Generated .h Code:

```

/* BEGIN: SWC specific definitions */
#ifndef RTE_CORE
#ifndef PreSwc1SensorError_win
#define PreSwc1SensorError_win           352U
#endif /*PreSwc1SensorError_win*/
#endif /* RTE_CORE */
/* END: SWC specific definitions */
```

Hint Generation of the Code by RTE is supported only for Enum Literals coming for _swcd.arxml file. RTE does not support generation of code for Enum Literals coming from _BSWMD.arxml.

7.6.5.2 Rules for TextTable CompuMethod

Rule Data_118: Attributes which are allowed or mandated for TEXTTABLE CompuMethod

Instruction *compuInternalToPhys* is the only allowed direction of the conversion for *CompuMethod* of category TEXTTABLE.

45 *compuInternalToPhys* shall contain *compuScales* consisting of *UpperLimit* and *LowerLimit*, both with *INTERVAL-TYPE="-CLOSED"* and both shall specify the same numerical internal value.

The result of conversion shall be placed in *VT*. All *VTs* of a single *CompuMethod* shall be unique.

CompuDefaultValue should not be used.

Rule Data_145: Removed

Instruction

Status: removed

Rule Data_166: Naming Convention for CompuMethod of category TEXTTABLE

Instruction

Class: NamingConvention

65 The name of a *CompuMethod* of category TEXTTABLE shall follow the syntax:

*Txt (_{TextValue}) * (_Counter) ?*

where

- ▶ {TextValue}: The text Values according their order.
- ▶ {Counter}: A positive integer to make the name unique in the given name space.

Example to derive name of the CompuMethod If texts are "true" and "false" then CompuMethod shall be named as:

Txt_true_false

Rule Data_181: *Vt* shall either be a valid C-Identifier or *Symbol* shall be defined

Instruction Content of *Vt* elements shall follow C-Identifier syntax.

If *Vt* is explicitly designed to have no C-Identifier syntax developer shall define *Symbol* additionally. In this case *Symbol* shall have C-identifier syntax.

Hint RTE_Gen creates enum literals from *Symbols* (if defined) or *Vt* (if no *Symbol* defined). Enum literals have to have C-Identifier syntax. RTEGen will not accept your input if C-identifier syntax is not followed.

7.7 Record Layout

Instruction Record Layouts are defined and provided from the both packages for the interpolation routines (IFL and IFX) . Developers shall make use of these Record Layouts for their data elements. This can be referred in "*Central Elements*" [A_CEL]

7.8 Addressing Methods

Instruction SwAddrMethods are defined as CentralElements . Developers shall make use of theseSwAddrMethods for their data elements. This can be referred in "*Central Elements*" [A_CEL]

7.8.1 Rules

Rule Data_122: Removed

Instruction

Status: removed

7.9 Unit and PhysicalDimension

7.9.1 Unit

An important part of the semantics associated with a data type is its physical units. A *Unit* is a physical quantity, defined and adopted by convention with which other particular quantities of same kind are compared to express their value.

All *Units* that might be defined, should stem from SI unit. [MOD_AISpecificationExamples]

For a *Unit*, *ShortName*, *displayName*, *factorSiToUnit* and *offsetSiToUnit* are mandatory elements.

A *displayName* specifies how the *Unit* should be specified in the document or in user interface tool.

In order to convert a SI unit to *Unit* or vice versa factor and offset are defined.

- *factorSiToUnit* – This is the factor for the conversion from and to SI unit
 ► *offsetSiToUnit* – This is the offset for the conversion from and to SI unit

For the calculation from SI unit to the defined unit the factor (*factorSiToUnit*) and the offset (*offsetSiToUnit*) are applied.

Unit = SI unit * factorSiToUnit + offsetSiToUnit

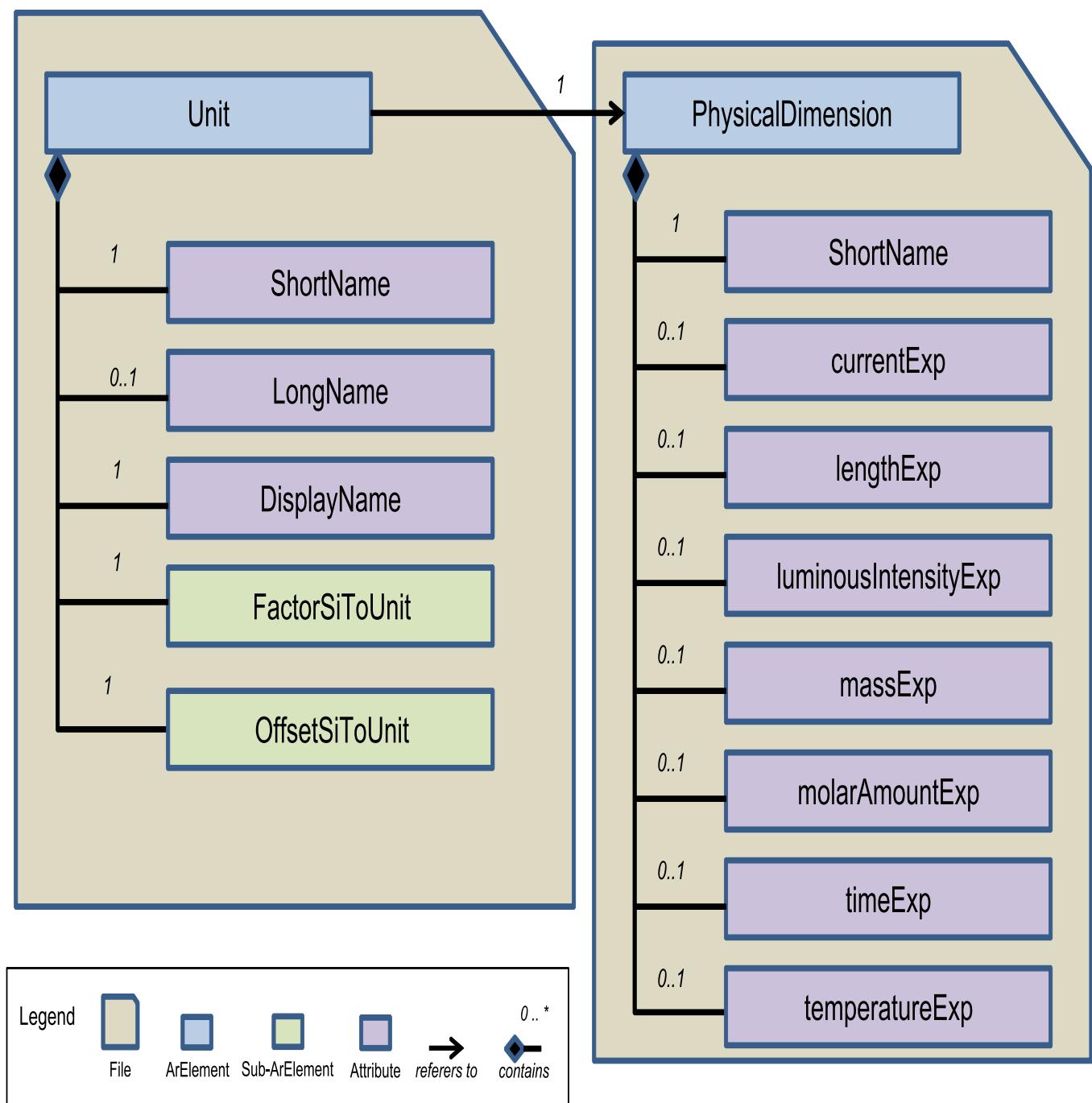
For the calculation from a unit to SI-unit the reciprocal of the factor (*factorSiToUnit*) and the negation of the offset (*offsetSiToUnit*) are applied:

$$\text{SI unit} = (\text{Unit} - \text{offsetSiToUnit}) / \text{factorSiToUnit}$$

The default value for factor *factorSiToUnit* is 1 and offset *offsetSiToUnit* is 0.

Units are defined as part of central elements and developers will make a reference to them.

Figure 52 Unit and PhysicalDimension



05

Example:

```

<UNIT>
  <SHORT-NAME>DegCgrd</SHORT-NAME>
  <LONG-NAME><L-4 L="EN">Degree Centigrade</L-4></LONG-NAME>
  <DESC><L-2 L="EN">temperature, no SI unit, (degC = K - 273.15)</L-2></DESC>
  <DISPLAY-NAME>degC</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO-UNIT>-273.15</OFFSET-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
    <!-- Refer to a Physical Dimension-->
  </PHYSICAL-DIMENSION-REF>
</UNIT>
```

10

CompuMethods shall refer to "NoUnit" as default unit, if there are no *Units* available. Such *Units* will make a reference to physicaldimension with exponent value 0 only.

15

Example For Unit "NoUnit":

```

<UNIT>
  <SHORT-NAME>NoUnit</SHORT-NAME>
  <LONG-NAME><L-4 L="EN">No Unit</L-4></LONG-NAME>
  <DISPLAY-NAME>noUnit</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>1.0</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO_UNIT>0.0</OFFSET-SI-TO_UNIT>
  <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
    <!-- Refer to a Physical Dimension whose exponent value is set to 0 only -->
  </PHYSICAL-DIMENSION-REF>
</UNIT>
```

20

25

30

35

7.9.2 Physical Dimension

Units are used to augment the value with additional information called physical dimensions (e.g., m/s or liter). Physical Dimensions are necessary for a correct interpretation of the physical values that serves as data for input and output processes. Each *Unit* will make a reference to one physical dimension.

40

The conversion of one *Unit* to another is possible only if their physical dimensions are identical. Two physical dimensions are said to be identical if following 2 conditions are satisfied :

45

- ▶ The values of *lengthExp*, *massExp*, *timeExp*, *currentExp*, *temperatureExp*, *molarAmountExp*, *luminousIntensityExp* are identical. Each of this quantity represents exponent value for length, mass, time, electric current, temperature, quantity of mass and luminous intensity respectively.
- ▶ The *ShortNames* are identical or physical dimension mapping exists that maps one of the physical dimension in the role of first physical dimension and other in the role of second physical dimension.

50

AUTOSAR defines "0" as the default value for all this 7 exponents. Negative Exponents are also allowed. For details refer TPS_SWCT_01060 [\[TPS_SWCT\]](#)

Physical Dimensions are defined as central elements and developer will only make a reference to them.

55

Example:

```

<PHYSICAL-DIMENSION>
  <SHORT-NAME>T1</SHORT-NAME>
  <LONG-NAME><L-4 L="EN">Temperature 1</L-4></LONG-NAME>
  <TEMPERATURE-EXP>1</TEMPERATURE-EXP>
</PHYSICAL-DIMENSION>
```

60

65

70

7.9.3 Rules for Units and PhysicalDimension

Rule Data_112: Attributes which are allowed or mandated for for *Unit*

Instruction

1. Every *Unit* shall contain *ShortName*.
2. Every *Unit* shall contain *factorSiToUnit*.
3. Every *Unit* shall contain *OffsetSiToUnit*.
4. Every *Unit* shall contain reference to *PhysicalDimension*.
5. Every *Unit* shall contain *DisplayName*.

The optionality or mandatoriness of the attribute described in the rule are BOSCH specific and it varies from the one provided by AUTOSAR. Details can be found in [\[TPS_SWCT\]](#)

Rule Data_121: Removed

Instruction

Status: removed

Rule Data_125: Where to Define *Units* and Physical Dimensions

Instruction

DerivedFrom: CEL_001

DerivedFrom: CEL_031

Units and Physical Dimensions shall be defined as central elements. Developers shall only make a reference to these *Units* and Physical Dimension and shall not create them.

Rule Data_168: Naming Convention for *Units*

Instruction

Class: NamingConvention

The name of a *Unit* shall be derived from its content of *DisplayName* as

- ▶ If the unit is a formula containing "x to the power of 2" the short name contain "Sqrd".
- ▶ If the unit is a formula containing "x to the power of 3" the short name shall contain "Cubd".
- ▶ If the unit is a formula containing "x to the power of number >3" the short name shall contain ToPwrOf <number>.
- ▶ If the unit is a formula containing the division the short name shall contain "Per".

Example:

m/s³ will be represented as MtrPerSecCubd.

Rule Data_180: Naming Convention for *PhysicalDimensions*

Instruction

Class: NamingConvention

The name of a *PhysicalDimension* shall be of the form:

<BaseDimensionKeyword1[-]{exp}><BaseDimensionKeyword2[-]{exp}>.....n where,

05 BaseDimensionKeyword: BaseDimension Key word represents the key word representing each of base quantities like length, mass, time etc. These keywords are standardised by Keyword data base.

exp is the exponent of the dimension. Exponent can be negative as well.

10 Example:

Force can be represented in terms of base quantity as $(\text{Mass} \times \text{Length}) / (\text{Time})^2$. The physical dimension for force can be represented as follows.

<PHYSICAL-DIMENSION>

```
15 <SHORT-NAME>Len1M1TiNeg2</SHORT-NAME>
    <LONG-NAME><L-4 L="EN">Length 1 Mass 1 Time -2</L-4></LONG-NAME>
    <LENGTH-EXP>1</LENGTH-EXP>
    <MASS-EXP>1</MASS-EXP>
    <TIME-EXP>-2</TIME-EXP>
</PHYSICAL-DIMENSION>
```

Rule Data 126: Removed

Instruction

Status: removed

Rule Data 152: Making a reference to *Units*

Instruction Units shall be referenced only from *CompuMethods* and *PhysConsts* and not from *SwDataDefProps*.

Rule Data 153: Accuracy of `factorSiToUnit` and `OffsetSiToUnit`

Instruction *factorSiToUnit* and *OffsetSiToUnit* shall use the same accuracy (as specified from the responsible AUTO-SAR working group). If a unit has not yet been specified by AUTOSAR the accuracy shall be set to 8 digits.

Hint Counted are the digits before and after the dot. Leading zeroes before the dot and trailing zeroes after the dot are not counted.

Rule Data 154: *PhysicalDimension* for Absolute and Relative Units

Instruction Distinct *PhysicalDimensions* shall be defined for absolute and relative *Units* that have a non-zero value for *offsetSiToIInit*.

The value of a given *Unit* may be interpreted as absolute value or difference. Thus, for each kind of *Unit* an appropriate *PhysicalDimension* has to be specified.

Reason: In order not to wrongly combine resp. intermix absolute and relative *Units* in computation methods. Only values with *Units* of compatible *PhysicalDimensions* might be converted.

60 Example: The temperature difference may be expressed either in degree celsius or Kelvin using the same numerical value. Hence, both a *PhysicalDimension* denoting the absolute temperature and a *PhysicalDimension* denoting the relative temperature is defined.

Example:

65 <PHYSICAL-DIMENSION>
 <SHORT-NAME>Temp_abs</SHORT-NAME>

```

05      <TEMPERATURE-EXP>1</TEMPERATURE-EXP>
</PHYSICAL-DIMENSION>
<PHYSICAL-DIMENSION>
10      <SHORT-NAME>Temp_rel</SHORT-NAME>
      <TEMPERATURE-EXP>1</TEMPERATURE-EXP>
</PHYSICAL-DIMENSION>

15      ....
<UNIT>
      <SHORT-NAME>Kelvin_absolute</SHORT-NAME>
      <DISPLAY-NAME>K</DISPLAY-NAME>
      <FACTOR-SI-TO_UNIT>1.0</FACTOR-SI-TO_UNIT>
      <OFFSET-SI-TO_UNIT>0.0</OFFSET-SI-TO_UNIT>
      <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
          rbaCUCEL_PhysicalDimensions/Temp_abs
      </PHYSICAL-DIMENSION-REF>
20      </UNIT>
<UNIT>
      <SHORT-NAME>Kelvin_relative</SHORT-NAME>
      <DISPLAY-NAME>K(rel)</DISPLAY-NAME>
      <FACTOR-SI-TO_UNIT>1.0</FACTOR-SI-TO_UNIT>
25      <OFFSET-SI-TO_UNIT>0.0</OFFSET-SI-TO_UNIT>
      <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
          rbaCUCEL_PhysicalDimensions/Temp_rel
      </PHYSICAL-DIMENSION-REF>
</UNIT>
30      <UNIT>
          <SHORT-NAME>Celsius_absolute</SHORT-NAME>
          <DISPLAY-NAME>°C</DISPLAY-NAME>
          <FACTOR-SI-TO_UNIT>1.0</FACTOR-SI-TO_UNIT>
          <OFFSET-SI-TO_UNIT>-273.14</OFFSET-SI-TO_UNIT>
          <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
              rbaCUCEL_PhysicalDimensions/Temp_abs
          </PHYSICAL-DIMENSION-REF>
      </UNIT>
<UNIT>
        <SHORT-NAME>Celsius_relative</SHORT-NAME>
        <DISPLAY-NAME>°C (rel)</DISPLAY-NAME>
        <FACTOR-SI-TO_UNIT>1.0</FACTOR-SI-TO_UNIT>
40        <OFFSET-SI-TO_UNIT>0.0</OFFSET-SI-TO_UNIT>
        <PHYSICAL-DIMENSION-REF DEST="PHYSICAL-DIMENSION">
            rbaCUCEL_PhysicalDimensions/Temp_rel
        </PHYSICAL-DIMENSION-REF>
    </UNIT>
45

```

50 **Hint** Naming Convention adopted for the *ShortName* in this example is not yet reviewed.

Rule Data_177: Optional/Mandatory Exponent values in *PhysicalDimension*

55 **Instruction** In a *PhysicalDimension*, exponent values may not be defined if a default value is available in AUTOSAR. However there are *PhysicalDimensions* for which default exponent values are not defined in AUTOSAR. In this case user shall specify the value for the Exponent.

60 **Hint** AUTOSAR defines "0" as default value for all the basic quantities. Developer has to provide default value for any other quantity.

65

05

7.10 UnitGroup

10
Units can be grouped with the help of *UnitGroup*. Logical grouping of *Units* allows MCD device to present different unit system to the user such that he can chose the most appropriate one.

15

7.10.1 Rules forUnitGroup

20

Rule Data_151: Usage of *UnitGroups*

25
Instruction *UnitGroups* shall not be used by developers as they are not supported by tool chain currently.

30

35

40

45

50

55

60

65

7.11 Overview of AUTOSAR Categories, Data types and related Elements

The following table gives an overview of the Categories in AUTOSAR 4.1 used for data types and related elements. (Referenced from *Document "Software Component Template" [TPS_SWCT]* Table 5.7)

Figure 53 Overview of AUTOSAR Categories, Data types and related Elements

Category	Applicable to ...										Use Case		Description		
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW
VALUE	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
VAL_BLK	x	x	x	x							x	x			
DATA_REFERENCE					x	x	x	x					x		
FUNCTION_REFERENCE					x	x	x	x					x		
TYPE_REFERENCE					x	x	x	x					x	x	

Figure 54 Overview of AUTOSAR Categories, Data types and related Elements(Contd...)

Category	Applicable to ...										Use Case	Description				
	ApplicationArrayType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ImplementationDataType	ImplementationDataElement	SwPointerTargetProps	SwServiceArg	SwSystemConst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW
STRUCTURE	x		x	x	x	x	x		x	x		x	x	x	x	x
UNION					x	x	x				x	x	x	x	x	x
ARRAY	x		x	x	x	x	x	x			x	x	x	x	x	x
BIT											x	x	x	x	x	x
HOST											x	x	x	x	x	x
STRING		x	x	x	x						x	x	x	x	x	x
BOOLEAN		x	x	x	x						x	x	x	x	x	x
COM_AXIS		x		x							x	x				
RES_AXIS		x		x							x	x				

Figure 55 Overview of AUTOSAR Categories, Data types and related Elements(Contd...)

Category	Applicable to ...										Use Case	Description				
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ImplementationDataType	ImplementationDataElement	SwPointerTargetProps	SwServiceArg	SwSystemConst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW
CURVE		X	X	X	X						X	X				Calibration parameter with one input value and one output value. That means output values can be defined depending on the input value . The granularity of implemented functionality can be changed by using different number of axis points. A CURVE has always one input axis and one output axis. The output axis is a characteristic of the curve and every time present but the input axis can be defined within the curve definition or separately.
MAP		X	X	X	X						X	X				Calibration parameter with two input values and one output value. That means output values can be defined depending on the input values .The granularity of implemented functionality can be changed by using different number of axis points for y- and x-axis. A MAP has always two input axes and one output axis. The output axis is a characteristic of the map and every time present but the input axes can be defined within the map definition or separately.

The following table gives an overview of the Categories vs ApplicationDataTypes in AUTOSAR 4.1. (Referenced from Document "Software Component Template" [TPS_SWCT] , Table 5.7, AR 4.2.1)

Hint In case of any confusion regarding the categories, data types and related elements w.r.t AUTOSAR 3.x use the rule applicable as in AUTOSAR 4.x.

Figure 56 Categories vs ApplicationDataTypes in AUTOSAR

	Root Element			Attribute Existence per Category									
	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP
Attributes of SwDataDefProps													
additionalNativeTypeQualifier				*	*	*	*	*	*	*	*	*	*
annotation	X	X	X										
baseType													
compuMethod	X	X	X	0..1	0..1			0..1	0..1			0..1	0..1
dataConstr	X	X	X	0..1	0..1				0..1			0..1	0..1
displayFormat	X	X	X	0..1	0..1			0..1	0..1			0..1	0..1
implementationDataType													
invalidValue	X	X	X	0..1				0..1	0..1				
mcFunction													
swAddrMethod	X	X	X	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment													
swBitRepresentation													
swCalibrationAccess	X			1	1	1	1	1	1	1	1	1	1
swCalprmAxisSet	X	X	X							1	1	1	1
swComparisonVariable													
swDataDependency													
swHostVariable													
swImplPolicy	X	X	X	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution	X	X	X	0..1									
swInterpolationMethod	X	X	X	0..1						0..1	0..1	0..1	0..1
swIsVirtual													
swPointerTargetProps													
swRecordLayout	X	X	X	0..1				0..1		1	1	1	1
swRefreshTiming	X	X	X	0..1	0..1			0..1	0..1				
swTextProps	X	X	X					1					
swValueBlockSize	X	X	X	1									
unit	X	X	X	0..1	0..1			0..1	0..1			0..1	0..1
valueAxisDataType	X	X	X		0..1					0..1	0..1	0..1	0..1
Other Attributes below the Root Element													
element: ApplicationRecordElement	X	X	X		1..*								
element: ApplicationArrayElement	X	X	X			1							
ApplicationArrayElement.array- SizeSemantics	X					0..1							
ApplicationArrayElement.maxNum- berOfElements	X					1							

Hint The detailed explanation regarding data types and categories are described in Document "Software Component Template" [TPS_SWCT] Table 5.8, AR 4.2.1

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 for a BSW module because it additionally contains technically 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.

BSWMD can be illustrated using two 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 of the model view and the file partitioning. This knowledge helps for the understanding of the description of the use cases and the location of the specific elements. Therefore both approaches are covered by this rule set.

First a small overview of common aspects of BSWMD is given in chapter [8.1 "Common Aspects of BSWMD" p. 290](#). Here some rules regarding the design and style of a BSWMD ARXML file are listed. The next chapter [8.2 "BSWMD Model View" p. 294](#) focuses on the model view containing general rules for the single model layers of a BSWMD file. The last chapter [8.3 "BSWMD Use Cases" p. 312](#) explains the different use cases provided by 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 AUTOSAR tag definition above should be written as a 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):

```

05   <?xml version...>
<AUTOSAR xmlsn:sxi=....>

10   <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>

20  <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 on the ARPackage structure are specified in [Chapter 10 "Rule Set: AUTOSAR Package Structure" p. 393](#). The relevant rules for BSWMD files are the rules [\[ARPac_41\] p. 402](#) and [\[ARPac_43\] p. 403](#) 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. 508](#). 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-PACKAGES>
      <AR-PACKAGE>
        <!-- Specifiy here the BSWMD specific ARPackages -->
        <!-- like BswModuleDescription, BswImplementation -->
        <!-- ApplicationDataTypes, BswModuleEntries, ... -->
        ...
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
```

Example for the ARPackage structure for the non AUTOSAR based module rba_Hugo:

```

<AR-PACKAGES>
  <AR-PACKAGE>
```

```

05   <SHORT-NAME>RB</SHORT-NAME>
10   <AR-PACKAGES>
15     <AR-PACKAGE>
20       <SHORT-NAME>RBA</SHORT-NAME>
25         <AR-PACKAGES>
30           <AR-PACKAGE>
35             <!-- Specifiy here the BSWMD specific ARPackages -->
40               <!-- like BswModuleDescription, BswImplementation -->
45                 <!-- ApplicationDataTypes, BswModuleEntries, ... -->
50                 ...
55               </AR-PACKAGE>
60             </AR-PACKAGES>
65           </AR-PACKAGE>
70         </AR-PACKAGES>
75       </AR-PACKAGE>
80     </AR-PACKAGES>
85   </AR-PACKAGES>
90

```

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 shall only be comments and have no character of documentation. Everything between the comment markers are ignored from the tools processing ARXML files. Using comments is optional. Example:

```

<AR-PACKAGES>
  <AR-PACKAGE>
    <!-- This is the AR-Package for the BSW module Adc -->
    <SHORT-NAME>AUTOSAR_Adc</SHORT-NAME>
    ...

```

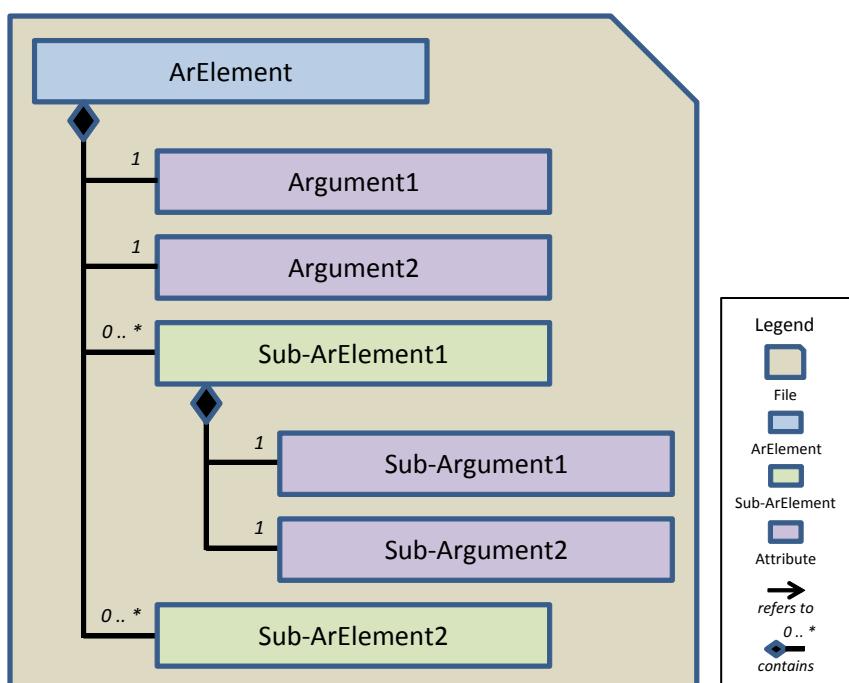
Rule BSWMD_Common_005: Order of Tag Sequences Within BSWMD Files

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 only accepts 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 are explicitly mentioned if needed.

In [Figure 57](#) a common example is given and after that the figure is translated to ARXML syntax.

Figure 57 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>

```

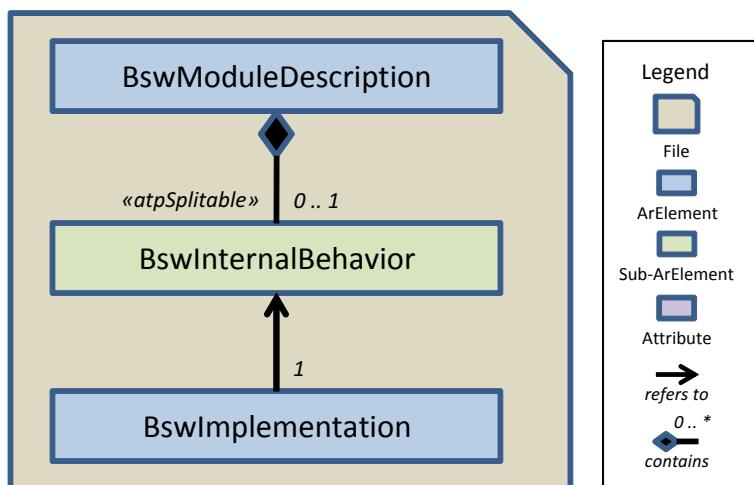
It is not possible to swap Argument1 and Argument2, also it is not possible to swap Sub-ArElement1 and Sub-ArElement2.

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 58](#).

Figure 58 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 focus on 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 characteristics of the BSW module (properties like software version and AUTOSAR version) are described and documented.

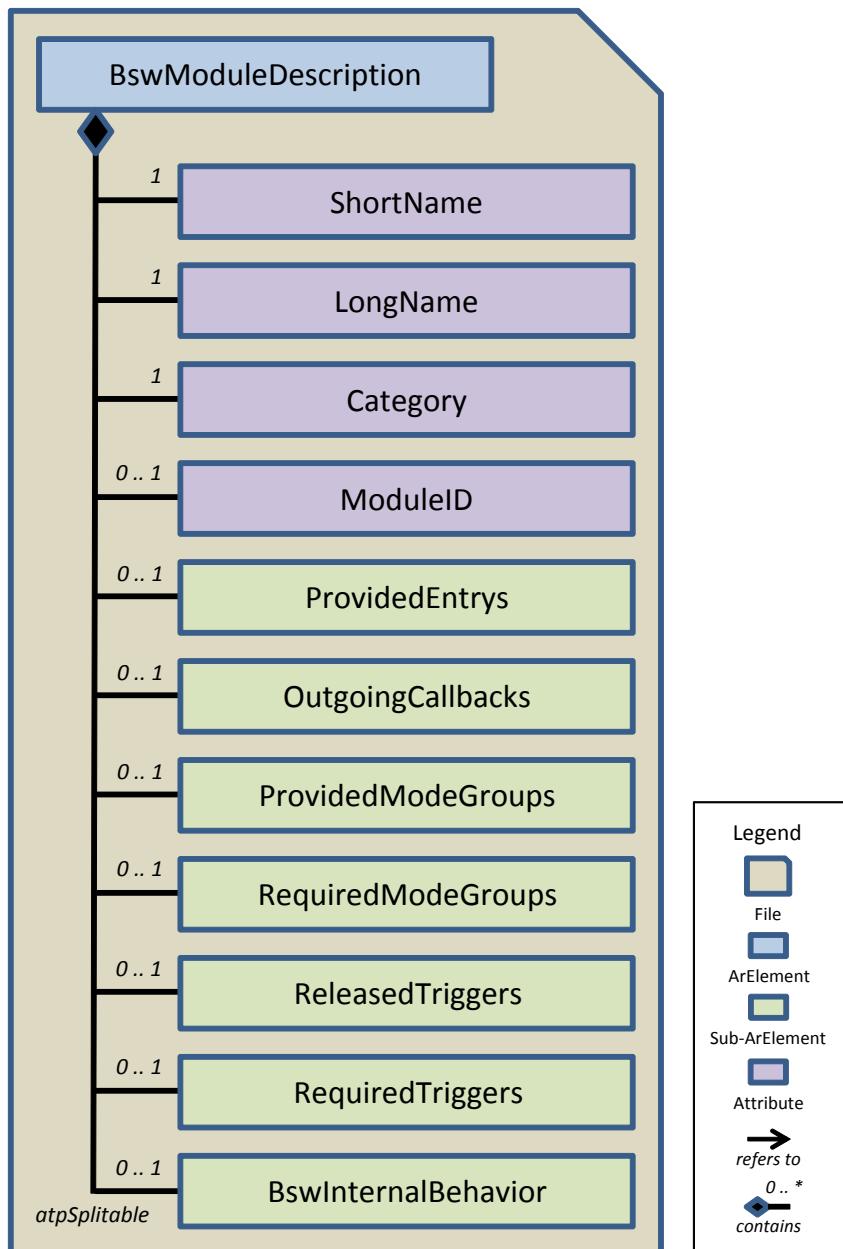
The multiplicity between the *BswModuleDescription* and the *BswInternalBehavior* is given by 0 .. 1 from point of the model view. This implies that the *BswInternalBehavior* is optional. But if parts of the *BswInternalBehavior* are needed (e.g. for the definition of measurement and calibration data, *BswEvents*, *BswModuleEntitys* or *BswModes*, 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 the same instances of a *BswInternalBehavior* based on the same *BswModuleDescription*. This kind of instantiation is currently not followed up by CUBAS. Having only a single instance of a *BswInternalBehavior* is the current focus of the description of the BSWMD layers. Finally, note that the term "behavior" has been chosen in analogy to the same 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 59* an overview of a *BswModuleDescription* is given. Only those elements are shown which are currently on focus and are described within the BSW Coding Guideline. Updates are done gradually.

Figure 59 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>
            <LONG-NAME>
                <!-- Rule [BSWMD_ModuleDesc_002] -->
                <!-- Rule [BSWMD_ModuleDesc_003] -->

```

```

05      <L-4 L="EN">ADC Driver</L-4>
</LONG-NAME>
<CATEGORY>BSW_MODULE</CATEGORY>
<MODULE-ID>123</MODULE-ID>

10     <!-- Use Case specific contents: -->
<PROVIDED-ENTRYS>
  ...
</PROVIDED-ENTRYS>
<OUTGOING-CALLBACKS>
  ...
</OUTGOING-CALLBACKS>
<PROVIDED-MODE-GROUPS>
  ...
</PROVIDED-MODE-GROUPS>
<REQUIRED-MODE-GROUPS>
  ...
</REQUIRED-MODE-GROUPS>
<RELEASED-TRIGGERS>
  ...
</RELEASED-TRIGGERS>
<REQUIRED_TRIGGERES>
  ...
</REQUIRED_TRIGGERES>

15     <!-- Internal behavior -->
<INTERNAL-BEHAVIOR>
  <BSW-INTERNAL-BEHAVIOR>
    ...
  </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIOR>
</BSW-MODULE-DESCRIPTION>
  ...
</ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>

```

```

<!-- Rule [BSWMD_ModuleDesc_004] -->
<!-- Rule [BSWMD_ModuleDesc_005] -->

<!-- Provided/Eported Entrys -->
<!-- Refer to Chapter 8.3.4.4.1 p. 365 -->
<!-- Outgoing Callbacks -->
<!-- Refer to Chapter 8.3.4.4.2 p. 367 -->
<!-- Provided/Eported Modes -->
<!-- Refer to Chapter 8.3.4.4.1 p. 365 -->
<!-- Required/Imported Modes -->
<!-- Refer to Chapter 8.3.4.4.2 p. 367 -->
<!-- Released/Exported Triggers -->
<!-- Refer to Chapter 8.3.4.5.1 p. 375 -->
<!-- Required/Imported Triggers -->
<!-- Refer to Chapter 8.3.4.5.2 p. 377 -->

<!-- BSW Internal Behavior -->
<!-- Refer to Chapter 8.2.1.2 p. 299 -->

```

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. 292](#)). Otherwise the tool chain cannot interpret the BSWMD file correctly.

45 Rule BSWMD_ModuleDesc_001: ShortName of ARPackage Child Element BswModuleDescription Instruction

50 **Class:** NamingConvention

DerivedFrom: [\[ARPac_14\] p. 397](#)

The ShortName of the ARPackage for the *BswModuleDescription* shall be set to "BswModuleDescriptions".

```

55   ...
<AR-PACKAGE>
  <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-DESCRIPTION>
    ...

```

60 Rule BSWMD_ModuleDesc_002: ShortName of the BswModuleDescription

65 Instruction

Class: NamingConvention

The ShortName of the *BswModuleDescription* shall be identical to the module prefix of the BSW module (but without a VendorId and a VendorApilnfix).

The module prefix shall be built conform to rule [\[CDGNaming_001\] p. 111](#) but without the case that the module prefix contains a VendorId or BOSCH-specific name (VendorApilnfix). Possible existing VendorIds and VendorApilnfixes shall be specified with special tags of the BswImplementation (see rule [\[BSWMDImpl_003\] p. 302](#)) 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. 306](#). 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

Class: NamingConvention

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. 508](#). 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 shall be provided.

Example:

```
<LONG-NAME>
  <L-4 L="EN">ADC Driver</L-4>
</LONG-NAME>
```

The LongName is relevant both 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* (default), *BSW_CLUSTER* (avoid this setting) or *LIBRARY*.

The generic category attribute shall be used for a general classification of a BswModuleDescription as shown in the following table.

Table 42 Selectable Values for Category of BswModuleDescription

Category Value	Explanation
BSW_MODULE	Specifies a single BSW module (default)
BSW_CLUSTER	Specifies a BSW module cluster (Not to be used, see explanation below)
LIBRARY	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 are 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. 508](#).

Definition of the term BSW_CLUSTER: AUTOSAR allows integrating several BSW modules (or even the whole BSW including the AUTOSAR Services) in a single cluster, treating this BSW cluster as one entity. It must be known how the

05 cluster interacts with other modules / clusters in order to integrate it. Tests for clusters must know what parts (operation signatures and configurable functionality) are actually supported by the object under test. A BSW_CLUSTER supports the description of BSW module clusters which implement several BSW modules.

10 Within CUBAS no clusters are implemented because development is made conform to Implementation Conformance Class ICC3 (explanation see below). Every specified BSW module is implemented as single module. There is a clustering of some BSW modules to packages but this is done on file group level and not on implementation level of the BSW modules. Therefore only the categories BSW_MODULE and LIBRARY have to be used while BSW_MODULE is the default.

15 Implementation Conformance Class (ICC)

20 Three implementation conformance classes are distinguished:

ICC1 In an ICC1 cluster the basic software is regarded as a black box. An ICC1 cluster offers a software-component interface and/or an AUTOSAR network interface to provide the functional behavior as specified in the AUTOSAR specifications on ICC3 level.

ICC2 Each ICC2 cluster presents a subset of the clustered ICC3 module's interfaces. ICC2 cluster provides the functional behavior as specified in the AUTOSAR specifications on ICC3 level. The number of Cluster Features in an ICC2 cluster is a subset of the union of the number of features of the clustered ICC3 modules.

ICC3 For ICC3 the AUTOSAR BSW consists of BSW modules as defined in the Basic Software Module List, including the RTE. ICC3 is the highest level of granularity. All Basic Software modules as defined in the BSW module list including the RTE, must comply with the defined interfaces and functionality as specified in their respective Software specification document (SWS).

30 Clusters only appear in ICC1 and ICC2 but not in ICC3. Because CUBAS is conform to ICC3 no clusters are used.

Rule BSWMD_ModuleDesc_005: BSW Module Identifier

35 Instruction

Scope: Software based on an AUTOSAR SWS

Within the BswModuleDescription a module identifier shall be set within the tag <**MODULE-ID**>.

40 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. 508](#).

45 This rule is similar to rule [\[BSW_VersionInfo_003\] p. 152](#). 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.

50

55

60

65

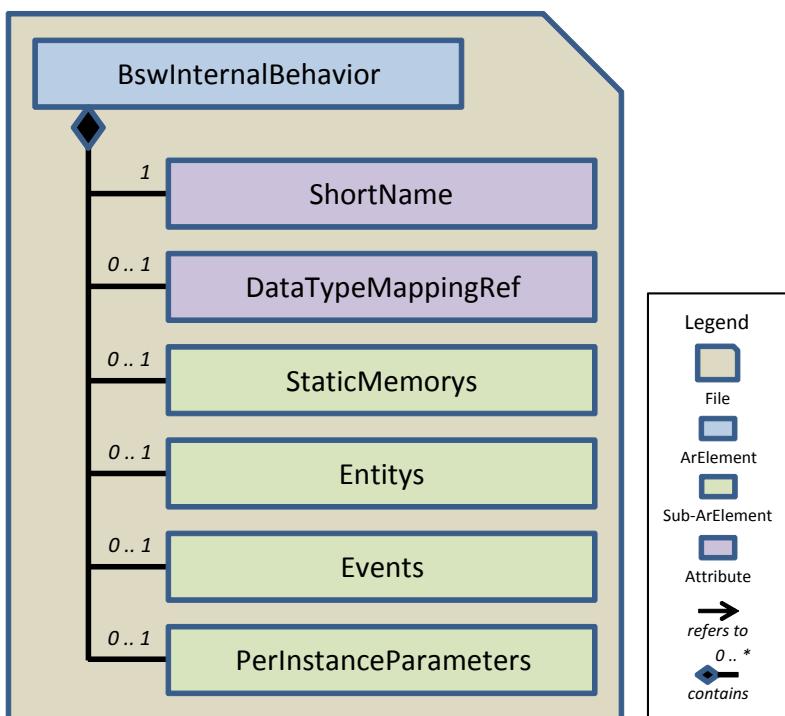
70

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 schedulable entity. More details about those issues are given in the chapter "BSWMD Use Cases" p. 312 .

In *Figure 60* an overview of a *BswInternalBehavior* is given. Only those elements are shown which are currently on focus and are described within the BSW Coding Guideline. Updates are done gradually.

15 Figure 60 Details of *BswInternalBehavior*



20 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 25 following ARXML snippet shows an example for the AUTOSAR BSW module Adc.

```

45 <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>

50  <!-- Use Case specific contents: -->
  <STATIC_MEMORYS>
    ...
  </STATIC_MEMORYS>
  <ENTITYS>
    ...
  </ENTITYS>
  <EVENTS>
    ...
  </EVENTS>
  <PER-INSTANCE-PARAMETERS>
    ...
  </PER-INSTANCE-PARAMETERS>

```

55 <!-- Measurement Data -->
 <!-- Refer to Chapter 8.3.1.1 p. 314 -->

60 <!-- Properties of Code Fragments -->
 <!-- Refer to Chapter 8.3.4.2 p. 347 -->

65 <!-- BSW Module Events -->
 <!-- Refer to Chapter 8.3.4.3 p. 356 -->

<!-- Calibration Data -->
 <!-- Refer to Chapter 8.3.1.2 p. 316 -->

05 </BSW-INTERNAL-BEHAVIOR>
 </INTERNAL-BEHAVIORS>

10 **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. 292](#)). Otherwise the tool chain cannot interpret the BSWMD file correctly.

15 **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.

20 **Rule BSWMD_IntBehav_001:** ShortName of the BswInternalBehavior

Instruction

25 **Class:** NamingConvention

25 The ShortName of the *BswInternalBehavior* shall be set to "BswInternalBehavior".

30 **Rule BSWMD_IntBehav_002:** Reference to DataTypeMapping

30 **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:

- 35 ▶ 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 correctly and fully specified ARPackage path to the DataTypeMapping
 ▶ The DataTypeMapping shall be enclosed from the tags **<DATA-TYPE-MAPPING-REFS>**

40 *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 have to be set. For more details on
 that topic refer to chapter [7 "Rule Set: Data Description" p. 220](#).

45 The DataTypeMappingRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator
 cannot interpret the BSWMD file.

```
50     <DATA-TYPE-MAPPING-REFS>
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">...ARPackagePath...</DATA-TYPE-MAPPING-REF>
    </DATA-TYPE-MAPPING-REFS>
```

55

60

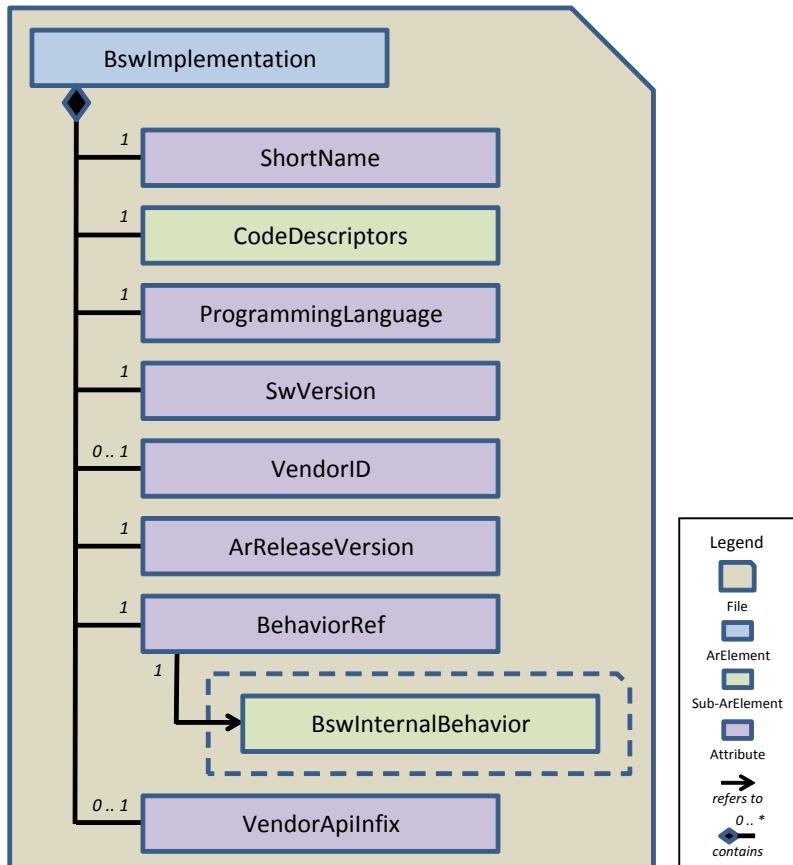
65

70

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 61](#) an overview of a *BswImplementation* is given. Only those elements are shown which are currently on focus and are described within the BSW Coding Guideline. Updates are done gradually.

Figure 61 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>AUTOSAR_Adc</SHORT-NAME>
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswImplementations</SHORT-NAME>          <!-- Rule [BSWMD_Impl_001] -->
    <ELEMENTS>
      <BSW-IMPLEMENTATION>
        <SHORT-NAME>Adc</SHORT-NAME>                      <!-- Rule [BSWMD_Impl_002] -->
        <CODE-DESCRIPTORS>                                    <!-- Rule [BSWMD_Impl_004] -->
          <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>
  
```

```

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] -->
10  <BEHAVIOR-REF DEST="BSW-INTERNAL-BEHAVIOR">           <!-- Rule [BSWMD_Impl_008] -->
    /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior
</BEHAVIOR-REF>
<VENDOR-API-INFIX>SpecificName</VENDOR-API-INFIX>  <!-- Rule [BSWMD_Impl_003] -->
</BSW-IMPLEMENTATION>
15  </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>

```

20 **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. 292](#)). Otherwise the tool chain cannot interpret the BSWMD file correctly.

25 **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

Class: NamingConvention

DerivedFrom: [\[ARPac_14\] p. 397](#)

35 The ShortName of the ARPackage for the *BswImplementation* shall be set to "BswImplementations".

```

...
<AR-PACKAGE>
  <SHORT-NAME>BswImplementations</SHORT-NAME>
  <ELEMENTS>
    <BSW-IMPLEMENTATION>
    ...

```

Rule BSWMD_Impl_002: ShortName of the BswImplementation

Instruction

Class: NamingConvention

50 The ShortName of the *BswImplementation* shall be identical to the module prefix of the BSW module.

55 The module prefix shall be built conform to rule [\[CDGNaming_001\] p. 111](#) but without the case that the module prefix contains a VendorId or BOSCH specific name (VendorApiInfix). Possible existing VendorIds and VendorApiInfixes shall be specified with special tags of the *BswImplementation* (see rule [\[BSWMD_Impl_003\] p. 302](#)) 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: Only relevant for BSW driver modules which are based on AUTOSAR specifications and are providing multiple instances (number of implementation > 1)

65 A vendor identification shall be specified with tag **<VENDOR-ID>**.

65 Additionally a specific name shall be specified as VendorApiInfix using the tag **<VENDOR-API-INFIX>**.

05 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 VendorApilnfix 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 VendorApilnfix shall be different.

10
15 But in contrast to the definition of the module prefix on C language level (the third use case of rule [\[CDGNaming_001\] p. 111](#)) the ARXML-based description provides additional tags to specify the VendorId and the VendorApilnfix. 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 generates complete module prefixes based on the given ShortName, VendorId and VendorApilnfix.

20 For BOSCH the Vendor-Id has to be set to the value "6" as it is specified by HIS. The VendorApilnfix is an additional name which has to be specified conform to the commitment specified in rule [\[CDGNaming_001\] p. 111](#). It is not allowed to specify a VendorId and VendorApilnfix 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 implemented. In all other cases both tags shall not be set.

25 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:

```
30 <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>
```

35 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. This is done if the code is provided as object code (which is the default setting for the *RTE generator*) if no code descriptor is set. But usually BSW modules are delivered as source code. Therefore it is necessary to provide the code descriptor with the category SWSRC. This reduces 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.

50

Rule BSWMD_Impl_005: Documentation of Programming Language

55 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 the 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. 19](#)) the tag <PROGRAMMING-LANGUAGE> has to be set always to "C".

60 Rule BSWMD_Impl_006: Provision of Software Version Number

65 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.

05 The software version number shall be updated directly before a new version of the software is released.

10 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)

15 Increasing a more significant digit of a version number resets all less significant digits.

Example:

```
<SW-VERSION>1.0.0</SW-VERSION>
```

20 The software version number shall also be conform to the vendor-specific versioning which is used in the SCM.

25 **Hint** There is a similar rule to provide version information also in the module header file: Rule [\[BSW_VersionInfo_004\]](#) p. 153. 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

30 **Scope:** Software based on an AUTOSAR SWS

35 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.

40 The AUTOSAR version information shall be updated when the module supports a new version of an AUTOSAR specification.

45 The AUTOSAR version information shows the version of the AUTOSAR specification which an appropriate implementation of a BSW module is based on.

50 Example:

```
<AR-RELEASE-VERSION>4.0.3</AR-RELEASE-VERSION>
```

55 **Hint** There is a similar rule to provide version information also in the module header file: Rule [\[BSW_VersionInfo_004\]](#) p. 153. 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

55 **Instruction** A reference to the *BswInternalBehavior* (with a correctly and fully specified ARPackage path) shall be specified within the *BswImplementation* using the tag **<BEHAVIOR-REF>** containing the destination type "BSW--INTERNAL-BEHAVIOR".

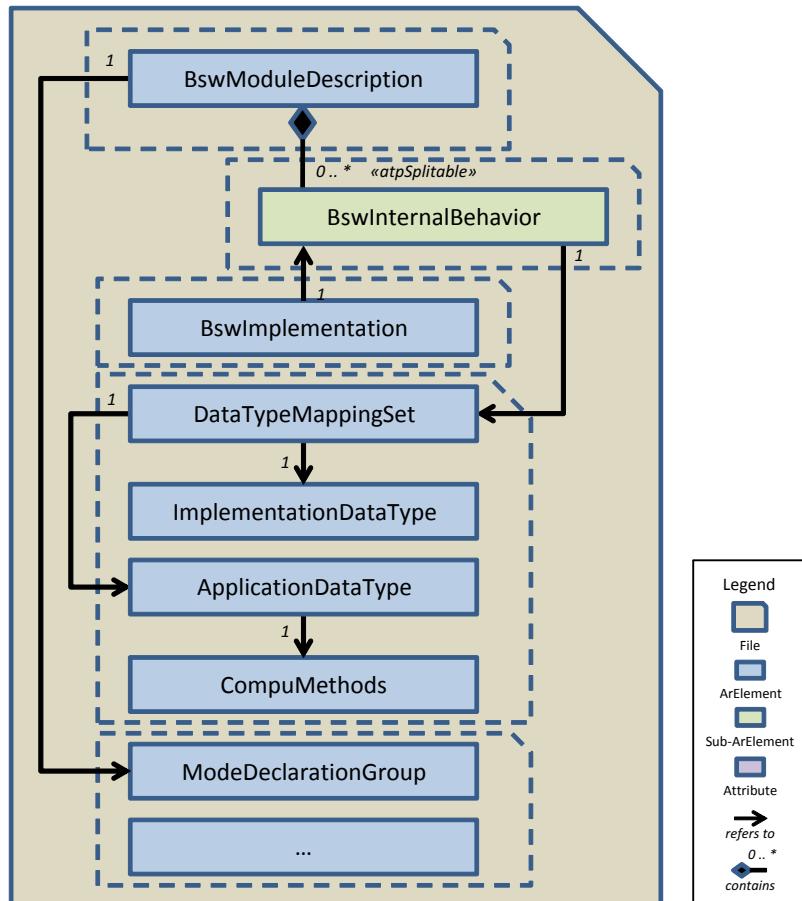
60 The reference is the link between the *BswInternalBehavior* and *BswImplementation* which is also illustrated in [Figure 58](#) p. 294.

65 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>
```

05

8.2.2 Splitting of BSWMD Files

10
15
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 be generally split into different files. But it is not the goal to have many BSWMD files for a single BSW module. A split shall only be done if there is a need to do so e.g. if a use case demands it. Additionally the split of BSWMD files is limited to some restrictions which are explained in this chapter.20
25
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. Typically aggregations cannot be split because they connect internal sub-parts of existing ArElements. But there are exceptional cases where a split is explicitly allowed. Such cases are marked as «*atpSplittable*». This means that it is possible to distribute the aggregated elements over several physical files. References instead are using ARPackage paths to establish the connection between two elements. The path points to the element which should be related. Therefore the path has to be correctly and completely specified. Is this the case then the related element could be located in another file.30
35
Figure 62 shows a rough overview of the file splitting context containing the different relations (But not all ArElements and relations are shown). The figure does 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.30
Figure 62 Example for Splitting of BSWMD Files65
Based on a BSWMD file containing the *BswModuleDescription* it is possible to split further elements to separate files. [Table 43](#) gives an overview and links to the related rules specifying the details. A split shall be done as less as possible

and only if there is a technical reason. Without a technical reason the listed elements shall be located together with the BswModuleDescription in a single file. Rule [\[BSWMD_Split_001\]](#) handles that point.

Table 43 Overview of Splittable Elements

Splittable Element	Description	Related rule
BswInternalBehavior	The BswInternalBehavior is aggregated to the BswModuleDescription with an aggregation relation. Here AUTOSAR allows a split because the aggregation is marked as «atpSplittable».	[BSWMD_Split_002] and [BSWMD_Split_003]
BswImplementation	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.	[BSWMD_Split_004] and [BSWMD_Split_005]
ARPackage Elements	ARPackage Elements can generally be split because they have generally a «atpSplittable» flag.	[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 any 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-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** -->

```

05      </BSW-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
...
10     </BSW-MODULE-DESCRIPTION>
...
15     </ELEMENTS>
</AR-PACKAGE>
</AR-PACKAGES>
```

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 shows 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>
        ...
30      <!-- Specify all relevant attributes and elements -->
      <!-- of the BswModuleDescription here -->
      <!-- Details are specified in Chapter 8.2.1.1 p. 295 -->
        ...
      </BSW-MODULE-DESCRIPTION>
        ...
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
```

BSWMD file containing the BswInternalBehavior:

```

40 <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>
            ...
50          <!-- Specify all relevant attributes and elements -->
          <!-- of the BswInternalBehavior here -->
          <!-- Details are specified in Chapter 8.2.1.2 p. 299 -->
            ...
          </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 sub-elements 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 «*atpSplittable*» and therefore all elements of the *BswInternalBehavior* can only be 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 independent from an ECU configuration and exist permanently. Since 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 could depend on ECU configuration values.

Details about the contents of the *BswImplementation* are given in chapter [8.2.1.2 "BswInternalBehavior" p. 299](#).

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* into a separate 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 58 p. 294](#)). Therefore the *BswImplementation* is self-contained and could be split to a separate file. A use case could be if multiply instantiated BSW driver modules are available. In this case multiple *BswImplementations* exist because the *VendorApilnfix* needs to be unambiguously specified (see rule [\[BSWMD_Impl_003\] p. 302](#)). But in all other cases, 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. 301](#).

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. 301](#).

Rule BSWMD_Split_006: Splitting of Other BSWMD Specific ARPackages to Single BSWMD Files

Instruction *ARPackage*s 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* itself cannot be split to different BSWMD files.

More details about the content of such AR Packages could be found in chapters 7 "Rule Set: Data Description" p. 220 and 8.3 "BSWMD Use Cases" p. 312 .

30

35

40

45

50

55

89

65

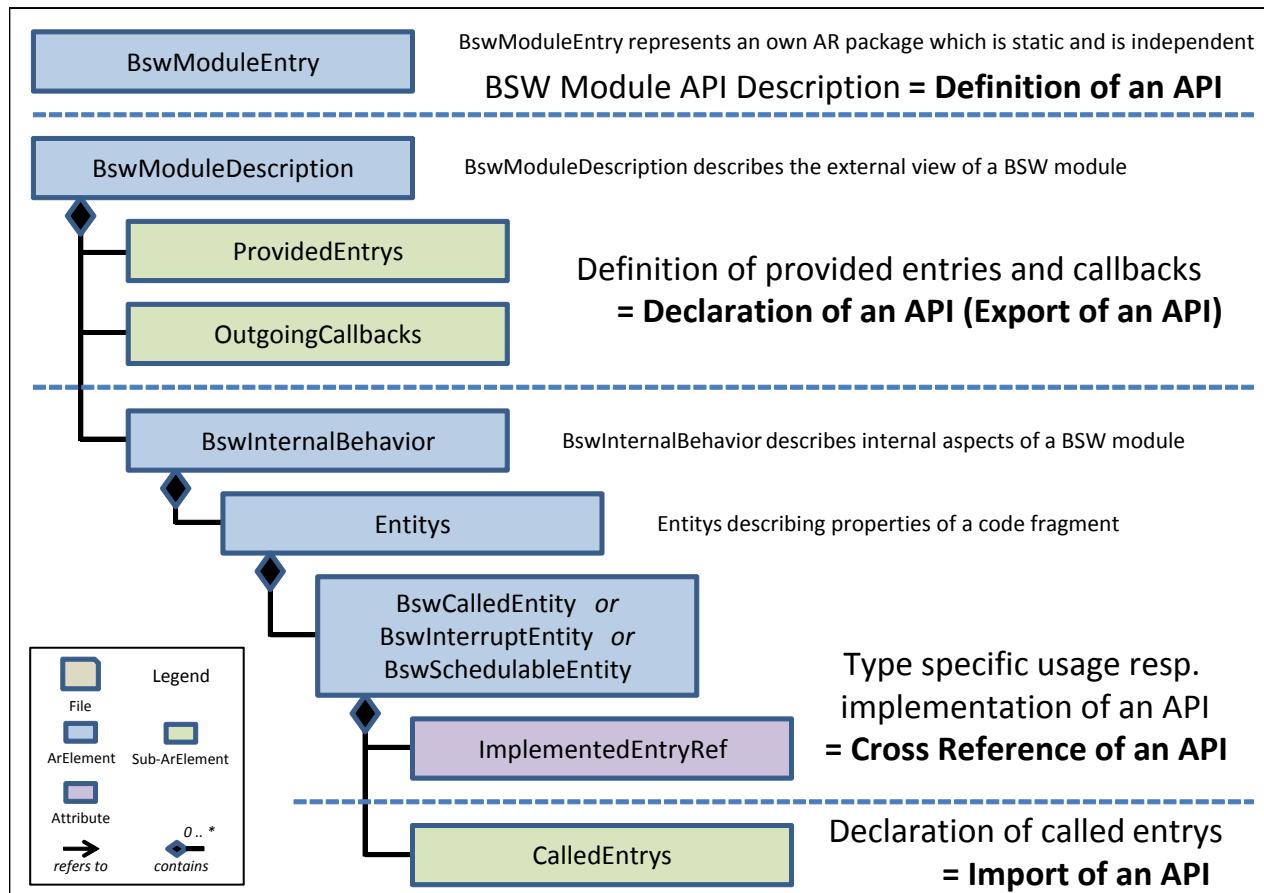
CDG-SMT | BSW Coding Guideline | Volker Kairies | CDG-SMT/ESM1 | 1.10 | 2016-01-31 | released for CDG-SMT | 2016-02-03 14:37:32 | 4.7.1 | © Robert Bosch GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties.

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

8.2.3 API Description and Visibility from Model Point of View

To describe APIs of a BSW module several elements are involved (BswModuleEntry, special tags of the BswModuleDescription and the BswModuleEntitys as part of the BswInternalBehavior). Similar to the C language there are definitions and declarations which represent the different characteristics of an API. Within a BSWMD file both the kind of implementation is specified and the export and import relation is documented. In [Figure 63](#) an overview of the different elements which are involved in the description of an API (based on the model view) is shown. This figure focuses on the relevant tag items and is not complete regarding a valid ARXML description.

Figure 63 Overview of Elements Which are Involved in the Description of an API



Model Level: Definition of an API

The basic element of description of an API is the BswModuleEntry. It is used to model the signature of a C-function call. The BswModuleEntry is located in an own AR package and represents therefore a static and independent superset of all APIs of a BSW module. How a BswModuleEntry has to be specified is described in the use case chapter "["BSW Module API Description \(BswModuleEntry\)" p. 326](#)" .

Model level: Declaration of an API (Export of an API)

A BswModuleEntry is referred as ProvidedEntry or OutgoingCallback within the BswModuleDescription. This is a mandatory reference to declare the API as exported API. Here the BswModuleDescription represents the external view of a BSW module. More information about it can be found in chapter "["BswModuleDescription" p. 295](#)". The ProvidedEntry corresponds to the declaration of an exported BSW module API to be used by other BSW modules. (Outgoing)Callbacks are APIs which are declared from a BSW Module and which are called if another BSW module requires it. Both elements, the ProvidedEntries and the OutgoingCallbacks, are hence relevant to declare a BswModuleEntry as "real" API of a BSW module. How to specify the declaration of an API is explained in chapter "["Export of a BswModuleEntry" p. 345](#)" .

Model level: Cross Reference of an API

05 Within the `BswInternalBehavior` `BswModuleEntitys` have to be specified. A `BswModuleEntity` describes additional properties of a `BswModuleEntry` which is then classified as called, interrupt or schedulable entity. Therefore an implemented BSW API, described as `BswModuleEntry`, is extended with information related to the call of the API, described as `BswModuleEntity`. There is some kind of cross reference between a `BswModuleEntity` and a `BswModuleEntry` to combine all information for a complete description of the characteristics of a BSW API. Such information are typical properties of internal aspects of a BSW module API and therefore they are described as part of the `BswInternalBehavior` (more details about the `BswInternalBehavior` can be found in chapter "["BswInternalBehavior" p. 299](#)"). How to specify a `BswModuleEntity` is explained in chapter "["Properties of a Code Fragment \(BswModuleEntity\)" p. 347](#)" .

10

15 Model level: Import of an API

15 As special case for called APIs the import relation can be stated. Doing that is optional because it has no tool relevant impact. It could only be used to analyze the call chain among several modules in order to setup a proper scheduling.

20 Details for this point are described in the chapter for `BswModuleEntitys` "["Properties of a Code Fragment \(BswModuleEntity\)" p. 347](#)" .

25

Hint regarding file splitting and dependency to ECU configuration

30 In chapter "["Splitting of BSWMD Files" p. 305](#)" restrictions are described regarding the split of BSWMD files. These restrictions have now also effects on the description and visibility of BSW module APIs. Because different BSWMD elements are involved also different BSWMD files could be used. The `BswModuleEntry` represents an own AR package and can be provided in a separate BSWMD file as static superset of defined BSW module APIs (see Rule [\[BSWMD_Split_006\] p. 308](#)). The declaration of APIs is done by `ProvidedEntry` and `OutgoingCallback` items which are part of the `BswModuleDescription`. Both elements are not specified as "atpSplittable" and therefore all declarations of APIs have to be located in the same BSWMD file containing the `BswModuleDescription`. If the export of APIs is dependent on ECU configuration the configuration dependent ones and the static ones have to be generated into the same BSWMD file. A distinction between static and non-static declarations located to different BSWMD files is not possible. The same constraint also applies to the definition of the `BswModuleEntitys`. The different kinds of entities are not declared as "atpSplittable" and have to be located in the same BSWMD file (even if some of them are dependent on ECU configuration). The only possibility is to split the `BswInternalBehavior` as mother element of `BswModuleEntitys` to a separate BSWMD file (conform to Rule [\[BSWMD_Split_002\] p. 306](#)).

40

8.2.4 Differences between BSWMD and SWCD

45 An ASW component and a BSW module uses different kinds of ARXML files to describe its context specific methods and use cases. In normal case not all use cases and methods are needed within a BSW module which are available on ASW level. 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 (this kind of communication is not required to be specified within a BSWMD file). It is not a secret that with a BSWMD file the same things can be described as with a SWCD file which is typically used for ASW. But there are two file formats available to clearly separate 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 for example the following use cases:

- 55
- ▶ 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

60 The listed use cases are relevant if a BSW module represents an AUTOSAR Service component, ECU Abstraction component or Complex Driver Components. All other use cases can be handled by a BSWMD ARXML file only and therefore the BSWMD is the standard file for the data description of a BSW module. 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, [\[G_ArCaDe\]](#)) which describes all use cases of the application software ASW.

65

05

8.3 BSWMD Use Cases

10

The present rule set describes which parts of the BSWMD shall be currently used and how they shall be used. In general the following use cases can be described within the BSWMD:

- 15
-
- 10
-
- 20
-
- 25
-
- 30
-
- 35
-
- 40
-
- 45
-
- 50
-
- 55
-
- 60
-
- 65
-
- 70
-
- ▶ BSW Measurement and Calibration Support (*)
 - ▶ Scheduling of BSW via *RTE* (*)
 - ▶ BSW Documentation
 - ▶ BSW Interface Elements (*)
 - *BswModuleEntry*: BSW Module API Description
 - *BswModuleEntity*: Properties of a Code Fragment
 - *BswEvents*: Activation of *BswScheduledEntitys*
 - *BswModes*: Controlling Activities of BSW Modules
 - *BswTriggers*: Triggering of BSW Modules
 - ▶ 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 Guideline.

8.3.1 BSW Measurement and Calibration Support

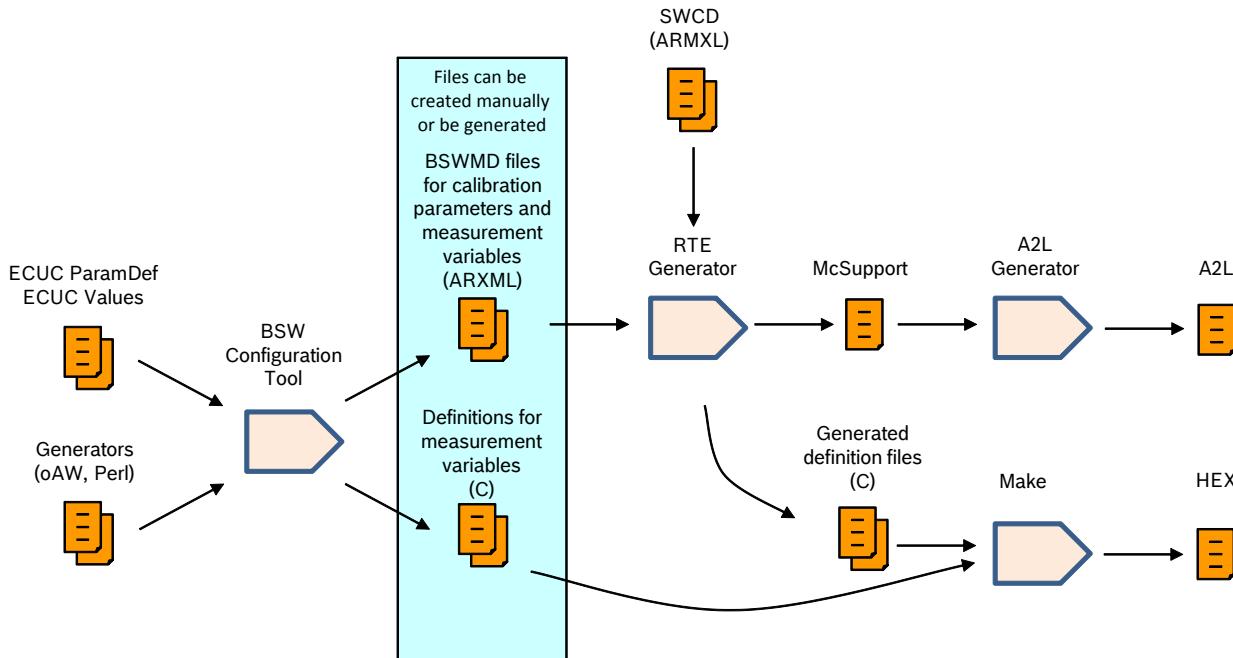
Starting with AUTOSAR release 4.0.3 there is a possibility to describe calibration parameters in BSWMD files very similar to application software in SWCD files. Previous AUTOSAR releases did not have a sufficient support for calibration parameters of BSW modules. The relevant point is that now calibration parameters can be defined as PerInstanceParameters as part of the *BswInternalBehavior*. Then, 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 out 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 *StaticMemorys* 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. However, 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 by calibration tools. The C files (generated by 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 64* 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 by scripts of the ECU configuration process is also illustrated.

Figure 64 Data Flow for BSW Measurement and Calibration Support



8.3.1.1 Definition of Measurement Variables and Measurement Points

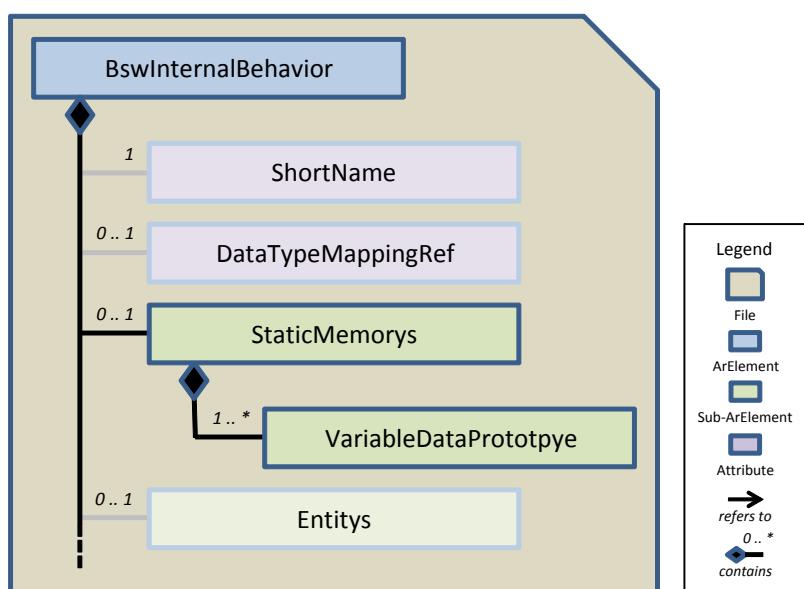
This chapter defines how measurement variables / points have to be defined. Both elements can be measured from an application tool. A measurement variable is an normal internal variable, where values can be written and read from the software. A measurement point is a special element which is especially set from the software to be observed from the application tool.

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 7.2.5 "VariableDataPrototype \(Measurement Points\)" p. 253](#). [Figure 65](#) shows at which 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. 299](#). The figure also shows that the *StaticMemory* has to be set only once if measurement variables shall be defined and then inside the *StaticMemorys* multiple definitions of *VariableDataPrototypes* can be set.

Figure 65 Overview of the 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-MEMORIES>
      <VARIABLE-DATA-PROTOTYPE>          <!-- StaticMemorys used for measurement variables -->
        ...
        <!-- First measurement variable -->
        <!-- Details are in Chapter 7.2.5 p. 253 -->
      </VARIABLE-DATA-PROTOTYPE>
      <VARIABLE-DATA-PROTOTYPE>          <!-- Second measurement variable -->
        ...
        <!-- Details are in Chapter 7.2.5 p. 253 -->
      </VARIABLE-DATA-PROTOTYPE>
      ...
      <!-- And so on -->
    </STATIC-MEMORY>
  <ENTITYS>
```

```

05      ...
 64    </ENTITYS>
 65    ...
 66  </BSW-INTERNAL-BEHAVIOR>
 67</INTERNAL-BEHAVIORS>
```

10 Rule BSWMD_MCSupport_002: Definition of Measurement Variables in C Files

15 **Instruction** The representation of measurement variables or measurement points 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 the definitions in the module BSWMD file (regarding data type and name of the measurement variable). The measurement variable or measurement points shall be defined as global variable and not as static variable.

20 For measurement variables or measurement points the part in the BSWMD file and the respective part in the C file shall match together. The SW developer of the BSW module is responsible to ensure that. The variable definition for the measurement variable or measurement point must be a global definition and the keyword static shall not be used (even if the definition in the ARXML file uses a STATIC-MEMORYS structure).

25 To define the measurement variables or measurement point 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. 75](#). For measurement variables special MemMap keywords are provided. Also for measurement points specific memory section shall be used which allows only a read access.

30 Example for the representation of a measurement variable in a module's C file:

```

35  /* Memory location for 32 bit measurement variable (which is cleared after each reset): */
 64 #define MODULE_START_SEC_INTERNAL_VAR_NO_INIT_32
 65 #include "Module_MemMap.h"
 66 uint32 Module_MeasurementPoint;
 67 #define MODULE_STOP_SEC_INTERNAL_VAR_NO_INIT_32
 68 #include "Module_MemMap.h"
```

40 Rule BSWMD_MCSupport_004: Naming Convention for Measurement Variables

45 Instruction

Class: NamingConvention

45 The ShortName of a measurement variable shall be conform to the following convention: <ComponentPrefix>_<pp><DescriptiveText>.

50 With

- 55 ▶ <ComponentPrefix> which represents the module prefix which has to be conform to rule [\[CDGNaming_001\] p. 111](#)
- ▶ <pp> which represents a physical and logical type which can be picked up from appendix ["Physical and Logical Types" p. 511](#)
- ▶ <DescriptiveText> is the description of the measurement variable (Camel case without underlines)

55 A measurement variable is a global variable which can be written and read within the ECU software. A measurement variable has no special postfix at the end of the name.

60 Rule BSWMD_MCSupport_008: Naming Convention for Measurement Points

65 Instruction

Class: NamingConvention

65 The ShortName of a measurement point shall be conform to the following convention: <ComponentPrefix>_<pp><DescriptiveText>_MP.

With

- ▶ <ComponentPrefix> which represents the module prefix which has to be conform to rule [\[CDGNaming_001\] p. 111](#)
- ▶ <pp> which represents a physical and logical type which can be picked up from appendix ["Physical and Logical Types" p. 511](#)
- ▶ <DescriptiveText> is the description of the measurement variable (Camel case without underlines)
- ▶ _MP is the characteristic suffix for a measurement point

The difference between a measurement variable and a measurement point is that measurement points are strictly read only variables from point of application tool and strictly write only variables from point of software development. Only the measurement and calibration tool chain is allowed to read measurement points. The software on the ECU has no read access to the measurement points. Here only values are written to a measurement point. Measurement points are implemented by the location of them to specific memory sections.

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 off 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* has 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. 220](#). Finally meaningful descriptions have 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

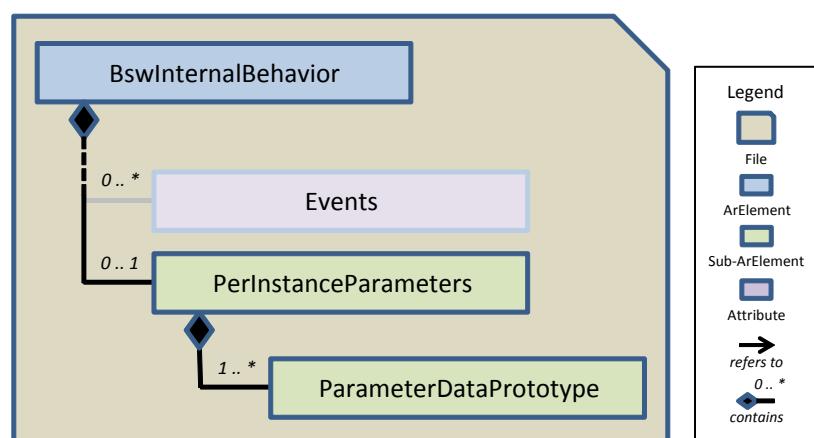
Instruction Calibration parameters which shall exist in the *A2L* file shall be defined using *PerInstanceParameters* as part of the *BswInternalBehavior* in a BSWMD file. The definition using *ConstantMemorys*, which is still available in AUTOSAR 4.0.3, shall NOT be used.

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.

05 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.
 10 How a ParameterDataPrototype has to be specified is described in [Chapter 7.2.2 "ParameterDataPrototype \(Calibration Parameters\)" p. 246](#). 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. 299](#).
 Figure 66 shows all of the details.

15 **Hint** The reason to use PerInstanceParamters is that the RTE generates the entries in the *McSupport* file and also generates the relevant entries in the Rte.c file. Both aspects shall be addressed by the software developer. With ConstantMemorys only entries in the MCSupport file are created but no typedefs in the C code. At last the RTE also considers DSERAP concepts for calibration.

20 Figure 66 Overview of Specification of PerInstanceParameters



25 Example for the definition of calibration parameters as part of the BswInternalBehavior:

```

40 <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>
      <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 7.2.2 p. 246](#) -->
 !-- Second calibration parameter -->
 !-- Details are in [Chapter 7.2.2 p. 246](#) -->
 !-- And so on -->

60 Rule BSWMD_MCSupport_005: Naming Convention for Calibration Parameters

65 Instruction

Class: NamingConvention

The ShortName of a calibration parameter shall be conform to the following naming convention: <ComponentPrefix> - <pp><DescriptiveText> _ <EX>

05
With

- ▶ <ComponentPrefix> which represents the module prefix which has to be conform to rule [\[CDGNaming_001\] p. 111](#)
- ▶ <pp> which represents a physical and logical type which can be picked up from appendix "Physical and Logical Types" [p. 511](#)
- ▶ <DescriptiveText> is the description of the calibration parameter
- ▶ <EX> is characteristic suffix for a calibration parameter and is conform to the following table.

15
Table 44 Suffixes for Calibration Parameter

Suffix <EX>	Description
C	Calibration parameter
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

30
Hint Typically within BSW modules curves, maps and axis are not used.35

Rule BSWMD_MCSupport_006: Accessing a Calibration Parameter

40

Instruction

Class: NamingConvention

To access a calibration parameter the following interface shall be used: SchM_CData_<ComponentPrefix>[_<vi>_<ai>]-<Name>().

50
Where

- ▶ <ComponentPrefix> is the prefix of the BSW module which has to be conform to rule [\[CDGNaming_001\] p. 111](#)
- ▶ [_<vi>_<ai>] is omitted where <vi> is the vendorId of the calling BSW module and <ai> is the vendorApilnfix of the calling BSW module
- ▶ <Name> is the ShortName of the PerInstanceParameter, rule [\[BSWMD_MCSupport_003\]](#)

55
The SchM_CData interface is generated by 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 the following code:

```
#include "SchM_Hugo.h"

uint8 Hugo_CalValue_u8;

Hugo_CalValue_u8 = SchM_Hugo_Hugo_stValue_C();
```

60
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

Scheduling of functions is a main feature of an embedded system. Such that BSW functions can be scheduled within an operation system some elements have to be specified within a BSWMD file. An explanation for that point is given in sub chapter [8.3.2.1 "Mandatory Elements for Scheduling"](#). Based on these elements the Basic Software Scheduler Manager (SchM), a part of the RTE generator, provides all relevant information that a function can be scheduled, refer to sub-chapter [8.3.2.2 "BSW Scheduler Manager \(SchM\)"](#). For BSW Service Modules, ECU Abstraction components and Complex Driver Components in addition to a BSWMD file also a SWCD file is required. A mapping between the SwcInternalBehavior and the BswInternalBehavior has to be made to coordinate the API generation and the scheduling mechanism. Details are described in [8.3.2.3 "Mapping Between BSW and a Corresponding SWC"](#).

Additionally the sequence order of scheduled functions is relevant. To calculate the correct sequence of processes/runnables in an OS task the Execution Order Constraint has to be specified. How this has to be done is described in chapter [9 "Rule Set: Execution Order Constraint" p. 383](#). Further information about scheduling can be found in chapter Execution Architecture of the Architecture Guideline [\[G_Arch\]](#).

8.3.2.1 Mandatory Elements for Scheduling

To describe a schedulable function the following three BSWMD elements are mandatory:

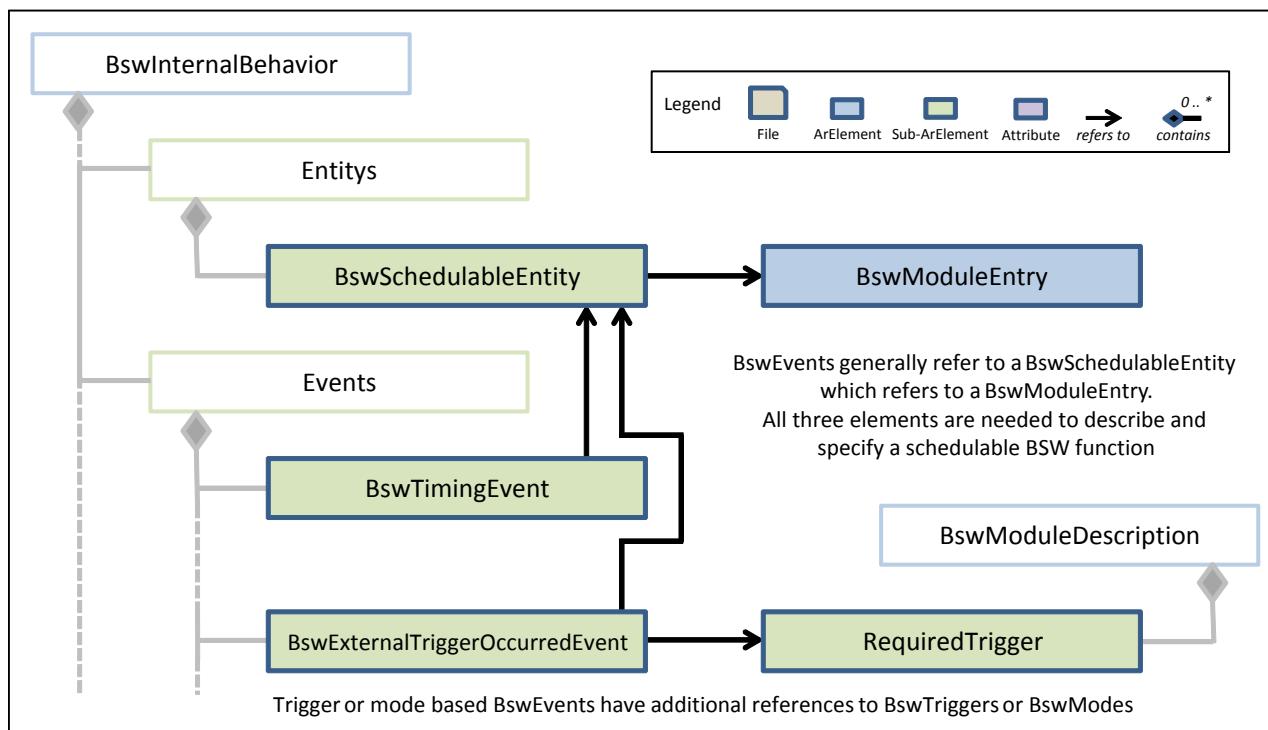
BswModuleEntry: To describe the function prototype. Refer to [Chapter 8.3.4.1 "BSW Module API Description \(BswModuleEntry\)" p. 326](#).

BswSchedulableEntity: To specify additional properties of the function and to specify the relation to other functions, modes and triggers. Refer to [Chapter 8.3.4.2 "Properties of a Code Fragment \(BswModuleEntity\)" p. 347](#).

BswEvent: To specify the conditions that a function can be scheduled (based on modes or triggers) or solely to specify the timing period when a function has to be called. How to specify a *BswEvent* is described in [Chapter 8.3.4.3 "BswEvents" p. 356](#).

Figure 67 roughly depicts the relationship between *BswEvents* (here *BswTimingEvent* and *BswExternalTriggerOccurredEvent*), *BswSchedulableEntitys* and *BswModuleEntries*. When a scheduled function shall be invoked by a *BswTimingEvent* only a proper defined *BswTimingEvent* (timing period shall be specified) has to be referenced from the *BswSchedulableEntity*. For more complex events (e.g. *BswExternalTriggerOccurredEvent* or *ModeSwitchEvents*) additional references (to *BswTriggers* and *BswModes*) have to be properly defined.

Figure 67 Overview Scheduling Use Case



For each BSW module which contains *BswSchedulableEntitys* the RTE generator generates a Module Interlink Header File with the name SchM_<Module>.h. It has to be included in a BSW module in order to access and use the SchM elements.

How to integrate a *BswSchedulableEntity* to a task of the operation system is out of scope of the BSW Coding Guideline. This context is described in the integration guideline, see [\[G_Integ\]](#).

Rule BSWMD_Schedule_001: Mandatory Elements for the Scheduling Use Case

Instruction To be able to schedule a BSW function the following elements shall be specified and related to each other: *BswModuleEntry*, *BswSchedulableEntity*, *BswEvent*.

The following sub-chapters describe the scheduling relevant elements as well as their relationships:

- ▶ How to specify a *BswModuleEntry* is described in [Chapter 8.3.4.1 "BSW Module API Description \(BswModuleEntry\)" p. 326](#).
- ▶ How to specify a *BswScheduledEntity* is described in [Chapter 8.3.4.2 "Properties of a Code Fragment \(BswModule-Entity\)" p. 347](#). (Remark: Only *BswScheduledEntitys* are relevant for the scheduling use case. *BswCalledEntitys* and *BswInterruptEntitys* have another use case which is described in the linked chapter above.)
- ▶ How to specify a *BswEvent* is described in [Chapter 8.3.4.3 "BswEvents" p. 356](#). (Remark: Different kinds of *BswEvents* can activate a *BswScheduledEntity*. The simplest one is a *BswTimingEvent*, other events are based on *BswModes* or *BswTriggers*. Details can be found in the linked chapter above.)

Rule BSWMD_Schedule_002: Dependency Between BswEvent and BswScheduledEntity

Instruction If a BSW function has to be scheduled in different tasks, or is called based on different conditions every time, a particular *BswEvent* shall be defined. The dependency between a *BswEvent* and a *BswScheduledEntity* is a 'n to 1' relationship.

The 'n to 1' relationship between a *BswEvent* and a *BswScheduledEntity* is an important but also critical point. Typically the definition of a function is done by the BSW module developer. Also the location of a function to different tasks is done by an integrator. This means that the definition of a *BswModuleEntity* and the definition of a *BswEvent* could be done from different persons and at different times. Both elements are part of the *BswInternalBehavior*. However, in [Chapter 8.2.2 "Splitting of BSWMD Files" p. 305](#) the *BswInternalBehavior* can't be split. This means that the definition of *BswScheduledEntitys* cannot be located in a single ARXML file while the *BswEvents* are located in another file. If a BSW module developer does not provide all needed *BswEvents* an integrator has no chance to schedule a function properly. It is not allowed that an integrator touches an existing BSWMD file of a BSW module and changes its content e.g. by entering an additional *BswEvent*.

Basically the explained point is a conceptual gap from AUTOSAR. Currently discussions are triggered that for example the *BswEvents* are also specified as *atpSplittable*. But until now such a change of the concept is not yet decided and thus the existing method has to be applied. The BSW module developer and the integrator have to coordinate the needs for schedulable functions and have to provide all relevant elements. If for example a *BswEvent* is missing an update of the BSW module has to be made by the BSW module developer.

8.3.2.2 BSW Scheduler Manager (SchM)

The handling of the BSW Scheduler Manager has to be explained in the next version of the BSW Coding Guideline.

8.3.2.3 Mapping Between BSW and a Corresponding SWC

Remark: This chapter will be reworked with the next version of the BSW Coding Guideline.

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 *BswScheduledEntity* (from a BSW)
Means: Model that a Runnable Entity is equal to a *BswScheduledEntity*
- ▶ *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

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>SwcBswMappings</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-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>
```

Hint The BswEntityRef and the SwcRunnableRef have 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 BswEntityRef and the SwcRunnableRef are written in three lines only for clearness.

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>
```

Hint The SwcBswMappingRef 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 SwcBswMappingRef is written in three lines only for clearness.

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:

```

05   <?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">
10  <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>
</AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>

```

Hint The BswModeGroupRef, ContextPPortRef and TargetModeGroupRef have 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 BswModeGroupRef, ContextPPortRef and TargetModeGroupRef are written in three lines only for clearness.

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:

```

65  <?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns="http://autosar.org/schema/r4.0"

```

```

05      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-TRIGGERS>
<SWC-BSW-SYNCHRONIZED-TRIGGER>
    <BSW-Trigger-Ref DEST="Trigger">
        <!-- Refer to the Snippet of a (External) Trigger (for BSW) -->
    </BSW-Trigger-Ref>
    <SWC-Trigger-IRef>
        <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
            <!-- 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>
</AR-PACKAGES>
</AUTOSAR>

```

Hint The BswTriggerRef, ContextPPortRef and TargetTriggerRef have 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 BswTriggerRef, ContextPPortRef and TargetTriggerRef are written in three lines only for clearness.

8.3.3 BSW Documentation

How the documentation should be made in detail is not on focus of the BSW Coding Guideline. Rudimentary tags items like LongName, Desc or Introduction are mentioned e.g. in the description of BSW Module APIs (see chapter "["BSW Module API Description \(BswModuleEntry\)" p. 326](#)) but the internal tag structure and internal details are specified in the documentation guideline (see [\[DocuGuide\]](#)).

8.3.4 BSW Interface Elements

This chapter describes different kinds of interface elements of a BSW module: *BswModuleEntry*, *BswModuleEntity*, *BswModuleEvents*, *BswModes* and *BswTriggers*.

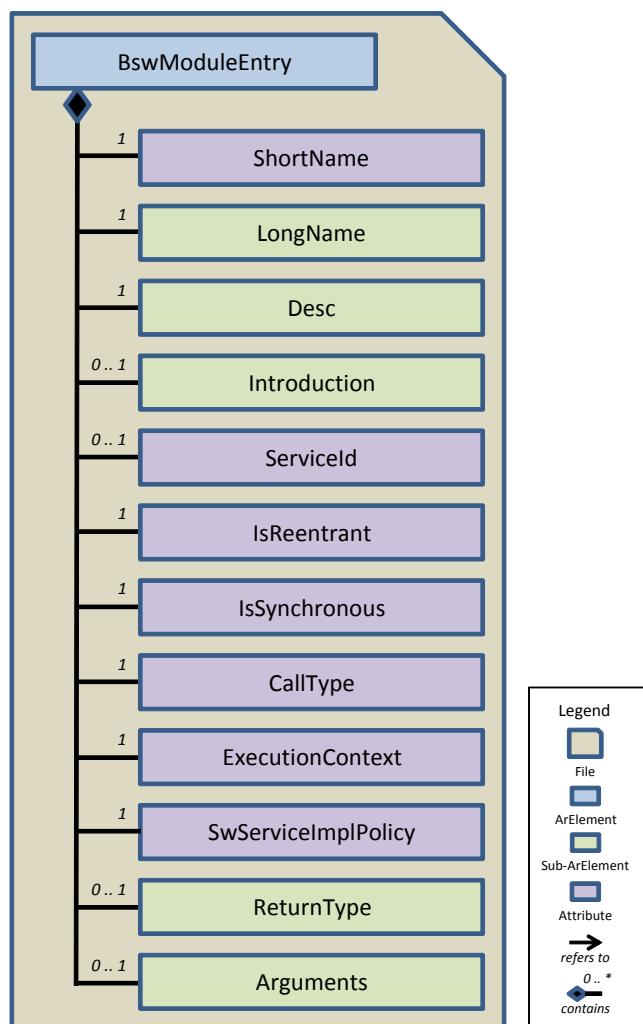
8.3.4.1 BSW Module API Description (BswModuleEntry)

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 [8.3.4.1.1 p. 326](#). How a return value and arguments of an API are specified is defined in chapter [8.3.4.1.2 p. 332](#).

8.3.4.1.1 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. [Figure 68](#) illustrates the details of a *BswModuleEntry*. Details for the definition of the return value and the arguments are defined in [Chapter 8.3.4.1.2 p. 332](#).

Figure 68 Details of a BswModuleEntry



The following ARXML snippet shows an example for the *BswModuleEntry* of the API *Adc_ReadGroup*.

```

05 <AR-PACKAGES>
<AR-PACKAGE>
  <SHORT-NAME>BswModuleEntrys</SHORT-NAME>      <!-- Rule [BSWMD_Entry_001] -->
  <ELEMENTS>
    <BSW-MODULE-ENTRY>
      <SHORT-NAME>Adc_ReadGroup</SHORT-NAME>      <!-- First BswModuleEntry -->
      <LONG-NAME>
        ...
      </LONG-NAME>
      <DESC>
        ...
      </DESC>
      <INTRODUCTION>
        ...
      </INTRODUCTION>
      <SERVICE-ID>0x04</SERVICE-ID>      <!-- Rule [BSWMD_Entry_003] -->
      <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>
        UNSPECIFIED
      </EXECUTION-CONTEXT>
      <SW-SERVICE-IMPL-POLICY>
        STANDARD
      </SW-SERVICE-IMPL-POLICY>
      <RETURN-TYPE>          <!-- Refer to Chapter 8.3.4.1.2 p. 332 -->
        ...
      </RETURN-TYPE>
      <ARGUMENTS>
        ...
      </ARGUMENTS>
    </BSW-MODULE-ENTRY>
    <BSW-MODULE-ENTRY>      <!-- Second BswModuleEntry -->
      ...
    </BSW-MODULE-ENTRY>
    ...
  </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. 292](#)). Otherwise the tool chain cannot interpret the BSWMD file correctly.

50 Rule BSWMD_Entry_001: ShortName of the ARPackage Kind Element BswModuleEntry Instruction

Class: NamingConvention

DerivedFrom: [\[ARPac_14\] p. 397](#)

The ShortName of the *ArPackage* for the *BswModuleEntry* shall be set to "BswModuleEntrys".

Example:

```

...
<AR-PACKAGE>
  <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
  <ELEMENTS>
    <BSW-MODULE-ENTRY>
    ...

```

Rule BSWMD_Entry_002: ShortName of the BswModuleEntry

Instruction

Class: NamingConvention

The ShortName of the *BswModuleEntry* shall be set identical to the name of the C-function (but without a VendorId and a VendorApiInfix).

The name of the C function and therefore the ShortName of the *BswModuleEntry* shall be conform to the naming convention (rule [\[CDGNaming_003\] p. 113](#)). 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 are specified with special tags of the BswImplementation (see rule [\[BSWMD_Impl_003\] p. 302](#)) and are not set in the ShortName of the *BswModuleEntry*.

Rule BSWMD_Entry_009: LongName of the BswModuleEntry

Instruction The tag item <LONG-NAME> shall be used to specify a headline as explanation of the *BswModuleEntry*.

The long name is targeted to human readers and acts as a headline. It is relevant for documentation purpose and contains a short explanation of a *BswModuleEntry*.

Example for the definition of a long name:

```
<BSW-MODULE-ENTRY>
  <SHORT-NAME>Adc_ReadGroup</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Adc service to read a group conversion results</L-4>
  </LONG-NAME>
  ...
</BSW-MODULE-ENTRY>
```

Hint The internal tag structure of a long name is specified in the documentation guideline (see [\[DocuGuide\]](#)).

Rule BSWMD_Entry_010: Description of the BswModuleEntry

Instruction The tag item <DESC> shall be used to specify a short description of a *BswModuleEntry*.

This represents a general but brief (one paragraph) description of a *BswModuleEntry*. It is relevant for documentation purpose. The description should help a human reader to identify or understand the aim of a *BswModuleEntry*. More elaborate documentation (in particular how the *BswModuleEntry* is built or used) should be specified (in addition to the description) within the introduction tag structure (see Rule [\[BSWMD_Entry_011\]](#)).

Example for the definition of a description:

```
<BSW-MODULE-ENTRY>
  <SHORT-NAME>Adc_ReadGroup</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Adc service to read a group conversion results</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">Reads the group conversion result of the last completed conversion.</L-2>
  </DESC>
  ...
</BSW-MODULE-ENTRY>
```

Hint The internal tag structure of a description is specified in the documentation guideline (see [\[DocuGuide\]](#)).

Rule BSWMD_Entry_011: Introduction of the BswModuleEntry

Instruction The tag item **<INTRODUCTION>** should be used to specify a more detailed explanation of a *BswModuleEntry*.

In addition to the description of a *BswModuleEntry* (see Rule [\[BSWMD_Entry_010\]](#)) a more detailed explanation of a *BswModuleEntry* can be given with the introduction. The introduction is a kind of "documentation block". Within the introduction different kinds of documentation items like notes, lists or simple paragraphs can be used to provide a detailed explanation of a *BswModuleEntry*. The introduction is still only optional and could be used after consideration of effort and necessity.

Example for the definition of an introduction:

```

<BSW-MODULE-ENTRY>
  <SHORT-NAME>Adc_ReadGroup</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Adc service to read a group conversion results</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="EN">Reads the group conversion result of the last completed conversion.</L-2>
  </DESC>
  <INTRODUCTION>
    <NOTE NOTE-TYPE="CAUTION">
      <P>
        <L-1 L="EN">The group channel values are stored in ascending channel number order.</L-1>
      </P>
    </NOTE>
    <P>
      <L-1 L="EN">Adc_ReadGroup reads the raw converted values without further scaling.</L-1>
    </P>
  </INTRODUCTION>
  ...
</BSW-MODULE-ENTRY>

```

Hint The internal tag structure of an introduction is specified in the documentation guideline (see [\[DocuGuide\]](#)).

Rule BSWMD_Entry_003: Serviceld 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.

The following two settings are possible: *true* or *false*

Explanation of the selectable settings:

Table 45 List of Settings for the 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 be defined 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.

The following two settings are possible: *true* or *false*

Explanation of the selectable settings:

Table 46 List of Settings for the 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.

The following settings are possible: *CALLBACK*, *INTERRUPT*, *REGULAR* or *SCHEDULED*.

The call type denotes how a BswModuleEntry is called. One of the following settings has to be chosen:

Table 47 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. 330 .

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.

The 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 of 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 48 List of Execution Contexts of a BswModuleEntry

Literal	Description
HOOK	Context of an OS "hook" routine
INTERRUPT-CAT-1	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 <i>BswInterruptEntity</i> is implemented by the interrupt service routine, which is directly called from the interrupt vector (not via the OS).
INTERRUPT-CAT-2	Cat2 interrupt routines are controlled by the OS and they are allowed to make OS calls. The <i>BswInterruptEntity</i> is implemented by the interrupt handler, which is called from the OS.
TASK	Task context
UNSPECIFIED	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.

Hint It is not recommended to use interrupts of *INTERRUPT-CAT-1* because interrupts should be under control of the OS.

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.

The 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:

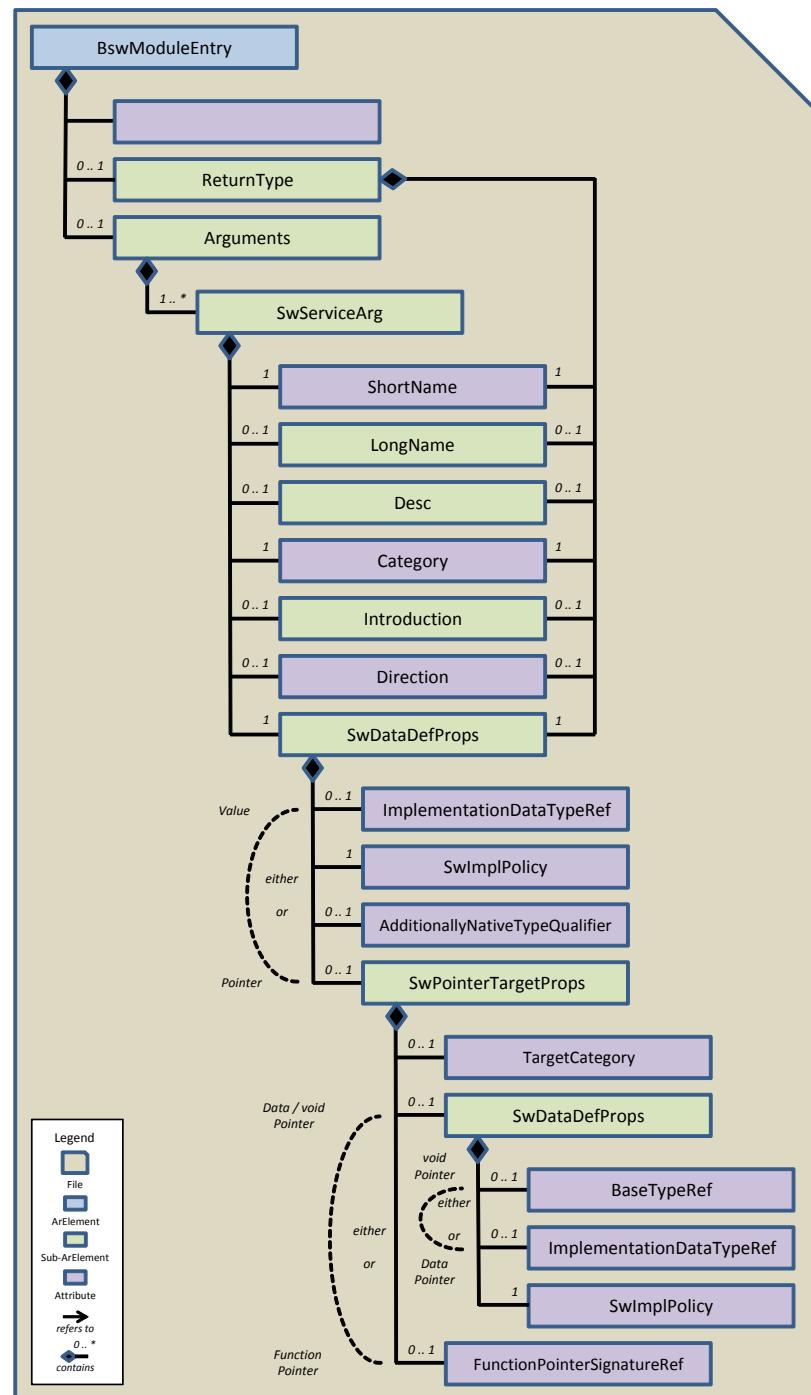
Table 49 List of SwServiceImplPolicy of a BswModuleEntry

Literal	Description
INLINE	Inline service definition. This service is defined using the macro LOCAL_INLINE. For more details refer to chapter 3.2.5 "Abstraction of Inline Functions" p. 81 .
INLINE-CONDITIONAL	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 on the ECU configuration settings)
MACRO	Macro service definition. Additional conditions for the definition of a macro is given in rule [BSWMD_EntryRetArg_012] p. 343 .
STANDARD	Standard service and default value, if nothing is defined

8.3.4.1.2 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 69](#) gives an overview of the tag structure.

Figure 69 Details of a Return Value and Arguments of a BswModuleEntry



The dotted lines in the figure shows 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 the 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

05 an argument is only a value then the ImplementationDataTypeRef within the outer *SwDataDefProps* is applied but the element *SwPointerTargetProps* with all sub-elements and attributes is not used.

10 The following examples and rules are provided to represent the definition of return values and arguments. For the sake of clearness the optional documentation items (LongName, Desc, Introduction) are only shown in the examples for a return value.

Definition of return value, e.g. uint8:

```

<RETURN-TYPE>
  <SHORT-NAME>ReturnValue</SHORT-NAME>          <!-- Rule [BSWMD_EntryRetArg_001] -->
  <LONG-NAME>                                <!-- Rule [BSWMD_EntryRetArg_015] -->
  ...
  </LONG-NAME>                                <!-- Rule [BSWMD_EntryRetArg_016] -->
  <DESC>                                         ...
  </DESC>                                         <!-- Rule [BSWMD_EntryRetArg_003] -->
  <CATEGORY>TYPE_REFERENCE</CATEGORY>        <!-- Rule [BSWMD_EntryRetArg_017] -->
  <INTRODUCTION>                            ...
  </INTRODUCTION>                            <!-- Rule [BSWMD_EntryRetArg_004] -->
  <DIRECTION>OUT</DIRECTION>            <!-- Rule [BSWMD_EntryRetArg_005] -->
  <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>STANDARD</SW-IMPL-POLICY>  <!-- Rule [BSWMD_EntryRetArg_009] -->
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>                <!-- : -->
  </SW-DATA-DEF-PROPS>                          <!-- Rule [BSWMD_EntryRetArg_005] -->
</RETURN-TYPE>
```

40 **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 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>
      <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>
          volatile
        </ADDITIONAL-NATIVE-TYPE-QUALIFIER>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>                <!-- : -->
    </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] -->
    <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>
            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>
</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):

```

<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>
            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">
```

```

05      /AUTOSAR_Platform/SwBaseTypes/void
</BASE-TYPE-REF>
10     <SW-IMPL-POLICY>                      <!-- Rule [BSWMD_EntryRetArg_009] -->
15       STANDARD
16     </SW-IMPL-POLICY>
17     </SW-DATA-DEF-PROPS-CONDITIONAL>
18     </SW-DATA-DEF-PROPS-VARIANTS>
19     </SW-DATA-DEF-PROPS>
20   </SW-POINTER-TARGET-PROPS>
21     </SW-DATA-DEF-PROPS-CONDITIONAL>
22     </SW-DATA-DEF-PROPS-VARIANTS>
23   </SW-DATA-DEF-PROPS>
24 </SW-SERVICE-ARG>
25 ...
26 </ARGUMENTS>
27
28 Hint The BaseTypeRef has to be written in a single line in a productive BSWMD file. Otherwise the RTE generator
29 cannot interpret the BSWMD file. In the example above the BaseTypeRef is written in three lines only for clearness.
30
31 Definition of a function pointer as argument:
32 <ARGUMENTS>
33   <SW-SERVICE-ARG>
34     <SHORT-NAME>ptrFktArg_pfc</SHORT-NAME>          <!-- Rule [BSWMD_EntryRetArg_002] -->
35     <CATEGORY>FUNCTION_REFERENCE</CATEGORY>          <!-- Rule [BSWMD_EntryRetArg_003] -->
36     <DIRECTION>INOUT</DIRECTION>                  <!-- Rule [BSWMD_EntryRetArg_004] -->
37     <SW-DATA-DEF-PROPS>                            <!-- Rule [BSWMD_EntryRetArg_008] -->
38       <SW-DATA-DEF-PROPS-VARIANTS>                 <!-- : -->
39         <SW-DATA-DEF-PROPS-CONDITIONAL>
40           <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY> <!-- Rule [BSWMD_EntryRetArg_009] -->
41             <SW-POINTER-TARGET-PROPS>
42               <FUNCTION-POINTER-SIGNATURE-REF DEST="BSW-MODULE-ENTRY">
43                 /AUTOSAR_Adc/BswModuleEntrys/Adc_ReadGroup
44               </FUNCTION-POINTER-SIGNATURE-REF>
45             <SW-POINTER-TARGET-PROPS>
46               </SW-DATA-DEF-PROPS-CONDITIONAL>
47             </SW-DATA-DEF-PROPS-VARIANTS>                <!-- : -->
48           </SW-DATA-DEF-PROPS>
49         </SW-SERVICE-ARG>
50 ...
51 </ARGUMENTS>
52
53 Hint The FunctionPointerSignatureRef has to be written in a single line in a productive BSWMD file. Otherwise the
54 RTE generator cannot interpret the BSWMD file. In the example above the FunctionPointerSignatureRef is written in
55 three lines only for clearness.
56
57 Hint Note that the order of the elements matters and thus the specified sequence here has to be followed (as defined
58 in rule [BSWMD_Common_005] p. 292). Otherwise the tool chain cannot interpret the BSWMD file correctly.
59
60 Rule BSWMD_EntryRetArg_001: ShortName of a Return Value of a BswModuleEntry
61
62 Instruction
63
64 Class: NamingConvention
65 The ShortName of a return value of a BswModuleEntry shall be set to 'ReturnValue'.
66
67 For a return value the ShortName has no functional relevance but the tag <SHORT-NAME> is still a mandatory one.
68 Therefore the dummy-name 'ReturnValue' shall be set.
69
70
71 CDG-SMT | BSW Coding Guideline | Volker Kairies | CDG-SMT/ESM1 | 1.10 | 2016-01-31 | released for CDG-SMT | 2016-02-03 14:37:32 | 4.7.1 | © Robert Bosch
72 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.
```

Rule BSWMD_EntryRetArg_002: ShortName of an Argument of a BswModuleEntry

Instruction

Class: NamingConvention

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 shall be defined conform to rule [\[CDGNaming_006\] p. 116](#).

Rule BSWMD_EntryRetArg_015: LongName of the BswModuleEntry

Instruction The tag item **<LONG-NAME>** may be used to specify a headline as explanation of the return value or the argument of a *BswModuleEntry*.

The long name is targeted to human readers and acts as a headline. It is relevant for documentation purpose and contains a short explanation of the return value or the argument of a *BswModuleEntry*. This tag item is only optional.

Example for the definition of a long name for an argument:

```
<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>Arg1_u8</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">First argument of the function</L-4>
    </LONG-NAME>
    ...
  </SW-SERVICE-ARG>
<ARGUMENTS>
```

Hint The internal tag structure of a long name is specified in the documentation guideline (see [\[DocuGuide\]](#)).

Rule BSWMD_EntryRetArg_016: Description of the BswModuleEntry

Instruction The tag item **<DESC>** should be used to specify a short description of the return value or the argument of a *BswModuleEntry*.

This represents a general but brief (one paragraph) description of the return value or the argument of a *BswModuleEntry*. It is relevant for documentation purpose. The description should help a human reader to identify or understand the aim of a return value or argument. More elaborate documentation (in particular how the return value or argument is built or used) could be specified (in addition to the description) within the introduction tag structure (see Rule [\[BSWMD_EntryRetArg_017\]](#)). By the way the desc item is only an optional one but it could be used if a description can be specified for the specific return value or argument.

Example for the definition of a description for an argument:

```
<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>Arg1_u8</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">First argument of the function</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">The parameter represents an ID to specify a channel.</L-2>
    </DESC>
    ...
  </SW-SERVICE-ARG>
<ARGUMENTS>
```

Hint The internal tag structure of a description is specified in the documentation guideline (see [\[DocuGuide\]](#)).

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).

The 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 50 List of Categories for a Return Value or Argument of a BswModuleEntry

Literal	Description
<i>TYPE_REFERENCE</i>	The element is defined via a 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_017: Introduction of the BswModuleEntry

Instruction The tag item **<INTRODUCTION>** may be used to specify a more detailed explanation of the return value or the argument of a *BswModuleEntry*.

In addition to the description (see Rule [\[BSWMD_EntryRetArg_016\]](#)) a more detailed explanation of the return value or the argument of a *BswModuleEntry* can be given with the introduction. The introduction is a kind of "documentation block". Within the introduction different kinds of documentation items like notes, lists or simple paragraphs can be used to provide a detailed explanation of a *BswModuleEntry*. The introduction is still only optional and could be used after consideration of effort and necessity.

Example for the definition of an introduction an argument:

```

<ARGUMENTS>
  <SW-SERVICE-ARG>
    <SHORT-NAME>Arg1_u8</SHORT-NAME>
    <LONG-NAME>
      <L-4 L="EN">First argument of the function</L-4>
    </LONG-NAME>
    <DESC>
      <L-2 L="EN">The parameter represents an ID to specify a channel.</L-2>
    </DESC>
    <INTRODUCTION>
      <P>
        <L-1 L="EN">Possible range of the IDs:</L-1>
      </P>
      <LIST TYPE="UNNUMBER">
        <ITEM>
          <P>
            <L-1 L="EN">0...10: Low level of IDs.</L-1>
          </P>
        </ITEM>
        <ITEM>

```

```

05      <P>
       <L-1 L="EN">10...99: High level of IDs.</L-1>
     </P>
   </ITEM>
 </LIST>
</INTRODUCTION>
...
</SW-SERVICE-ARG>
<ARGUMENTS>
```

15 **Hint** The internal tag structure of an introduction is specified in the documentation guideline (see [\[DocuGuide\]](#)).

20 Rule BSWMD_EntryRetArg_004: Direction of a Return Type or an Argument of a BswModuleEntry

25 **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.

The following values are possible: *IN*, *INOUT* or *OUT*.

30 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.

35 The direction allows to declare the data flow of the arguments of a *BswModuleEntry*. One of the following settings has to be chosen:

Table 51 List of Direction Values for a Return Value or Argument of a BswModuleEntry

Literal	Description
<i>IN</i>	Return value: Not used for return values. Argument: 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>	Return value: Not used for return values. Argument: 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>	Return value: The only possible direction for a return value Argument: 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.

50 Rule BSWMD_EntryRetArg_005: Definition of a Value as Return Value or Argument of a BswModuleEntry

55 **Instruction** To specify a value as return value or argument of a *BswModuleEntry* the 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 correctly and complete specified 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 [\[BSWMD_EntryRetArg_003\]](#))

65 The reference to an *ImplementationDataType* has to be set as shown in the following example.

```

05   <RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>TYPE_REFERENCE</CATEGORY>           <!-- Category for a value -->
...
10  <SW-DATA-DEF-PROPS>                      <!-- SwDataDefProps tag structure -->
    <SW-DATA-DEF-PROPS-VARIANTS>            <!-- containing two sub tag structure -->
        <SW-DATA-DEF-PROPS-CONDITIONAL>
            <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                /AUTOSAR_Platform/ImplementationDataTypes/uint8  <!-- Referred Impl.DataType -->
            </IMPLEMENTATION-DATA-TYPE-REF>
...
15    </SW-DATA-DEF-PROPS-CONDITIONAL>          <!-- : -->
    </SW-DATA-DEF-PROPS-VARIANTS>            <!-- : -->
</SW-DATA-DEF-PROPS>                      <!-- Closing the SwDataDefProps -->
</RETURN-TYPE> or </SW-SERVICE-ARG>

```

20 **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.

25 In the example above a reference to a centrally defined primitive *ImplementationDataType* was used. It is also possible to refer to a locally specified *ImplementationDataType*. Also this *ImplementationDataType* could be a complex one (array or structure, as described in "[Rule Set: Data Description](#)" p. 220).

30 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.

35 Rule BSWMD_EntryRetArg_006: Definition of a Data Pointer as Return Value or Argument of a BswModuleEntry

40 **Instruction** To specify a data pointer as return value or argument of a *BswModuleEntry* the following settings shall be made:

- 45
- ▶ 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 correctly and 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 [\[BSWMD_EntryRetArg_003\]](#))
- 50

The reference to an *ImplementationDataType* has to be set as shown in the following example.

```

<RETURN-TYPE> or <SW-SERVICE-ARG>
...
55  <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>
...
60    <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 -->

```

```

05           </IMPLEMENTATION-DATA-TYPE-REF>
...
10           </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>

```

15 **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.

20 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 required if attributes of an element shall be specified and references to *ImplementationDataTypes* have to be set. In principle 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 [\[BSWMD_EntryRetArg_009\]](#))).

25 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 69 "Details of a Return Value and Arguments of a BswModuleEntry"](#) p. 332 the 'either ... or' decision between a simple value and the pointers is marked.

30 In the example above a reference to a centrally defined primitive *ImplementationDataType* was used. It is also possible to refer to a locally specified *ImplementationDataType*. Also this *ImplementationDataType* could be a complex one (array or structure, as described in ["Rule Set: Data Description"](#) p. 220).

35 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.

40 Rule BSWMD_EntryRetArg_007: Definition of a Void Pointer as Return Value or Argument of a BswModuleEntry

45 **Instruction** To specify a void pointer as return value or argument of a *BswModuleEntry* the following settings shall be made:

- 50 ▶ 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
- 55 ▶ The destination type of the *ImplementationDataTypeRef* shall be set to 'SW-BASE-TYPE'
- 60 ▶ The path '/AUTOSAR_Platform/SwBaseTypes/void' shall be set
- 65 ▶ 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.

```

65 <RETURN-TYPE> or <SW-SERVICE-ARG>
...
<CATEGORY>DATA_REFERENCE</CATEGORY>          <!-- Category for a data pointer -->

```

```

05 ...
<SW-DATA-DEF-PROPS>
<SW-DATA-DEF-PROPS-VARIANTS>
<SW-DATA-DEF-PROPS-CONDITIONAL>
...
10 <SW-POINTER-TARGET-PROPS>
...
<SW-DATA-DEF-PROPS>
<SW-DATA-DEF-PROPS-VARIANTS>
<SW-DATA-DEF-PROPS-CONDITIONAL>
15 <BASE-TYPE-REF DEST="SW-BASE-TYPE">
    /AUTOSAR_Platform/SwBaseTypes/void      <!-- Ref. to SwBaseType for void -->
</BASE-TYPE-REF>
...
20 </SW-DATA-DEF-PROPS-CONDITIONAL>
</SW-DATA-DEF-PROPS-VARIANTS>
</SW-DATA-DEF-PROPS>
</SW-POINTER-TARGET-PROPS>
</SW-DATA-DEF-PROPS-CONDITIONAL>
25 </SW-DATA-DEF-PROPS-VARIANTS>
</SW-DATA-DEF-PROPS>
</RETURN-TYPE> or </SW-SERVICE-ARG>
```

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.

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 which 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.

40 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* the following settings shall be made:

- 45 ▶ A reference to a *BswModuleEntry* using the tag **<FUNCTION-POINTER-SIGNATURE-REF>** shall be set within Sw-PointerTargetProps which are part of the SwDataDefProps tag structure of the return value or argument definition
- 50 ▶ The destination type of the FunctionPointerSignatureRef shall be set to 'BSW-MODULE-ENTRY'
- 55 ▶ A correctly and complete path to a *BswModuleEntry* shall be set
- 60 ▶ The category of the return value or argument shall be set to 'FUNCTION-REFERENCE' (conform to rule [\[BSWMD-EntryRetArg_003\]](#))

55 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 -->
...
60 <SW-DATA-DEF-PROPS>
<SW-DATA-DEF-PROPS-VARIANTS>
<SW-DATA-DEF-PROPS-CONDITIONAL>
...
<SW-POINTER-TARGET-PROPS>                   <!-- SwPointerTargetProps structure -->
...
65 <FUNCTION-POINTER-SIGNATURE-REF DEST="BSW-MODULE-ENTRY">
    /AUTOSAR_Adc/BswModuleEntries/Adc_ReadGroup   <!-- Reference to a BswModuleEntry -->
```

```

05      </FUNCTION-POINTER-SIGNATURE-REF>
</SW-POINTER-TARGET-PROPS>
</SW-DATA-DEF-PROPS-CONDITIONAL>
</SW-DATA-DEF-PROPS-VARIANTS>
</SW-DATA-DEF-PROPS>
10     </RETURN-TYPE> or </SW-SERVICE-ARG>

```

```

      <!-- Closing the SwPointerTargetProps -->
      <!-- : -->
      <!-- : -->
      <!-- Closing the SwDataDefProps -->

```

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.

Compared with 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>`.

The 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 the context of data pointers. In all other cases (values, function pointers) the implementation policy shall be set to '*STANDARD*'.

The following table provides the description of the implementation policy:

Table 52 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 53](#) gives an overview of the four cases of a data pointer definition. Only in this context the implementation policy '*CONST*' is relevant.

Table 53 Overview of the 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 or const uint8 * ptrArg_pcu8</code>
Constant pointer to constant data	<code>uint8 const * const ptrArg_cpcu8 or const uint8 * const ptrArg_cpcu8</code>

The following example shows 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 -->
...
65    <SW-DATA-DEF-PROPS>                      <!-- Outer SwDataDefProps tag structure -->
...                                         <!-- for the return value or argument -->

```

```

05   <SW-IMPL-POLICY>CONST</SW-IMPL-POLICY>           <!-- Position for a constant pointer      -->
...
<SW-POINTER-TARGET-PROPS>                         <!-- SwPointerTargetProps structure       -->
...
<SW-DATA-DEF-PROPS>                                <!-- Inner SwDataDefProps tag structure -->
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     -->
15   </SW-DATA-DEF-PROPS>                            <!-- Closing the outer SwDataDefProps     -->
</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.

The 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 7.1.5 "ImplementationDataType" p. 237](#).

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`. The category tag shall be set to `MACRO`.

Macros do not have explicit data types for their return types and arguments. Therefore no reference to an `ImplementationDataType` is needed if a return type or argument of a macro is specified. Hence no `SwDataDefProps` has to be set. Only the common arguments like `ShortName`, `Category` and `Direction` shall be set within the definition of a return type or argument of a macro. The category tag shall be set to "MACRO" in this case. Also consider that the `SwServiceImplPolicy` of the owning `BswModuleEntry` has to be set to `MACRO` too (see rule [\[BSWMD_Entry_008\] p. 331](#)).

The following example gives a short overview:

```

<BSW-MODULE-ENTRY>
...
<SW-SERVICE-IMPL-POLICY>MACRO</SW-SERVICE-IMPL-POLICY>
...
<RETURN-TYPE>
  <SHORT-NAME>ReturnValue</SHORT-NAME>
```

```

05   <CATEGORY>MACRO</CATEGORY>
06   <DIRECTION>OUT</DIRECTION>
07   </RETURN-TYPE>
08   <ARGUMENTS>
09     <SW-SERVICE-ARG>
10       <SHORT-NAME>Arg1</SHORT-NAME>
11       <CATEGORY>MACRO</CATEGORY>
12       <DIRECTION>IN</DIRECTION>
13     </SW-SERVICE-ARG>
14     ...
15   </ARGUMENTS>
16 </BSW-MODULE-ENTRY>

```

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.

Within the *BswModuleEntry* it is not allowed to implement an empty return type by providing the type void, i.e. creating a reference to the *SwBaseType void*.

Rule BSWMD_EntryRetArg_014: BswModuleEntry with no Argument

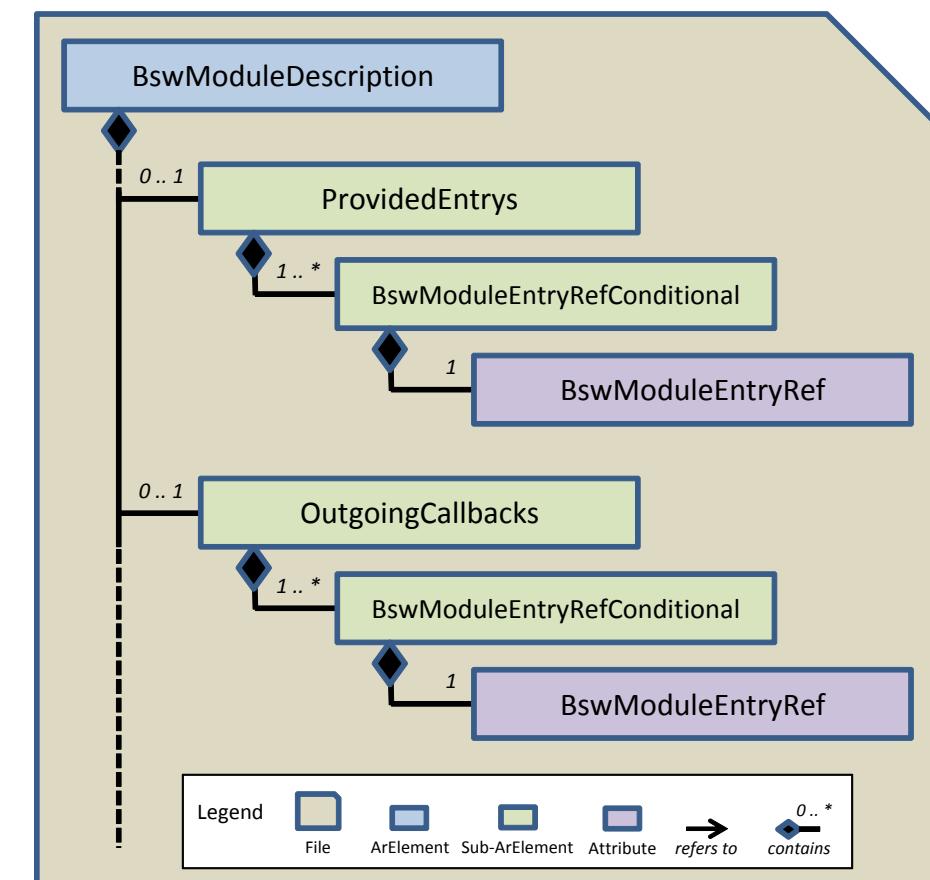
Instruction In case of an empty argument list ("void" in C) no arguments shall be specified within the *BswModuleEntry*.
 The **<ARGUMENT>** tag structure shall not be set.

Within the *BswModuleEntry* it is not allowed to implement an empty argument type by providing the type void, i.e. creating a reference to the *SwBaseType void*.

8.3.4.1.3 Export of a BswModuleEntry

That a BswModuleEntry is visible to the outside of a BSW module an export of a BswModuleEntry has to be made. This is comparable to the declaration of an API. This declaration is done using the tag items ProvidedEntries and OutgoingCallbacks which are part of the BswModuleDescription (an overview of the BswModuleDescription is given in [Chapter 8.2.1.1 "BswModuleDescription" p. 295](#)). There is a differentiation between callbacks and non-callbacks which depends on the category of a BswModuleEntry. Callback functions are exported using the tag item OutgoingCallbacks. All other APIs (also "main"-functions called by the BSW Scheduler) are exported using the tag item ProvidedEntries. In [Figure 70](#) and overview of the relevant tag items to define the declaration of an BswModuleEntry is illustrated.

Figure 70 Details on the Declaration of an BswModuleEntry



The following ARXML snippet shows an example for the declaration of provided BswModuleEntries and callbacks.

```

<BSW-MODULE-DESCRIPTION>
  <SHORT-NAME>Adc</SHORT-NAME>
  ...
  <MODULE-ID>123</MODULE-ID>
  <PROVIDED-ENTRIES>
    <BSW-MODULE-ENTRY-REF-CONDITIONAL>          <!-- Rule [BSWMD_EntryExport_001] -->
      <BSW-MODULE-ENTRY-REF DEST="BSW-MODULE-ENTRY">  <!-- First provided BswModuleEntry -->
        /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
      </BSW-MODULE-ENTRY-REF>
    </BSW-MODULE-ENTRY-REF-CONDITIONAL>
    <BSW-MODULE-ENTRY-REF-CONDITIONAL>          <!-- Second provided BswModuleEntry -->
      ...
    <BSW-MODULE-ENTRY-REF-CONDITIONAL>
      ...
  </PROVIDED-ENTRIES>
  <OUTGOING-CALLBACKS>
    <BSW-MODULE-ENTRY-REF-CONDITIONAL>          <!-- Rule [BSWMD_EntryExport_002] -->
      ...
    </BSW-MODULE-ENTRY-REF-CONDITIONAL>          <!-- First provided Callback -->
  </OUTGOING-CALLBACKS>
</BSW-MODULE-DESCRIPTION>
  
```

```

05   <BSW-MODULE-ENTRY-REF DEST="BSW-MODULE-ENTRY">
      /AUTOSAR_Adc/BswModuleEntries/Adc_Callback1
    </BSW-MODULE-ENTRY-REF>
  </BSW-MODULE-ENTRY-REF-CONDITIONAL>
<BSW-MODULE-ENTRY-REF-CONDITIONAL>      <!-- Second provided Callback -->
10    ...
  <BSW-MODULE-ENTRY-REF-CONDITIONAL>
    ...
</OUTGOING-CALLBACKS>
</BSW-MODULE-DESCRIPTION>

```

15 **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. 292](#)). Otherwise the tool chain cannot interpret the BSWMD file correctly.

20 **Hint** The BswModuleEntryRef 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 BswModuleEntryRef is written in three lines only for clearness.

25 Rule BSWMD_EntryExport_001: Export of a BswModuleEntry of Type 'INTERRUPT', 'REGULAR' or 'SCHEDULED'

30 **Instruction** BswModuleEntrys of call type 'INTERRUPT', 'REGULAR' or 'SCHEDULED' shall be declared as provided entry to be exported outside of a BSW module. This shall be done in the following way:

- 35 ▶ A tag structure shall be created starting with tag item **<PROVIDED-ENTRYS>**
- ▶ As next tag layer the tag item **<BSW-MODULE-ENTRY-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several BswModuleEntrys have to be exported.
- ▶ The reference to a BswModuleEntry shall be set within the tag item **<BSW-MODULE-ENTRY-REF>**
- ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY'
- ▶ A correctly and complete path to a BswModuleEntry shall be set

40 The call type is specified with the BswModuleEntry. For more details refer to rule [\[BSWMD_Entry_006\] p. 330](#).

45 Rule BSWMD_EntryExport_002: Export of a BswModuleEntry of Type 'CALLBACK'

50 **Instruction** BswModuleEntrys of call type 'CALLBACK' shall be declared as outgoing callback to be exported outside of a BSW module. This shall be done in the following way:

- 55 ▶ A tag structure shall be created starting with tag item **<OUTGOING-CALLBACKS>**
- ▶ As next tag layer the tag item **<BSW-MODULE-ENTRY-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several BswModuleEntrys have to be exported.
- ▶ The reference to a BswModuleEntry shall be set within the tag item **<BSW-MODULE-ENTRY-REF>**
- ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY'
- ▶ A correctly and complete path to a BswModuleEntry shall be set

60 The call type is specified with the BswModuleEntry. For more details refer to rule [\[BSWMD_Entry_006\] p. 330](#).

8.3.4.2 Properties of a Code Fragment (BswModuleEntity)

A *BswModuleEntity* is related to a *BswModuleEntry*. While the *BswModuleEntry* describes the API interface of a BSW module (i.e. the signature of a C-function prototype. For more details refer to [Chapter 8.3.4.1 "BSW Module API Description \(BswModuleEntry\)" p. 326](#)) the *BswModuleEntity* describes additional properties of the code fragment. It is mandatory that for each *BswModuleEntity* a *BswModuleEntry* exists because both items belong together. The focus of a *BswModuleEntity* is the representation of a piece of code with an associated set of attributes. For example within a *BswModuleEntity* *BswModes* and *BswTriggers* could be referred which are provided or received from the corresponding *BswModuleEntity*.

The *BswModuleEntity* is a part of the *BswInternalBehavior* which describes the internal aspects of a BSW module (see [Chapter 8.2.1.2 "BswInternalBehavior" p. 299](#)). 'BswModuleEntity' is a common term for an entity. In general three different kinds of *BswModuleEntitys* are distinguished which are shown in [Table 54](#).

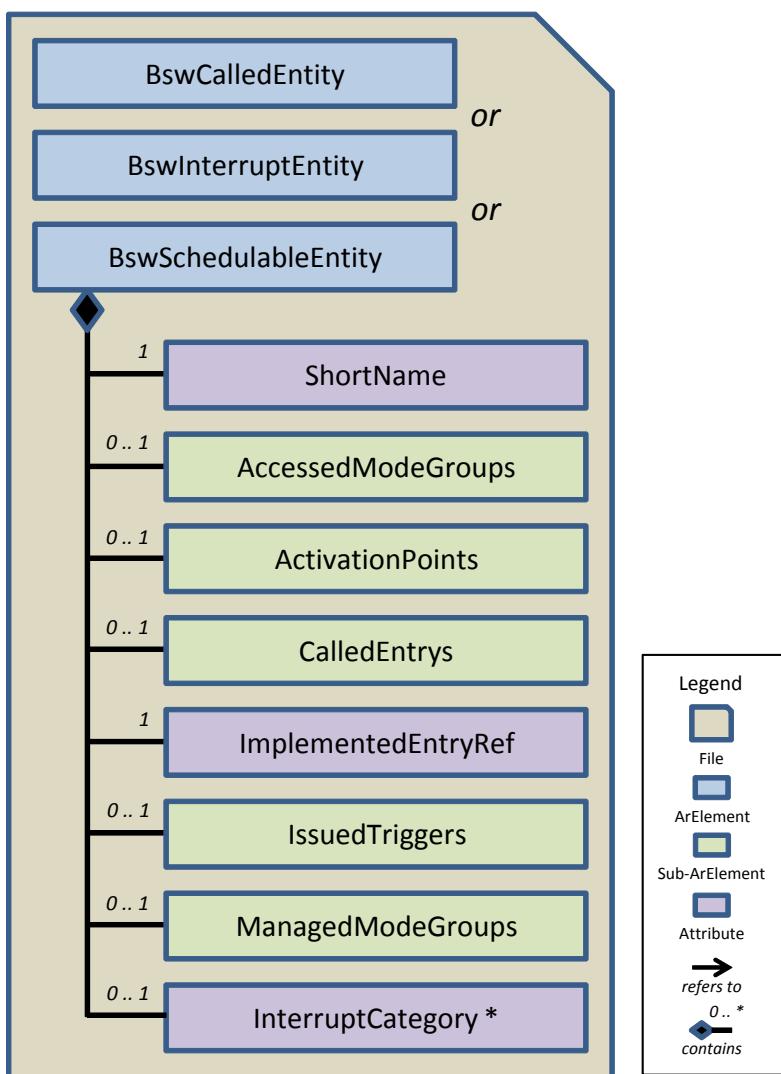
Table 54 Kinds of BswModuleEntitys

Kind of <i>BswModuleEntity</i>	Description
<i>BswCalledEntity</i>	BSW module entity, which implements a 'C' function interface which is directly called by other BSW modules or clusters.
<i>BswInterruptEntity</i>	BSW module entity, which implements an interrupt which is called by an interrupt controller
<i>BswSchedulableEntity</i>	BSW module entity, which is scheduled by the BSW Scheduler (SchM) which is generated by the RTE generator. It implements a so-called "main" function.

It is one of the characteristic of a *BswModuleEntity* that three different ARXML items exist for the three different kinds of entitys (<BSW-CALLED-ENTITY>, <BSW-INTERRUPT-ENTITY>, <BSW-SCHEDULABLE-ENTITY>).

The mandatory elements for defining a *BswModuleEntity* are the short name and a reference to a *BswModuleEntry*. Optional additional references to *BswModes* and *BswTriggers* can be specified (therefore different ARXML tags are available). A complete overview of the structure of a *BswModuleEntity* is given in [Figure 71](#).

Figure 71 Details of BswModuleEntitys



* Only relevant for BswInterruptEntity

The attribute 'InterruptCategory' is only relevant for BswInterruptEntitys. Rule [BSWMD_Entity_009] specifies the handling of that attribute.

The following ARXML code snippet shows an example for the simplest form of a definition of a BswModuleEntity (here a BswCalledEntity is specified of module Adc). The definition of the entitys is a part of the BswInternalBehavior and the tag structure item <ENTITYS>.

```

<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  ...
  <STATIC-MEMORYS>...</STATIC-MEMORYS>

  <ENTITYS>
    <BSW-CALLED-ENTITY>
      <SHORT-NAME>CE_Func1</SHORT-NAME>
      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
      </IMPLEMENTED-ENTRY-REF>
    </BSW-CALLED-ENTITY>
  </ENTITYS>

  <EVENTS>...</EVENTS>
  ...
</BSW-INTERNAL-BEHAVIOR>
  
```

<!-- Rule [BSWMD_Entity_001] -->
 <!-- Rule [BSWMD_Entity_002] -->
 <!-- Rule [BSWMD_Entity_003] -->
 <!-- Rule [BSWMD_Entity_001] -->

05 Hint The ImplementedEntryRef 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 ImplementedEntryRef is written in three lines only for clearness.

10 In [Table 55](#) an overview of the additional references within a *BswModuleEntity* is listed. Specific examples how the additional references have to be set are given in the related rules.

15 Table 55 Overview Additional References within a *BswModuleEntity*

Reference	Description	Related Rule
AccessedMode-Groups	This attribute describes that the entity receives a mode switch from other BSW modules. This information is used to configure the BSW Scheduler to be able to create a BSW Scheduler API.	[BSWMD_Entity_004]
ActivationPoints	An activation point is used by the <i>BswModuleEntity</i> to activate one or more internal triggers .	[BSWMD_Entity_005]
CalledEntries	This attribute is used to declare which entry of another module (or the same module) is called by this code entity (usually by a C function call). Note that this is not a mandatory information in order to be able to integrate a module, but it is a very important information if an integrator wants to analyze a call chain among several modules in order to setup a proper scheduling.	[BSWMD_Entity_006]
IssuedTriggers	This attribute describes that the entity activates a trigger for other BSW modules . This information is used to configure the BSW Scheduler to be able to create a BSW Scheduler API.	[BSWMD_Entity_007]
ManagedMode-Groups	This attribute describes that the entity initiates a mode switch for other BSW modules . This information is used to configure the BSW Scheduler to be able to create a BSW Scheduler API.	[BSWMD_Entity_008]

40 Rule BSWMD_Entity_001: Kind of BswModuleEntity

45 Instruction For the different kinds of *BswModuleEntitys* the following different tag items shall be used to define them within the *BswInternalBehavior* inside the **<ENTITY>** item:

50 **<BSW-CALLED-ENTITY> ... </BSW-CALLED-ENTITY>** for *BswCalledEntitys*
<BSW-INTERRUPT-ENTITY> ... </BSW-INTERRUPT-ENTITY> for *BswInterruptEntity*
<BSW-SCHEDULABLE-ENTITY> ... </BSW-SCHEDULABLE-ENTITY> for *BswSchedulableEntity*

55 Rule BSWMD_Entity_002: ShortName of a BswModuleEntity

Instruction

55 Class: NamingConvention

The ShortName of a *BswModuleEntity* shall be conform to the following convention: **<Prefix>_<DescriptiveText>**.

60 The **<Prefix>** is derived from the specific kind of the *BswModuleEntity*. [Table 56](#) specifies the prefixes.

65 Table 56 Definition of Prefixes for the BswModuleEntitys

Prefix <Prefix>	Description
CE	<i>BswCalledEntity</i>
IE	<i>BswInterruptEntity</i>
SE	<i>BswSchedulableEntity</i>

Rule BSWMD_Entity_003: Reference to a BswModuleEntry

Instruction Every kind of *BswModuleEntity* shall refer to at least one *BswModuleEntry*. This shall be done in the following way:

- ▶ A reference shall be set using the tag item **<IMPLEMENTED-ENTRY-REF>**.
- ▶ The destination type of the *ImplementedEntryRef* shall be set to 'BSW-MODULE-ENTRY'.
- ▶ A correctly and complete path to a *BswModuleEntry* shall be set
- ▶ The *BswModuleEntry* shall be specified as *ProvidedEntry* within the enclosing *BswModuleDescription* (conform to rule [\[BSWMD_EntryExport_001\] p. 346](#))

In addition the following compatibility aspects shall be considered:

- ▶ *BswCalledEntity*: The referred *BswModuleEntry* shall be of Call Type "REGULAR" or "CALLBACK" and the Execution Context shall be set to "UNSPECIFIED", "HOOK" or "TASK".
- ▶ *BswInterruptEntity*: The referred *BswModuleEntry* shall be of Call Type "INTERRUPT" and the Execution Context shall be set to "INTERRUPT-CAT-2".
- ▶ *BswSchedulableEntity*: The referred *BswModuleEntry* shall be of Call Type "SCHEDULED" and the Execution Context shall be set to "TASK".

Details how to define a *BswModuleEntry* are described in chapter [8.3.4.1 "BSW Module API Description \(BswModuleEntry\)" p. 326](#).

The different kinds of CallTypes and ExecutionContexts are explained in rule [\[BSWMD_Entry_006\] p. 330](#) and rule [\[BSWMD_Entry_007\] p. 330](#). An explanation of the *ProvidedEntries* is given in chapter ["Export of a BswModuleEntry" p. 345](#).

Rule BSWMD_Entity_004: Reference to Mode Switch Which is Received from a BswModuleEntity

Instruction Scope: *BswModuleEntity* receiving mode switches (defined as *BswModeDeclarations*)

If a *BswModuleEntity* receives one or more mode switches, references to *BswModeDeclarations* shall be set in the following way:

- ▶ A tag structure shall be created starting with tag item **<ACCESSED-MODE-GROUPS>**
- ▶ As next tag layer the tag item **<MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several references to different *BswModeDeclarations* have to be set.
- ▶ The reference to a *BswModeDeclaration* shall be set within the tag item **<MODE-DECLARATION-GROUP-PROTOTYPE-REF>**
- ▶ The destination type of the *ModeDeclarationGroupPrototypeRef* shall be set to 'MODE-DECLARATION-GROUP'
- ▶ A correctly and complete path to a *BswModeDeclaration* shall be set (which is declared as *RequiredModeGroup* within the enclosing *BswModuleDescription*)

The following example shows references to received mode switches. Definitions for the referred *BswModeDeclarations* shall be available and have to be defined as described in chapter ["BswModes" p. 364](#).

```

<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  ...
  <ENTITYS>
    <BSW-CALLED-ENTITY>
      <SHORT-NAME>CE_Func1</SHORT-NAME>
      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        /AUTOSAR_AdC/BswModuleEntries/Adc_Func1
      </IMPLEMENTED-ENTRY-REF>
    <ACCESSED-MODE-GROUPS>
  
```

```

05    <!-- First reference to a mode switch -->
10   <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
15     <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST="MODE-DECLARATION-GROUP">
20       /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDG_Group1
25     </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
30   </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
35     <!-- Second reference to a mode switch -->
40   <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
45     <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST="MODE-DECLARATION-GROUP">
50       /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDG_Group1
55     </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
60   </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
65     ...
70   </ACCESSED-MODE-GROUPS>
75   </BSW-CALLED-ENTITY>
80 </ENTITYS>
85   ...
90 </BSW-INTERNAL-BEHAVIOR>

```

Hint The ImplementedEntryRef and the ModeDeclarationGroupPrototypeRef have to be written in single lines in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementedEntryRef and ModeDeclarationGroupPrototypeRef are written in three lines only for clearness.

30 Rule BSWMD_Entity_005: Reference to an Internal Trigger Which is Activated from a BswModuleEntity

Instruction Scope: BswModuleEntity activating one or more internal triggers (defined as ReleasedTrigger)

If a BswModuleEntity activates one or more internal triggers references shall be set in the following way:

- ▶ A tag structure shall be created starting with tag item **<ACTIVATION-POINTS>**
- ▶ As next tag layer the tag item **<BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several references to different BswModeDeclarations have to be set.
- ▶ The reference to a BswTrigger shall be set within the tag item **<BSW-INTERNAL-TRIGGERING-POINT-REF>**
- ▶ The destination type of the BswInternalTriggeringPointRef shall be set to 'BSW-INTERNAL-TRIGGERING-POINT'
- ▶ A correctly and complete path to a BswInternalTriggeringPoint shall be set (which is declared as BswInternalTriggeringPoint within the enclosing BswInternalBehavior)

The following example shows references to BswTriggers. Definitions for the referred BswInternalTriggeringPoints shall be available and have to be defined as described in chapter "BswTriggers" p. 375 .

```

50 <BSW-INTERNAL-BEHAVIOR>
55   <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
60   ...
65   <ENTITYS>
70     <BSW-CALLED-ENTITY>
75       <SHORT-NAME>CE_Func1</SHORT-NAME>
80       <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
85         /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
90       </IMPLEMENTED-ENTRY-REF>
95     <ACTIVATION-POINTS>
100       <!-- First reference to an internal trigger -->
105       <BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
110         <BSW-INTERNAL-TRIGGERING-POINT-REF DEST="BSW-INTERNAL-TRIGGERING-POINT">
115           /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/ITP_Trigger1
120         </BSW-INTERNAL-TRIGGERING-POINT-REF>
125       </BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
130       <!-- Second reference to an internal trigger -->
135       <BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
140         <BSW-INTERNAL-TRIGGERING-POINT-REF DEST="BSW-INTERNAL-TRIGGERING-POINT">

```

```

05      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/ITP_Trigger2
</BSW-INTERNAL-TRIGGERING-POINT-REF>
</BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
...
</ACTIVATION-POINTS>
</BSW-CALLED-ENTITY>
</ENTITYS>
...
</BSW-INTERNAL-BEHAVIOR>
```

15 **Hint** The ImplementedEntryRef and the BswInternalTriggeringPointRef have to be written in single lines in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementedEntryRef and BswInternalTriggeringPointRef are written in three lines only for clearness.

20 Rule BSWMD_Entity_006: Reference to an Entry of Another Module (or the Same Module) Which is Called from a BswModuleEntity

25 **Instruction Scope:** BswModuleEntity calling entries of another BSW module (or the same module)

If a BswModuleEntity calls BswModuleEntries of another (or the same) BSW modules references shall be set in the following way:

- 30 ▶ A tag structure shall be created starting with tag item **<CALLED-ENTRYS>**
- ▶ As next tag layer the tag item **<BSW-MODULE-ENTRY-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several BswModuleEntries have to be referred.
- ▶ The reference to a BswModuleEntry shall be set within the tag item **<BSW-MODULE-ENTRY-REF>**
- ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY'
- 35 ▶ A correctly and complete path to a BswModuleEntry shall be set (which is provided as OutgoingCallback, Provided-Entry or as RequiredEntry within the enclosing BswModuleDescription)

40 The following example shows CalledEntries. Definitions for the referred BswModuleEntries shall be available and has to be defined as described in chapter "*BSW Module API Description (BswModuleEntry)*" p. 326

```

<BSW-INTERNAL-BEHAVIOR>
<SHORT-NAME>BswInternalBehavior</SHORT-NAME>
...
<ENTITYS>
  <BSW-CALLED-ENTITY>
    <SHORT-NAME>CE_Func1</SHORT-NAME>
    <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
      /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
    </IMPLEMENTED-ENTRY-REF>
    <CALLED-ENTRYS>
      <!-- First reference to a called entry -->
      <BSW-MODULE-ENTRY-REF-CONDITIONAL>
        <BSW-MODULE-ENTRY-REF DEST="BSW-MODULE-ENTRY">
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
        </BSW-MODULE-ENTRY-REF>
      </BSW-MODULE-ENTRY-REF-CONDITIONAL>
      <!-- Second reference to a called entry -->
      <BSW-MODULE-ENTRY-REF-CONDITIONAL>
        <BSW-MODULE-ENTRY-REF DEST="BSW-MODULE-ENTRY">
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func2
        </BSW-MODULE-ENTRY-REF>
      </BSW-MODULE-ENTRY-REF-CONDITIONAL>
      ...
    </CALLED-ENTRYS>
  </BSW-CALLED-ENTITY>
</ENTITYS>
```

```
...
</BSW-INTERNAL-BEHAVIOR>
```

Hint The ImplementedEntryRef and the BswModuleEntryRef have to be written in single lines in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementedEntryRef and BswModuleEntryRef are written in three lines only for clearness.

Rule BSWMD_Entity_007: Reference to a Trigger Which is Activated for Other BSW Modules

Instruction Scope: BswModuleEntity activating triggers for other BSW modules

If a BswModuleEntity activates triggers for other BSW modules references shall be set in the following way:

- ▶ A tag structure shall be created starting with tag item **<ISSUED-TRIGGERS>**
- ▶ As next tag layer the tag item **<trigger-ref-conditional>** shall be set. This tag layer can be used multiple times if several references to different BswModeDeclarations have to be set.
- ▶ The reference to a BswModeDeclaration shall be set within the tag item **<trigger-ref>**
- ▶ The destination type of the TriggerRef shall be set to 'TRIGGER'
- ▶ A correctly and complete path to a trigger shall be set (which is declared as ReleasedTrigger within the enclosing BswModuleDescription)

The following example shows references to BswTriggers. Definitions for the referred triggers shall be available and have to be defined as described in chapter "BswTriggers" p. 375 .

```
<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  ...
  <ENTITYS>
    <BSW-CALLED-ENTITY>
      <SHORT-NAME>CE_Func1</SHORT-NAME>
      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
      </IMPLEMENTED-ENTRY-REF>
      <ISSUED-TRIGGERS>
        <!-- First reference to an activated trigger -->
        <trigger-ref-conditional>
          <trigger-ref DEST="TRIGGER">
            /AUTOSAR_Adc/BswModuleDescriptions/Adc/ETP_Trigger1
          </trigger-ref>
        </trigger-ref-conditional>
        <!-- Second reference to an activated trigger -->
        <trigger-ref-conditional>
          <trigger-ref DEST="TRIGGER">
            /AUTOSAR_Adc/BswModuleDescriptions/Adc/ETP_Trigger2
          </trigger-ref>
        </trigger-ref-conditional>
        ...
      </ISSUED-TRIGGERS>
    </BSW-CALLED-ENTITY>
  </ENTITYS>
  ...
</BSW-INTERNAL-BEHAVIOR>
```

Hint The ImplementedEntryRef and the TriggerRef have to be written in single lines in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementedEntryRef and TriggerRef are written in three lines only for clearness.

Rule BSWMD_Entity_008: Reference to Initiate a Mode Switch for Other BSW Modules

Instruction Scope: BswModuleEntity initiating mode switches for other BSW modules

If a BswModuleEntity initiates mode switches for other BSW modules references shall be set in the following way:

- ▶ A tag structure shall be created starting with tag item **<MANAGED-MODE-GROUPS>**
- ▶ As next tag layer the tag item **<MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>** shall be set. This tag layer can be used multiple times if several references to different BswModeDeclarations have to be set.
- ▶ The reference to a BswModeDeclaration shall be set within the tag item **<MODE-DECLARATION-GROUP-PROTOTYPE-REF>**
- ▶ The destination type of the ModeDeclarationGroupPrototypeRef shall be set to 'MODE-DECLARATION-GROUP'
- ▶ A correctly and complete path to a BswModeDeclaration shall be set (which is declared as RequiredModeGroup within the enclosing BswModuleDescription)

The following example shows references to received mode switches. How to define a BswModeDeclaration is explained in chapter "*BswModes*" p. 364 .

```

<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  ...
  <ENTITYS>
    <BSW-CALLED-ENTITY>
      <SHORT-NAME>CE_Func1</SHORT-NAME>
      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        /AUTOSAR_Adc/BswModuleEntries/Adc_Func1
      </IMPLEMENTED-ENTRY-REF>
    <MANAGED-MODE-GROUPS>
      <!-- First reference to a mode switch -->
      <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
        <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST="MODE-DECLARATION-GROUP">
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDG_Group1
        </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
      </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
      <!-- Second reference to a mode switch -->
      <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
        <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST="MODE-DECLARATION-GROUP">
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDG_Group2
        </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
      </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
      ...
    </MANAGED-MODE-GROUPS>
  </BSW-CALLED-ENTITY>
</ENTITYS>
...
</BSW-INTERNAL-BEHAVIOR>

```

Hint The ImplementedEntryRef and the ModeDeclarationGroupPrototypeRef have to be written in single lines in a productive BSWMD file. Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the ImplementedEntryRef and ModeDeclarationGroupPrototypeRef are written in three lines only for clearness.

Rule BSWMD_Entity_009: Specifying the Category of the BswInterruptEntity

Instruction Scope: Only relevant for BswInterruptEntity

Every BswInterruptEntity shall specify the category of the interrupt service using the tag item **<INTERRUPT-CATEGORY>**.

The applied category shall be the same as used in the referred BswModuleEntry.

05 The following example shows the simplest form of a definition of a BswInterruptEntity:

```

<BSW-INTERNAL-BEHAVIOR>
  <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
  ...
  <ENTITYS>
    <BSW-INTERRUPT-ENTITY>
      <SHORT-NAME>IE_Interrupt1</SHORT-NAME>
      <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
        /AUTOSAR_Adc/BswModuleEntrys/Adc_Isr1
      </IMPLEMENTED-ENTRY-REF>
      <INTERRUPT-CATEGORY>CAT1</INTERRUPT-CATEGORY>
    </BSW-INTERRUPT-ENTITY>
  </ENTITYS>
  ...
</BSW-INTERNAL-BEHAVIOR>
```

20

Hint The ImplementedEntryRef 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 ImplementedEntryRef is written in three lines only for clearness.

25

The corresponding BswModuleEntry shall have a proper call type and execution context. For more details refer to "["BSW Module API Description \(BswModuleEntry\)" p. 326](#) and in detail to the rules [\[BSWMD_Entry_006\] p. 330](#) and [\[BSWMD_Entry_007\] p. 330](#). The following example gives a rough overview:

30

```

<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>BswModuleEntrys</SHORT-NAME>
    <ELEMENTS>
      <BSW-MODULE-ENTRY>
        <SHORT-NAME>Adc_Isr1</SHORT-NAME>
        ...
        <CALL-TYPE>INTERRUPT</CALL-TYPE>
        <EXECUTION-CONTEXT>INTERRUPT-CAT-1</EXECUTION-CONTEXT>
        ...
      </BSW-MODULE-ENTRY>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
```

45

50

55

60

65

70

8.3.4.3 BswEvents

The abstract *BswEvent* class is used as base class for all kinds of events which can start a *BswScheduledEntity*. There is a one to n connection between a *BswScheduledEntity* and a *BswEvent*. A *BswScheduledEntity* can be only activated by the BSW Scheduler according to the definition of a *BswEvent*. There are different types of *BswEvents* which provide different use cases for the activation of a schedulable entity. There are time based events, events which are associated with modes and others which are connected with triggers. The corresponding modes and triggers have to be specified to use that specific types of *BswEvents*. [Table 57](#) gives a rough overview of all kinds of *BswEvents*.

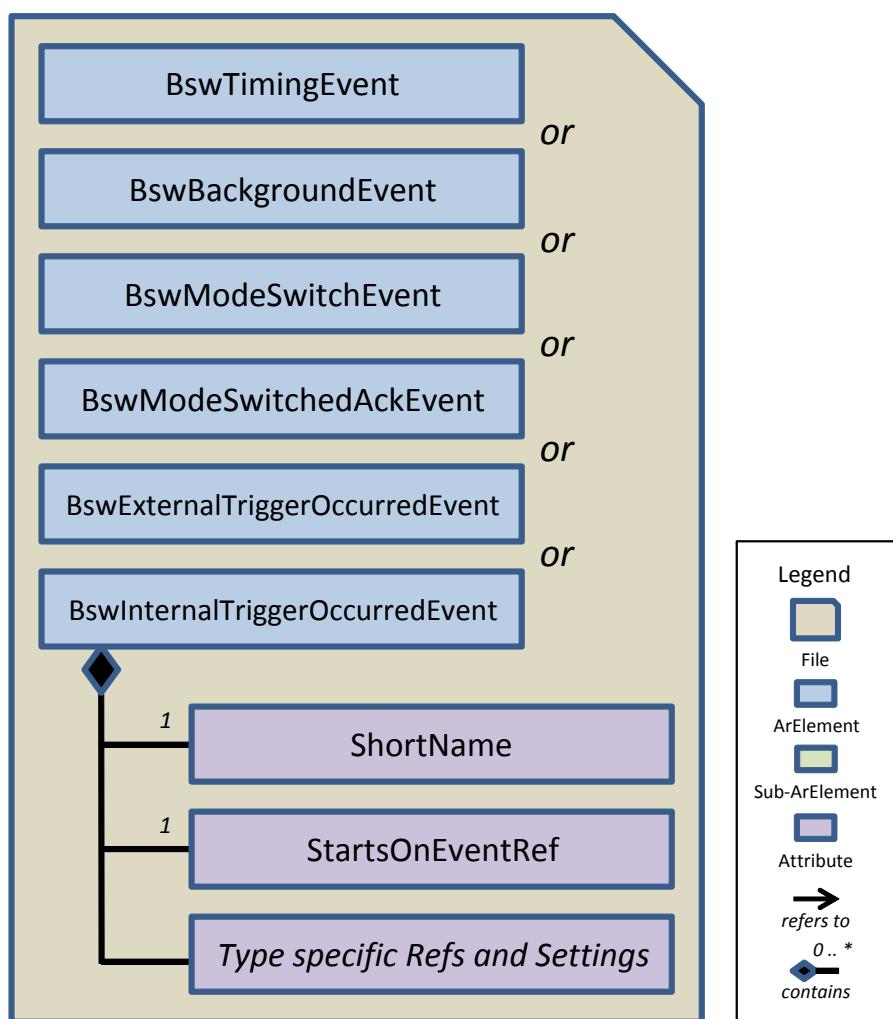
[Table 57](#) Overview of Different Kinds of *BswEvents*

BswEvent	Description
BswTimingEvent	A <i>BswTimingEvent</i> is the standard event for a periodic schedulable <i>BswModuleEntity</i> . A <i>BswTimingEvent</i> represents a recurring <i>BswEvent</i> driven by a defined time period. A <i>BswTimingEvent</i> is directly driven by the BSW Scheduler resp. OS without external source.
BswBackgroundEvent	A recurring <i>BswEvent</i> which is used to perform background activities. It is similar to a <i>BswTimingEvent</i> but has no fixed time period and is activated only with low priority. A <i>BswBackgroundEvent</i> is directly driven by the BSW Scheduler resp. OS without external source.
BswModeSwitchEvent	The <i>BswModeSwitchEvent</i> is a special kind of event and can be used to start a <i>BswModuleEntity</i> at the entry or exit of a specific mode or at the transition of two different modes. As the name of that event and the description suggests the <i>BswModeSwitchEvent</i> is associated with specific mode(s).
BswModeSwitchedAckEvent	At the sender side of a mode switch (i.e. in the module providing the mode group), a <i>BswModeSwitchedAckEvent</i> can be used to start a <i>BswModuleEntity</i> after a mode switch has been acknowledged by the BSW Scheduler. The event is raised after a switch of the referenced mode group has been acknowledged or an error occurs. The referenced mode group must be provided by the same BSW module.
BswExternalTriggerOccurred-Event	The <i>BswExternalTriggerOccurredEvent</i> specifies the fact, that the event is raised in response to a trigger issued by another BSW module. For instance this can be used to communicate ECU-external events, like wakeup-events or crank-shaft-events directly between BSW modules.
BswInternalTriggerOccurredEvent	A <i>BswEvent</i> which can happen sporadically. The event is activated by explicit calls from the module to the BSW Scheduler. The main purpose for such an event is to cause a context switch, e.g. from an ISR context into a task context. Activation and switching are handled within the same module or cluster only.

The definition of *BswEvents* is a part of the *BswInternalBehavior* (refer to [Chapter 8.2.1.2 "BswInternalBehavior" p. 299](#)) and thus it belongs to the internal view of a BSW module. Since the *BswInternalBehavior* is not splittable to different files (see [\[BSWMD_Split_003\] p. 308](#)) all specified *BswEvents* have to be located in a single BSWMD file. This restriction has to be adhered on (refer to chapter ["Scheduling of BSW via RTE" p. 319](#)).

The following [Figure 72](#) gives an overview of the relevant elements and attributes which are required to specify a *BswEvent*.

Figure 72 Details of BSW Events



The general structure of the different BSW Events is equal, i.e. they have a ShortName and a StartOnEventRef. As shown in the figure above there are type-specific references and settings. [Table 58](#) provides an overview of the specific tag items and the related rules.

Table 58 Overview of Type Specific References and Settings

BswEvent	Type Specific Reference or Setting	Related Rule
BswTimingEvent	Period: Requirement for the time period (in seconds) by which this event is triggered.	[BSWMD_Event_004]
BswBackgroundEvent	No special reference or setting available	-
BswModeSwitchEvent	Activation: Kind of activation w.r.t. to the referred mode.	[BSWMD_Event_005]
	ModelRefs: Reference to one or two Modes that initiate the BswModeSwitchEvent.	[BSWMD_Event_006]
BswModeSwitchedAckEvent	ModeGroupRef: A mode group provided by this module. The acknowledge of a mode switch of this group raises this event.	[BSWMD_Event_007]
BswExternalTriggerOccurred-Event	TriggerRef: The trigger associated with this event. The trigger is external to this module.	[BSWMD_Event_008]
BswInternalTriggerOccurred-Event	EventSourceRef: Reference to a BswInternalTriggeringPoint which activates this event.	[BSWMD_Event_009]

The following ARXML snippet shows an example of a BswTimingEvent. Additional examples are given in the dedicated rules.

```

05 <INTERNAL-BEHAVIORS>
10  <BSW-INTERNAL-BEHAVIOR>
15    <SHORT-NAME>BswInternalBehavior</SHORT-NAME>
     ...
     <ENTITYS>...</ENTITYS>

20    <EVENTS>
25      <BSW-TIMING-EVENT>                                <!-- Rule [BSWMD_Event_001] -->
        <SHORT-NAME>TE_Adc_Event1</SHORT-NAME>           <!-- Rule [BSWMD_Event_002] -->
        <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY"> <!-- Rule [BSWMD_Event_003] -->
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
        </STARTS-ON-EVENT-REF>
        <PERIOD>0.1</PERIOD>                            <!-- Rule [BSWMD_Event_004] -->
      </BSW-TIMING-EVENT>
     ...
   </EVENTS>

30   <PER-INSTANCE-PARAMETERS>...</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. 292). Otherwise the tool chain cannot interpret the BSWMD file correctly.

Hint The StartsOnEventRef 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 StartsOnEventRef is written in three lines only for clearness.

35 Rule BSWMD_Event_001: Kind of BswEvents

Instruction For the different kinds of *BswEvents* the following different tag items shall be used to define them within the *BswInternalBehavior* inside the **<EVENTS>** item:

```

40  <BSW-TIMING-EVENT> ... </BSW-TIMING-EVENT> for BswTimingEvent
  <BSW-BACKGROUND-EVENT> ... </BSW-BACKGROUND-EVENT> for BswBackgroundEvent
  <BSW-MODE-SWITCH-EVENT> ... </BSW-MODE-SWITCH-EVENT> for BswModeSwitchEvent
  <BSW-MODE-SWITCHED-ACK-EVENT> ... </BSW-MODE-SWITCHED-ACK-EVENT> for BswModeSwitchedAckEvent
  <BSW-EXTERNAL-TRIGGER-OCCURED-EVENT> ... </BSW-EXTERNAL-TRIGGER-OCCURED-EVENT> for BswExternalTriggerOccuredEvent
  <BSW-INTERNAL-TRIGGER-OCCURED-EVENT> ... </BSW-INTERNAL-TRIGGER-OCCURED-EVENT> for BswInternalTriggerOccuredEvent

```

55 Rule BSWMD_Event_002: ShortName of a BswEvent

Instruction

Class: NamingConvention

The ShortName of a *BswEvent* shall be conform to the following convention: <Prefix>_<DescriptiveText>.

The <Prefix> is derived from the specific kind of the *BswEvent*. [Table 59](#) specifies the prefixes.

Table 59 Definition of Prefixes for the BswModuleEntitys

Prefix <Prefix>	Description
TE	BswTimingEvent

Prefix <Prefix>	Description
BE	BswBackgroundEvent
MSE	BswModeSwitchEvent
MSAE	BswModeSwitchedAckEvent
ETOE	BswExternalTriggerOccuredEvent
ITOE	BswInternalTriggerOccuredEvent

15 Rule BSWMD_Event_003: Reference to a BswModuleEntity

20 **Instruction** Every kind of a *BswEvent* shall refer to a single *BswModuleEntity* (which is started by the event) which is done in the following way:

- 25 ▶ The reference shall be set using the tag item **<STARTS-ON-EVENT-REF>**.
- ▶ The destination type of the StartsOnEventRef shall be set to the specific type of the *BswModuleEntity* ('BSW-CALLED-ENTITY', 'BSW-SCHEDULABLE-ENTITY' or 'BSW-INTERRUPT-ENTITY').
- ▶ A correctly and complete path to a *BswModuleEntity* shall be set.

The following example shows the implementation of the rule:

```
30 <EVENTS>
    <BSW-BACKGROUND-EVENT>
        <SHORT-NAME>BE_Func1</SHORT-NAME>
        <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
            /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
        </STARTS-ON-EVENT-REF>
    </BSW-BACKGROUND-EVENT>
</EVENTS>
```

35 **Hint** The StartsOnEventRef 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 StartsOnEventRef is written in three lines only for clearness.

40 Rule BSWMD_Event_004: Timing Period for BswTimingEvents

45 **Instruction**

Scope: Only relevant for *BswTimingEvents*

50 *BswTimingEvents* shall specify a time period (in seconds) using the tag item **<PERIOD>** and shall have a value greater than 0.

55 Via a timing period the *BswTimingEvent* specifies how often the referenced *BswSchedulableEntity* is scheduled. More details about the *BswSchedulableEntity* is given in chapter "*Properties of a Code Fragment (BswModuleEntity)*" p. 347.

The following ARXML snippet shows a *BswTimingEvent* using the timing period:

```
55 <EVENTS>
    <BSW-TIMING-EVENT>
        <SHORT-NAME>TE_Func1</SHORT-NAME>
        <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULED-ENTITY">
            /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
        </STARTS-ON-EVENT-REF>
        <PERIOD>0.1</PERIOD>           
    </BSW-TIMING-EVENT>
</EVENTS>
```

Hint The StartsOnEventRef 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 StartsOnEventRef is written in three lines only for clearness.

Rule BSWMD_Event_005: Activation of a BswModeSwitchEvent

Instruction

Scope: Only relevant for BswModeSwitchEvents

The kind of activation of a BswModeSwitchEvent shall be specified using the tag item **<ACTIVATION>**.

The following settings are possible: **ON-ENTRY, ON-EXIT, ON-TRANSITION**

If the activation is set to the value 'ON-TRANSITION' the BswModeSwitchEvent shall refer to two distinct modes belonging to the same instance of a ModeDeclarationGroup, their order defining the direction of the transition. In all other cases the BswModeSwitchEvent shall refer to exactly one mode.

The activation tag item specifies the kind of mode switch condition used for the activation of the BswModeSwitchEvent. This is used to start a respective BswModuleEntity at the entry or exit or on transition of a mode.

Table 60 List of Activation Literals of a BswModeSwitchEvent

Literal	Description
ON-ENTRY	On entering the referred mode
ON-EXIT	On exiting the referred mode
ON-TRANSITION	On transition of the 1st referred mode to the 2nd referred mode (default)

The following ARXML snippet shows a BswModeSwitchEvent with a single mode reference (because of activation is set to 'ON-EXIT'):

```

<EVENTS>
  <BSW-MODE-SWITCH-EVENT>
    <SHORT-NAME>MSE_Func1</SHORT-NAME>
    <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
    </STARTS-ON-EVENT-REF>
    <ACTIVATION>ON-EXIT</ACTIVATION>
    <MODE-IREFS>
      <MODE-IREF>
        <CONTEXT-MODE-DECLARATION-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
          /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDGP_Type1
        </CONTEXT-MODE-DECLARATION-GROUP-REF>
        <TARGET-MODE-REF DEST="MODE-DECLARATION">
          /AUTOSAR_Adc/ModeDeclarationGroups/ModeGroup1/Mode1
        </TARGET-MODE-REF>
      </MODE-IREF>
    </MODE-IREFS>
  </BSW-MODE-SWITCH-EVENT>
</EVENTS>

```

Hint The StartsOnEventRef, ContextModeDeclarationGroupRef and TargetModeRef have 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 StartsOnEventRef, ContextModeDeclarationGroupRef and TargetModeRef are written in three lines only for clearness.

Rule BSWMD_Event_006: References to Modes for BswModeSwitchEvents

Instruction

Scope: Only relevant for BswModeSwitchEvents

05 Based on the chosen kind of activation of a BswModeSwitchEvent one or two references to BSW modes shall be set. One reference is needed if the activation is set to 'ON-ENTRY' or 'ON-EXIT', two references are needed for 'ON-TRANSITION'.

10 The references shall be set in the following way:

- 15
- ▶ A tag structure shall be created starting with tag item **<MODE-IREFS>**
 - ▶ Within that one or two sub-structures shall be created using the tag item **<MODE-IREF>**
 - ▶ Within the Modelrefs, references to a BswModeDeclarationGroupPrototype and a BswModeDeclaration shall be set. Therefore the tag items **<CONTEXT-MODE-DECLARATION-GROUP-REF>** and **<TARGET-MODE-REF>** shall be used.
 - ▶ The destination type of the ContextModeDeclarationGroupRef shall be set to 'MODE-DECLARATION-GROUP--PROTOTYPE' and the destination type of the TargetModeRef shall be set to 'MODE-DECLARATION'
 - ▶ Complete paths to the BswModeDeclarationGroupPrototype and the BswModeDeclaration shall be set

20 Additional condition: The ModeDeclaration used by a BswModeSwitchEvent shall be specified as AccessedModeGroup as part of the BswScheduledEntity of the enclosing BswInternalBehavior of the BSW module.

25 The BswModeSwitchEvent brings the three items BswModuleEntity, BswMode and BswEvent together. It represents that a BswModuleEntity can be called based on BswModes. The activation level can be chosen.

30 The following ARXML snippet shows a BswModeSwitchEvent with two reference blocks (because of activation is set to 'ON-TRANSITION'):

```

35 <EVENTS>
<BSW-MODE-SWITCH-EVENT>
    <SHORT-NAME>MSE_Func1</SHORT-NAME>
    <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
        /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
    </STARTS-ON-EVENT-REF>
    <ACTIVATION>ON-TRANSITION</ACTIVATION>
    <MODE-IREFS>
        <MODE-IREF>                                <!-- First block with references -->
            <CONTEXT-MODE-DECLARATION-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDGP_Type1
            </CONTEXT-MODE-DECLARATION-GROUP-REF>
            <TARGET-MODE-REF DEST="MODE-DECLARATION">
                /AUTOSAR_Adc/ModeDeclarationGroups/ModeGroup1/Mode1
            </TARGET-MODE-REF>
        </MODE-IREF>
        <MODE-IREF>                                <!-- Second block with references -->
            <CONTEXT-MODE-DECLARATION-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
                /AUTOSAR_Adc/BswModuleDescriptions/Adc/MDGP_Type2
            </CONTEXT-MODE-DECLARATION-GROUP-REF>
            <TARGET-MODE-REF DEST="MODE-DECLARATION">
                /AUTOSAR_Adc/ModeDeclarationGroups/ModeGroup1/Mode2
            </TARGET-MODE-REF>
        </MODE-IREF>
    </MODE-IREFS>
</BSW-MODE-SWITCH-EVENT>
</EVENTS>

```

55 **Hint** The StartsOnEventRef, ContextModeDeclarationGroupRef and TargetModeRef have 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 StartsOnEventRef, ContextModeDeclarationGroupRef and TargetModeRef are written in three lines only for clearness.

Rule BSWMD_Event_007: Reference to a Mode Group for BswModeSwitchedAckEvents

Instruction

Scope: Only relevant for BswModeSwitchedAckEvents

Within a BswModeSwitchedAckEvent a reference to a ModeDeclarationGroupPrototype shall be set in the following way:

- ▶ The reference shall be set using the tag item **<MODE-GROUP-REF>**
- ▶ The destination type of the ModeGroupRef shall be set to 'MODE-DECLARATION-GROUP-PROTOTYPE'
- ▶ A correctly and fully specified path to the ModeDeclarationGroupPrototype shall be set

Additional condition: The ModeDeclarationGroupPrototype used by a BswModeSwitchedAckEvent shall be specified as ProvidedModeGroup as part of the BswModuleDescription of the same BSW module.

The following ARXML snippet shows a BswModeSwitchedAckEvent:

```
<EVENTS>
  <BSW-MODE-SWITCHED-ACK-EVENT>
    <SHORT-NAME>MSAE_Event1</SHORT-NAME>
    <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
    </STARTS-ON-EVENT-REF>
    <MODE-GROUP-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/MGDP_Type1
    </MODE-GROUP-REF>
  </BSW-MODE-SWITCHED-ACK-EVENT>
</EVENTS>
```

Hint The StartsOnEventRef and ModeGroupRef have 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 StartsOnEventRef and ModeGroupRef are written in three lines only for clearness.

Rule BSWMD_Event_008: Reference to an External Trigger for a BswExternalTriggerOccuredEvent

Instruction

Scope: Only relevant for BswExternalTriggerOccuredEvents

Within a BswExternalTriggerOccuredEvent a reference to a Trigger shall be set in the following way:

- ▶ The reference shall be set using the tag item **<TRIGGER-REF>**
- ▶ The destination type of the TriggerRef shall be set to 'TRIGGER'
- ▶ A correctly and complete path to the Trigger shall be set

Additional condition: A BswExternalTriggerOccuredEvent shall refer to a Trigger that is declared as RequiredTrigger within the BswModuleDescription of the same module.

The trigger is internal to this module. The following ARXML snippet shows a BswModeSwitchedAckEvents:

```
<EVENTS>
  <BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT>
    <SHORT-NAME>ETOE_Event1</SHORT-NAME>
    <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
    </STARTS-ON-EVENT-REF>
    <TRIGGER-REF DEST="TRIGGER">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/ETP_BswTrig1
    </TRIGGER-REF>
```

05 </BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT>
 </EVENTS>

10 **Hint** The StartsOnEventRef and TriggerRef have 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 StartsOnEventRef and TriggerRef are
 written in three lines only for clearness.

15 **Rule BSWMD_Event_009:** Reference to an Internal Trigger for a BswInternalTriggerOccuredEvent

Instruction

20 **Scope:** Only relevant for BswInternalTriggerOccuredEvents

25 Within a BswInternalTriggerOccuredEvent a reference to a BswInternalTriggeringPoint shall be set in the following
 way:

- ▶ The reference shall be set using the tag item **<EVENT-SOURCE-REF>**
- ▶ The destination type of the EventSourceRef shall be set to 'BSW-INTERNAL-TRIGGERING-POINT'
- ▶ A correctly and complete path to the Trigger shall be set

30 The trigger associated with this event. The trigger is external to this module.

35 The following ARXML snippet shows a BswModeSwitchedAckEvents:

```
<EVENTS>
  <BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
    <SHORT-NAME>ITOE_Event1</SHORT-NAME>
    <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/SE_Func1
    </STARTS-ON-EVENT-REF>
    <EVENT-SOURCE-REF DEST="BSW-INTERNAL-TRIGGERING-POINT">
      /AUTOSAR_Adc/BswModuleDescriptions/Adc/BswInternalBehavior/ITP_BswTrig1
    </EVENT-SOURCE-REF>
  </BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
</EVENTS>
```

40 **Hint** The StartsOnEventRef and EventSourceRef have to be written in a single line in a productive BSWMD file.
 45 Otherwise the RTE generator cannot interpret the BSWMD file. In the example above the StartsOnEventRef and
 EventSourceRef are written in three lines only for clearness.

8.3.4.4 BswModes

Remark: This chapter will be reworked with one of the next version of the BSW Coding Guideline.

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 are 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.

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>
  </AR-PACKAGES>
</AUTOSAR>
```

8.3.4.4.1 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 BswScheduledEntity (or BSWCalledEntity or BswInterruptEntity).

Example for defining BSW Mode Management on the Mode Management side (for a BswScheduledEntity activated by a BswTimingEvent 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>BswModuleEntries</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>
              ...
            <PROVIDED-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>
            </PROVIDED-MODE-GROUPS>
            ...
          <INTERNAL-BEHAVIORS>
            <BSW-INTERNAL-BEHAVIOR>
              <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
              <ENTITIES>
                <BSW-SCHEDULABLE-ENTITY>
                  <SHORT-NAME>BswScheduledEntityName</SHORT-NAME>
                  ...
                <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 -->
                    </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
                </MANAGED-MODE-GROUPS>
              </ENTITIES>
            </BSW-INTERNAL-BEHAVIOR>
          </INTERNAL-BEHAVIORS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

```

05           </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
06           </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
07           </MANAGED-MODE-GROUPS>
08           ...
09           <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
10             <!-- Refer to the Snippet of a BswModuleEntry -->
11             </IMPLEMENTED-ENTRY-REF>
12             ...
13             </BSW-SCHEDULABLE-ENTITY>
14           </ENTITYS>
15           <EVENTS>
16             <BSW-TIMING-EVENT>
17               <SHORT-NAME>BswTimingEventName</SHORT-NAME>
18               ...
19               <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
20                 <!-- Refer to the Snippet of a BswSchedulableEntity -->
21                 </STARTS-ON-EVENT-REF>
22                 ...
23                 <PERIOD>0.1</PERIOD>
24               </BSW-TIMING-EVENT>
25             </EVENTS>
26             ...
27             </BSW-INTERNAL-BEHAVIOR>
28           </INTERNAL-BEHAVIORS>
29           </BSW-MODULE-DESCRIPTION>
30         </ELEMENTS>
31         </AR-PACKAGE>
32         </AR-PACKAGES>
33         </AR-PACKAGE>
34       </AR-PACKAGES>
35     </AUTOSAR>

```

35

40

Hint The TypeTRef, ModeDeclarationGroupPrototypeRef, ImplementedEntryRef and StartsOnEventRef have 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 TypeTRef, ModeDeclarationGroupPrototypeRef, ImplementedEntryRef and StartsOnEventRef are written in three lines only for clearness.

45

Example of generated header files for a BSW module: SchM_ModuleName.h

This file contains the API for the Switching Modes:

SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName(data)

And the API for the BswSchedulableEntity (activated by TimingEvent):

FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void)

50

```

60   /**
61    * @file      SchM_ModuleName.h
62    *
63    * @brief     Basic Software Scheduler Module Interlink header file
64    *
65    * @note      AUTOMATICALLY GENERATED FILE! DO NOT EDIT!
66    *
67    * @note      Generated by ETAS GmbH RTA-RTE v5.4.0 (R4.0 backend version: v7.1.27 (Build 31685))
68    */
69
70 #ifndef SchM_ModuleName
71 #define SchM_ModuleName
72
73 #include "SchM_ModuleName_Type.h"
74 #include "Rte_Intl.h"
75 #ifdef __cplusplus
76 extern "C" {
77 #endif /* __cplusplus */

```

70

```

05 #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
10 #define RTE_PRV_ALL_API
#endif

15 /* API Mapping Macros */
16 #ifndef RTE_CORE

17 #define RTE_START_SEC_CODE
18 #include "MemMap.h" /*lint !e537 permit multiple inclusion */
19 FUNC(VAR(Std_ReturnType, AUTOMATIC), RTE_CODE)
20     SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName(VAR(uint8, AUTOMATIC) data);
21 #define RTE_STOP_SEC_CODE
22 #include "MemMap.h" /*lint !e537 permit multiple inclusion */
23 #if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
24 #define SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName( data )
25     (SchM_Switch_ModuleName_ModeDeclarationGroupPrototypeName(data))
26 #endif
27 #endif /* RTE_CORE */

28 ****
29 *** Schedulable Entity Prototypes
30 ***
31 ****

35 #define MODULENAME_START_SEC_CODE
36 #include "MemMap.h" /*lint !e537 permit multiple inclusion */
37 FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void);
38 #define MODULENAME_STOP_SEC_CODE
39 #include "MemMap.h" /*lint !e537 permit multiple inclusion */

40 #ifdef __cplusplus
41 } /* extern C */
42 #endif /* __cplusplus */

43 #endif /* SchM_ModuleName */

```

8.3.4.4.2 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>

```

```

05      <SHORT-NAME>AUTOSAR_ModuleName</SHORT-NAME>
10      <AR-PACKAGES>
15          <AR-PACKAGE>
20              <SHORT-NAME>BswModuleEntries</SHORT-NAME>
25                  <ELEMENTS>
30                      <BSW-MODULE-ENTRY>
35                          <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
40                          ...
45                  </BSW-MODULE-ENTRY>
50          </ELEMENTS>
55      </AR-PACKAGE>
60      <AR-PACKAGE>
65          <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
70          <ELEMENTS>
75              <BSW-MODULE-DESCRIPTION>
80                  <SHORT-NAME>ModuleName</SHORT-NAME>
85                  ...
90                  <REQUIRED-MODE-GROUPS>
95                      <MODE-DECLARATION-GROUP-PROTOTYPE>
100                          <SHORT-NAME>ModeDeclarationGroupPrototypeName</SHORT-NAME>
105                          ...
110                          <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
115                              <!-- Refer to the Snippet of a ModeDeclarationGroup -->
120                      </TYPE-TREF>
125                      ...
130                  </MODE-DECLARATION-GROUP-PROTOTYPE>
135          </REQUIRED-MODE-GROUPS>
140      <INTERNAL-BEHAVIORS>
145          <BSW-INTERNAL-BEHAVIOR>
150              <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
155              ...
160          <ENTITYS>
165              <BSW-SCHEDULABLE-ENTITY>
170                  <SHORT-NAME>BswSchedulableEntityName</SHORT-NAME>
175                  ...
180                  <IMPLEMENTED-ENTRY-REF DEST="BSW-MODULE-ENTRY">
185                      <!-- Refer to the Snippet of a BswModuleEntry -->
190                  </IMPLEMENTED-ENTRY-REF>
195                  ...
200          </BSW-SCHEDULABLE-ENTITY>
205      </ENTITYS>
210      <EVENTS>
215          ...
220      </EVENTS>
225          ...
230          </BSW-INTERNAL-BEHAVIOR>
235      </INTERNAL-BEHAVIORS>
240      </BSW-MODULE-DESCRIPTION>
245      </ELEMENTS>
250  </AR-PACKAGE>
255  </AR-PACKAGES>
260 </AR-PACKAGE>
265 </AR-PACKAGES>
270 </AUTOSAR>

```

Hint The ModeDeclarationGroupPrototypeRef, TypeTRef and ImplementedEntryRef have 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 ModeDeclarationGroupPrototypeRef, TypeTRef and ImplementedEntryRef are written in three lines only for clearness.

Rule BSWMD_Mode_004: Definition of BswModeSwitchEvent

Instruction For each BSW Mode Switch Event, a **BswModeSwitchEvent** element shall be defined in a **BswInternalBehavior**.

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 is 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>
              ...
            <INTERNAL-BEHAVIORS>
              <BSW-INTERNAL-BEHAVIOR>
                <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                ...
              <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>
              </BSW-MODE-SWITCH-EVENT>
            </EVENTS>
          </INTERNAL-BEHAVIORS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

05 ...

10 <ACTIVATION>ON-ENTRY (or ON-EXIT or ON-TRANSITION) </ACTIVATION>

15 <MODE-IREF>

20 <MODE-IREF>

25 <CONTEXT-MODE-DECLARATION-GROUP-REF

30 DEST="MODE-DECLARATION-GROUP-PROTOTYPE">

35 <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->

40 </CONTEXT-MODE-DECLARATION-GROUP-REF>

45 <TARGET-MODE-REF DEST="MODE-DECLARATION">

50 <!-- Refer to the Snippet of a Mode of a ModeDeclarationGroup -->

55 </TARGET-MODE-REF>

60 </MODE-IREF>

65 </MODE-IREFS>

In case of **ACTIVATION** set to **ON-TRANSITION** a second **MODE-IREF** is needed (then BswScheduledEntity activation will take place when current mode is equal to the one stated in 1st MODE-IREF and next mode is 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>

```

Hint The ContextModeDeclarationGrouRef and TargetModeRef have 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 ContextModeDeclarationGrouRef and TargetModeRef are written in three lines only for clearness.

Example of generated header files for a BSW module: SchM_ModuleName.h

This file contains the API for the BswScheduledEntity:

FUNC(void, MODULENAME_CODE) ModuleName_BswModuleEntryName(void)

```

55  /** @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))
   */
60
65
#ifndef SchM_ModuleName
#define SchM_ModuleName

#include "SchM_ModuleName_Type.h"

```

```

05 #include "Rte_Intl.h"
# ifdef __cplusplus
extern "C" {
# endif /* __cplusplus */

10 #if !defined(RTE_RUNNABLEAPI_BswScheduledEntityName)
#define RTE_PRV_ALL_API
#endif

15 /* API Mapping Macros */
#ifndef RTE_CORE

16 #endif /* RTE_CORE */

20 //*****
21 /**
22 *** Schedulable Entity Prototypes
23 ***
24 //*****
```

```

25 #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 */

30 #ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

35 #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*:

```

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>
        <SHORT-NAME>BswModuleEntries</SHORT-NAME>
        <ELEMENTS>
          <BSW-MODULE-ENTRY>
            <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
            ...
          </BSW-MODULE-ENTRY>
        </ELEMENTS>
50      </AR-PACKAGE>
      <AR-PACKAGE>
        <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
        <ELEMENTS>
          <BSW-MODULE-DESCRIPTION>
            <SHORT-NAME>ModuleName</SHORT-NAME>
            ...
          </BSW-MODULE-DESCRIPTION>
        </ELEMENTS>
55      </AR-PACKAGE>
      <AR-PACKAGE>
        <SHORT-NAME>BswInternalBehaviors</SHORT-NAME>
        <ELEMENTS>
          <INTERNAL-BEHAVIORS>
            <BSW-INTERNAL-BEHAVIOR>
              <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
              ...
            </BSW-INTERNAL-BEHAVIOR>
          </INTERNAL-BEHAVIORS>
        </ELEMENTS>
60      </AR-PACKAGE>
    </AR-PACKAGES>
  </AUTOSAR>
```

```

05      <ENTITYS>
10        <BSW-SCHEDULABLE-ENTITY>
15          <SHORT-NAME>BswScheduledEntityName</SHORT-NAME>
20            ...
25          </BSW-SCHEDULABLE-ENTITY>
30        </ENTITYS>
35        <EVENTS>
40          <BSW-.....-EVENT>
45            <SHORT-NAME>BswEventName</SHORT-NAME>
50              <DISABLED-IN-MODE-IREF>
55                <CONTEXT-MODE-DECLARATION-GROUP-REF
60                  DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
65                  <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
70                </CONTEXT-MODE-DECLARATION-GROUP-REF>
75                <TARGET-MODE-REF DEST="MODE-DECLARATION">
80                  <!-- Refer to the Snippet of a Mode of a ModeDeclarationGroup -->
85                </TARGET-MODE-REF>
90              </DISABLED-IN-MODE-IREF>
95            </BSW-.....-EVENT>
100          </EVENTS>
105        </BSW-INTERNAL-BEHAVIOR>
110      </INTERNAL-BEHAVIORS>
115    </BSW-MODULE-DESCRIPTION>
120  </ELEMENTS>
125  </AR-PACKAGE>
130  </AR-PACKAGES>
135  </AR-PACKAGE>
140  </AR-PACKAGES>
145  </AUTOSAR>

```

35 **Hint** The ContextModeDeclarationGrouRef and TargetModeRef have 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 ContextModeDeclarationGrouRef and TargetModeRef are written in three lines only for clearness.

40

8.3.4.4.3 BSW Mode Access

Rule BSWMD_Mode_005: Definition of BSW Mode Access

45

Instruction If a BswScheduledEntity needs to access to the current Mode the **AccessedModeGroups** shall be defined in the **BswScheduledEntity** with a reference to the **ModeDeclarationGroupPrototype**.

50

If a BswScheduledEntity 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.

Example for defining the AccessedModeGroups in a BswScheduledEntity:

55

```

<?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>BswModuleEntries</SHORT-NAME>
      <ELEMENTS>
        <BSW-MODULE-ENTRY>
          <SHORT-NAME>BswModuleEntryName</SHORT-NAME>

```

60

65

70

```

05      ...
10      </BSW-MODULE-ENTRY>
15      </ELEMENTS>
</AR-PACKAGE>
<AR-PACKAGE>
10      <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
15      <ELEMENTS>
20          <BSW-MODULE-DESCRIPTION>
25              <SHORT-NAME>ModuleName</SHORT-NAME>
                  ...
20              <INTERNAL-BEHAVIORS>
25                  <BSW-INTERNAL-BEHAVIOR>
30                      <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                  ...
35                  <ACCESSED-MODE-GROUPS>
35                      <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
35                          <MODE-DECLARATION-GROUP-PROTOTYPE-REF
35                              DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
35                              <!-- Refer to the Snippet of a ModeDeclarationGroupPrototype -->
35                          </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
35                      </MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL>
35                  </ACCESSED-MODE-GROUPS>
35                  ...
35          </BSW-SCHEDULABLE-ENTITY>
35      </ENTITIES>
35      ...
40          </BSW-INTERNAL-BEHAVIOR>
40      </INTERNAL-BEHAVIORS>
40      </BSW-MODULE-DESCRIPTION>
40      </ELEMENTS>
40      </AR-PACKAGE>
40      </AR-PACKAGES>
40      </AR-PACKAGE>
40      </AR-PACKAGES>
40  </AUTOSAR>
```

45 **Hint** The ModeDeclarationGroupPrototypeRef 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 ModeDeclarationGroupPrototypeRef is written in three lines only for clearness.

50 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_ModeDeclarationGroupName()

```

55  /** @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))
   */
60
65
# ifndef SchM_ModuleName
# define SchM_ModuleName
# include "SchM_ModuleName_Type.h"
# include "Rte_Intl.h"
```

```

05 #ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

10 #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

15 /* 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

20 #endif /* RTE_CORE */

25 //*****
*** Schedulable Entity Prototypes
*** ****
30 //*****
```

35 #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 */

40 #ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

45 #endif /* SchM_ModuleName */

50

55

60

65

70 CDG-SMT | BSW Coding Guideline | Volker Kairies | CDG-SMT/ESM1 | 1.10 | 2016-01-31 | released for CDG-SMT | 2016-02-03 14:37:32 | 4.7.1 | © 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.

8.3.4.5 BswTriggers

Remark: This chapter will be reworked with one of the next version of the BSW Coding Guideline.

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.

8.3.4.5.1 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 **BswModuleDescription**.

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:

```
<?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>
      </AR-PACKAGE>
    </AR-PACKAGE>
  </AR-PACKAGES>
  <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>
</AUTOSAR>
```

```

05      ...
10    <ISSUED-TRIGGERS>
15    <trigger-ref-conditional>
20    <trigger-ref dest="trigger">
25    <!-- Refer to the Snippet of the ExternalTrigger -->
      </trigger-ref>
    </trigger-ref-conditional>
  </issued-triggers>
</BSW-SCHEDULABLE-ENTITY>
</entitys>
<events>
  ...
</events>
  ...
</BSW-INTERNAL-BEHAVIOR>
</internal-behaviors>
</BSW-MODULE-DESCRIPTION>
</elements>
</ar-package>
</ar-packages>
</ar-package>
</ar-packages>
</AUTOSAR>
```

30 **Hint** The TriggerRef 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 TriggerRef is written in three lines only for clearness.

35 Example of generated header files for a BSW module: SchM_ModuleName.h

35 This file contains the API for sending an External Trigger:

SchM_Trigger_ModuleName_ExternalTriggerName()

```

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))
 */
45

#ifndef SchM_ModuleName
#define SchM_ModuleName

50 #include "SchM_ModuleName_Type.h"
#include "Rte_Intl.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

55 #if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

60 /* API Mapping Macros */
#ifndef RTE_CORE

# 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 */
# if defined(RTE_PRV_ALL_API) || defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
```

```

05 #define SchM_Trigger_ModuleName_ExternalTriggerName() (SchM_Trigger_ModuleName_ExternalTriggerName())
#endif

10 #endif /* RTE_CORE */
...

```

8.3.4.5.2 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 **BswModuleDescription**.

On the receiver side, the BswScheduledEntity is 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 BswScheduledEntity shall be referenced in a **StartsOnEventRef**.

All these model elements shall exist.

Example for defining BswExternalTrigger on the Receiver side (for a BswScheduledEntity activated by a BswExternalTriggerOccuredEvent):

```

35 <?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>BswModuleEntries</SHORT-NAME>
                <ELEMENTS>
                    <BSW-MODULE-ENTRY>
                        <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
                        ...
                    </BSW-MODULE-ENTRY>
                </ELEMENTS>
            </AR-PACKAGE>
            <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
            <ELEMENTS>
                <BSW-MODULE-DESCRIPTION>
                    <SHORT-NAME>ModuleName</SHORT-NAME>
                    ...
                <REQUIRED-TRIGGERS>
                    <TRIGGER>
                        <SHORT-NAME>ExternalTriggerName</SHORT-NAME>
                    </TRIGGER>
                </REQUIRED-TRIGGERS>
                <INTERNAL-BEHAVIORS>
                    <BSW-INTERNAL-BEHAVIOR>
                        <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
                        ...
                    <ENTITIES>
                        <BSW-SCHEDULABLE-ENTITY>

```

```
05                                     <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 ExternalTrigger -->
        </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>
</AUTOSAR>
```

Hint The TriggerRef 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 TriggerRef is written in three lines only for clearness.

Example of generated header files for a BSW module: SchM_ModuleName.h

This file contains the API for the BswSchedulableEntity (activated by BswExternalTriggerOccuredEvent):

40 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!
   *
   * @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"
55 #ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#if !defined(RTE_RUNNABLEAPI_BswSchedulableEntityName)
#define RTE_PRV_ALL_API
#endif

/* API Mapping Macros */
60 #ifndef RTE_CORE
65 #endif /* RTE CORE */
```

```

05   ****
*** Schedulable Entity Prototypes
*** ****
10  #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 */

15  #ifdef __cplusplus
} /* extern C */
#endif /* __cplusplus */

20  #endif /* SchM_ModuleName */

```

25 8.3.4.5.3 BswInternalTriggers

Rule BSWMD_Trigger_004: Definition of BswInternalTriggers

Instruction For each Internal Trigger, a **BswInternalTriggeringPoint** element shall be defined in a **InternalTriggering-Points** element of a **BswInternalBehavior**.

The BswScheduledEntity is activated by a BswInternalTrigger Occured Event.

35 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 BswScheduledEntity 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 **BswScheduledEntity**.

The concerned BswScheduledEntity 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 BswScheduledEntity):

```

50  <?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>BswModuleEntries</SHORT-NAME>
            <ELEMENTS>
                <BSW-MODULE-ENTRY>
                    <SHORT-NAME>BswModuleEntryName</SHORT-NAME>
                    ...
                </BSW-MODULE-ENTRY>
            </ELEMENTS>
        </AR-PACKAGE>
    <AR-PACKAGE>

```

```

05 <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
10 <ELEMENTS>
15   <BSW-MODULE-DESCRIPTION>
20     <SHORT-NAME>ModuleName</SHORT-NAME>
25     <INTERNAL-BEHAVIORS>
30       <BSW-INTERNAL-BEHAVIOR>
35         <SHORT-NAME>BswInternalBehaviorName</SHORT-NAME>
40         <INTERNAL-TRIGGERING-POINTS>
45           <BSW-INTERNAL-TRIGGERING-POINT>
50             <SHORT-NAME>BswInternalTriggeringPointName</SHORT-NAME>
55             ...
60             </BSW-INTERNAL-TRIGGERING-POINT>
65           </INTERNAL-TRIGGERING-POINTS>
70         <ENTITYS>
75           <BSW-SCHEDULABLE-ENTITY>
80             <SHORT-NAME>Bsw_schedulableEntityName_Sender</SHORT-NAME>
85             ...
90             <ACTIVATION-POINTS>
95               <BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
100                 <BSW-INTERNAL-TRIGGERING-POINT-REF
105                   DEST="BSW-INTERNAL-TRIGGERING-POINT">
110                     <!-- Here add the Snippet for a BswInternalTriggeringPoint -->
115                   </BSW-INTERNAL-TRIGGERING-POINT-REF>
120                 </BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL>
125               </ACTIVATION-POINTS>
130               ...
135             </BSW-SCHEDULABLE-ENTITY>
140             <BSW-SCHEDULABLE-ENTITY>
145               <SHORT-NAME>Bsw_schedulableEntityName_Receiver</SHORT-NAME>
150               ...
155               ...
160             </BSW-SCHEDULABLE-ENTITY>
165           </ENTITYS>
170         <EVENTS>
175           <BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
180             <SHORT-NAME>BswInternalTriggerOccurredEventName</SHORT-NAME>
185             ...
190             <STARTS-ON-EVENT-REF DEST="BSW-SCHEDULABLE-ENTITY">
195               <!-- Refer to the Snippet of the Bsw_schedulableEntity (Receiver) -->
200             </STARTS-ON-EVENT-REF>
205             ...
210             <EVENT-SOURCE-REF DEST="BSW-INTERNAL-TRIGGERING-POINT">
215               <!-- Here add the Snippet for the BswInternalTriggeringPoint -->
220             </EVENT-SOURCE-REF>
225           </BSW-INTERNAL-TRIGGER-OCCURRED-EVENT>
230         </EVENTS>
235         ...
240       </BSW-INTERNAL-BEHAVIOR>
245       </INTERNAL-BEHAVIORS>
250     </BSW-MODULE-DESCRIPTION>
255   </ELEMENTS>
260 </AR-PACKAGE>
265 </AR-PACKAGES>
270 </AR-PACKAGE>
275 </AR-PACKAGES>
280 </AUTOSAR>

```

Hint The BswInternalTriggeringPointRef and EventSourceRef have 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 BswInternalTriggeringPointRef and EventSourceRef are written in three lines only for clearness.

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 BswScheduledEntity (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_BswScheduledEntityName_Sender) &&
!defined(RTE_RUNNABLEAPI_BswScheduledEntityName_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_Trig1MainFunction1)
#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 */
```

05

8.3.5 BSW Service Needs

Will be defined in a later version of the BSW Coding Guideline.

10

8.3.6 BSW Exclusive Areas

15 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 [\[BSW_Sched_001\]](#).

20

25

30

35

40

45

50

55

60

65

05

9 Rule Set: Execution Order Constraint

10

9.1 Introduction to ExecutionOrderConstraint

15

EOC means Execution Order Constraint. This is an AUTOSAR (AR4.x) concept to specify execution order of Executable-Entities respectively AbstractEvents.

20

Ordered elements of an EOC refers either to ExecutableEntities or to AbstractEvents:

- 25
-
- ▶ ExecutableEntities: RunnableEntity (ASW level), BswSchedulableEntity (BSW level)
 - ▶ AbstractEvents: RTEEvent (ASW level), BswEvent (BSW level)

30

Base of EOC is the description of RunnableEntities respectively RTEEvents in SWCD ARXML and the description of BswSchedulableEntities respectively BswEvents in BSWMD ARXML.

35

Order can be described for various scopes:

- 40
-
- ▶ SwComponent (SwcTiming)
 - ▶ BSW Module (BswModuleTiming)
 - ▶ ECU (EcuTiming) --> *preference is EOC on ECU level*
 - ▶ System (SystemTiming)
 - ▶ Virtual Functional Bus (VfbTiming)

45

Order between ExecutableEntities is described using a **successor** and **directSuccessor** relationship.

50

Hint It is recommended to create an EOC file with the tool iSolar. HowTo "EOC generation" with iSolar will be published with V2016.1 (M05/2016), see "iSolar@BoschConnect".

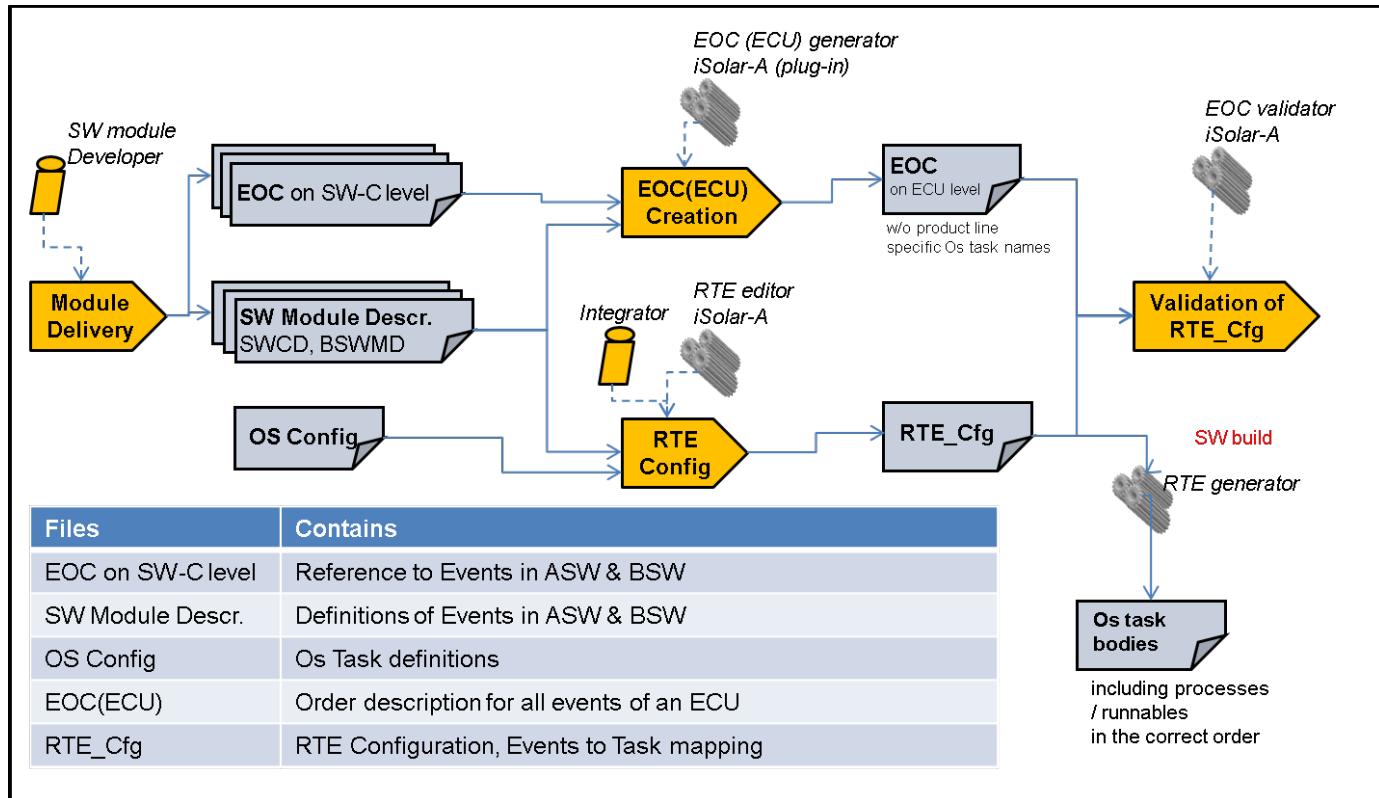
55

Hint Compared to MSR use case DySched an EOC does not contain information about the product line specific Os task in which ExecutableEntities/AbstractEvents are scheduled (Os task names are not standardized!).

60

For more information see the standard AUTOSAR specification "Timing Extensions", [./DocRef/DocRef_AutosarMethod/TPS-GST \[TPS_GST\]](#).

Figure 73 Scheduling and EOC information for validation



9.2 ExecutionOrderConstraint Description

Rule EOC_001: EOC file delivery

Instruction Each SW module (SWCD, BSWMD) which have scheduled entities (AbstractEvents) shall provide an EOC (Execution Order Constraint) file.

Class: NamingConvention

... for an EOC file: **[ModuleName]_EOC.arxml**

If there is no AbstractEvent in a SW module which shall be scheduled by an Os task no EOC file must be provided.

Hint Only event based EOC will be provided that means only objects which will be scheduled by an Os task.

Rule EOC_002: EOC for generated SWCD / BSWMD

Instruction For generated SWCD / BSWMD the corresponding EOC shall also be generated.

If the Os task (the time slot) for an AbstractEvent will be defined by a Customer Integration Team the EOC file must be generated during the SW build by a script file. This script file must be part of the SW module.

05
Figure 74 Static and Dynamic EOC description

static EOC description	dynamic EOC description
ModAAA – SWCD/BSWMD T1 → RE1 (10ms) T2 → RE2 (10ms) T3 → RE3 (10ms)	ModAAA – SWCD/BSWMD T1 → RE1 (10ms <i>or</i> 20ms) T2 → RE2 (10ms) T3 → RE3 (10ms)
ModAAA – EOC RE1 → RE2 → RE3	ModAAA – EOC T1 → RE1 (10ms) RE1 → RE2 → RE3 10ms <i>or</i> ModAAA – EOC T1 → RE1 (20ms) RE1 20ms RE2 → RE3 10ms
T RE	Customer Integration Team must define the Os task for T1 (<i>10ms or 20ms</i>), ModAAA specific EOC file must be generated by BCT/script.

30
Rule EOC_003: EOC file header

Instruction Each EOC ARXML file shall start with a header containing the definition of the used AUTOSAR schema.

35
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-2-1.xsd">

40
45
Hint 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.

It will be done automated if you use the tool iSolar.

50
Rule EOC_004: Scope of an EOC file

Instruction Goal is to describe the correct AbstractEvent sequence order on ECU level. Each EOC file shall have the scope of <ECU-TIMING>.

55
Rule EOC_005: EOC relevant objects

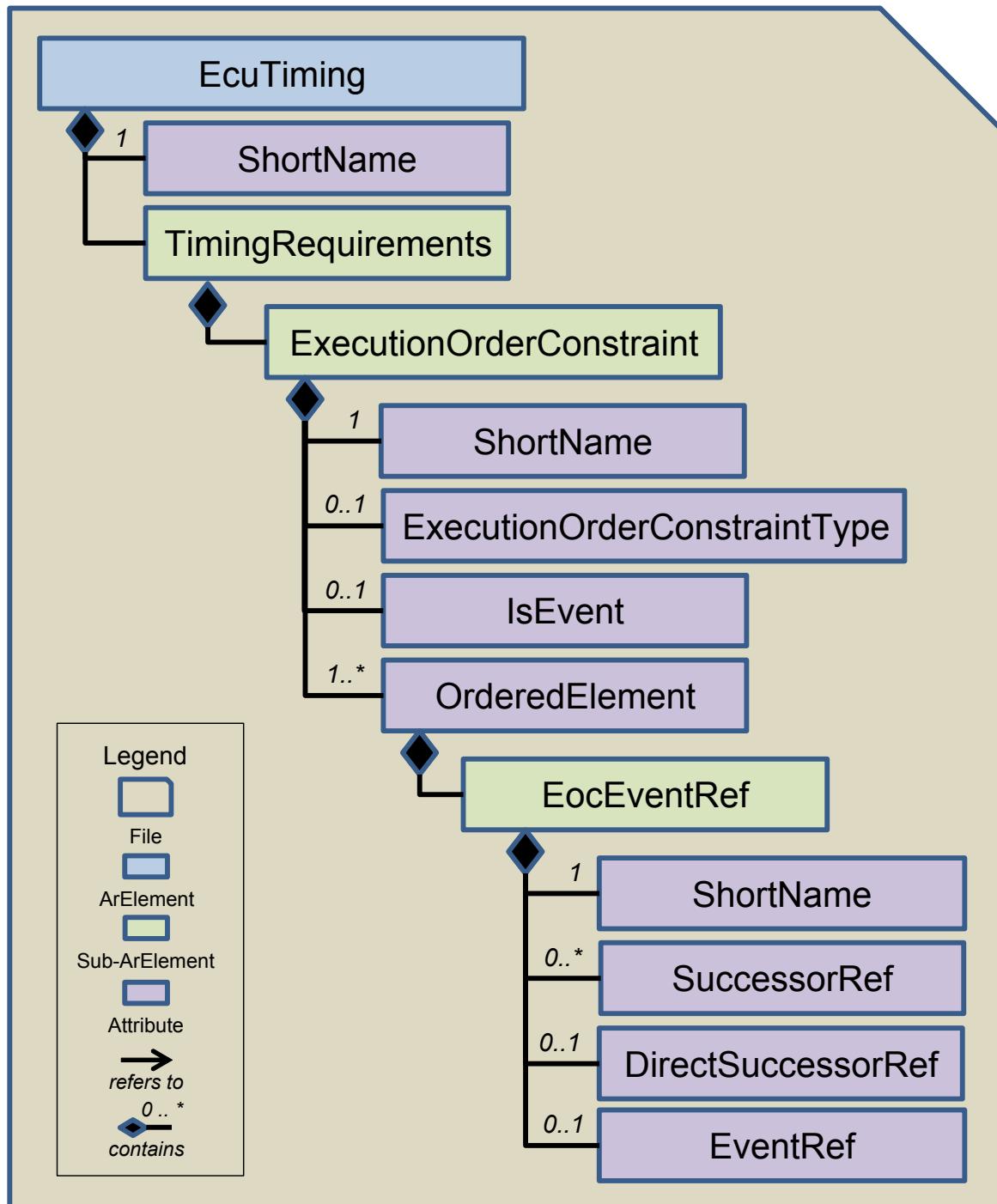
Instruction Each AbstractEvent shall be referenced by a <EXECUTION-ORDER-CONSTRAINT>. Each <EXECUTION-ORDER-CONSTRAINT> should refer to AbstractEvents and not to ExecutableEntities.

60
65
The element <EOC-EVENT-REF> is used to reference **RTEEvents** or **BswEvents**.

The <EXECUTION-ORDER-CONSTRAINT-TYPE> shall be set to *ORDINARY-EOC*.

The Attribute <IS-EVENT> shall be set to *true*.

Figure 75 EOC ARXML structure



EOC file example (ModBBB_Proc_10ms is successor of ModAAA_Proc_10ms)

```

<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-2-1.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>RB</SHORT-NAME>
        <AR-PACKAGES>
            <AR-PACKAGE>
                <SHORT-NAME>PCT_ModAAA</SHORT-NAME>
            <ELEMENTS>
    
```

```

05   <ECU-TIMING>
10     <SHORT-NAME>ModAAA_EcuTiming</SHORT-NAME>
15     <TIMING-REQUIREMENTS>
20       <EXECUTION-ORDER-CONSTRAINT>
25         <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
30         <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
35         <IS-EVENT>true</IS-EVENT>
40       <ORDERED-ELEMENTS>
45         <EOC-EVENT-REF>
50           <SHORT-NAME>EOCEventRef_ModAAA</SHORT-NAME>
55           <BSW-MODULE-INSTANCE-REF DEST="BSW-IMPLEMENTATION">/RB/PCT_ModAAA/Impl_ModAAA
60           </BSW-MODULE-INSTANCE-REF>
65             <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModAAA/ModAAA/IB_ModAAA/T_10ms
70             </EVENT-REF>
75           <SUCCESSOR-REFS>
80             <SUCCESSOR-REF DEST="EOC-EVENT-REF">/RB/PCT_ModBBB/ModBBB_EcuTiming
85               /EOC_T_10ms/EOCEventRef_ModBBB
90             </SUCCESSOR-REF>
95           </SUCCESSOR-REFS>
100         </EOC-EVENT-REF>
105       <EOC-EVENT-REF>
110         <SHORT-NAME>EOCEventRef_ModBBB</SHORT-NAME>
115         <BSW-MODULE-INSTANCE-REF DEST="BSW-IMPLEMENTATION">/RB/PCT_ModBBB/Impl_ModBBB
120         </BSW-MODULE-INSTANCE-REF>
125           <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModBBB/ModBBB/IB_ModBBB/T_10ms
130           </EVENT-REF>
135         </EOC-EVENT-REF>
140       </ORDERED-ELEMENTS>
145     </EXECUTION-ORDER-CONSTRAINT>
150   </TIMING-REQUIREMENTS>
155   </ECU-TIMING>
160 </ELEMENTS>
165 </AR-PACKAGE>
170 </AR-PACKAGES>
175 </AR-PACKAGE>
180 </AR-PACKAGES>
185 </AUTOSAR>

```

45 Rule EOC_006: EOC sequencing strategy

50 **Instruction** Each ordered element shall have a **successor** (<SUCCESSOR-REF>) or a **directSuccessor** (<DIRECT--
SUCCESSOR-REF>).

55 It is possible to describe the process/runnable sequence out of the successor view (use case 1a) or out of the
predecessor view (use case 1b).

60 **Hint** You can describe the sequence order only with **successor** / **directSuccessor**. AUTOSAR doesn't provide a
predecessor tag.

65 *use case 1:* If Proc1 is successor of Proc2 and Proc2 must not follow directly on Proc1 **successor** should be used.

70 ▶ In this case it doesn't matter where Proc2 is located in the Os task, only after Proc1.

75 *use case 2:* If Proc1 is the direct successor of Proc2 use **directSuccessor**.

80 ▶ In this case the sequence is hard coded, Proc2 directly after Proc1.

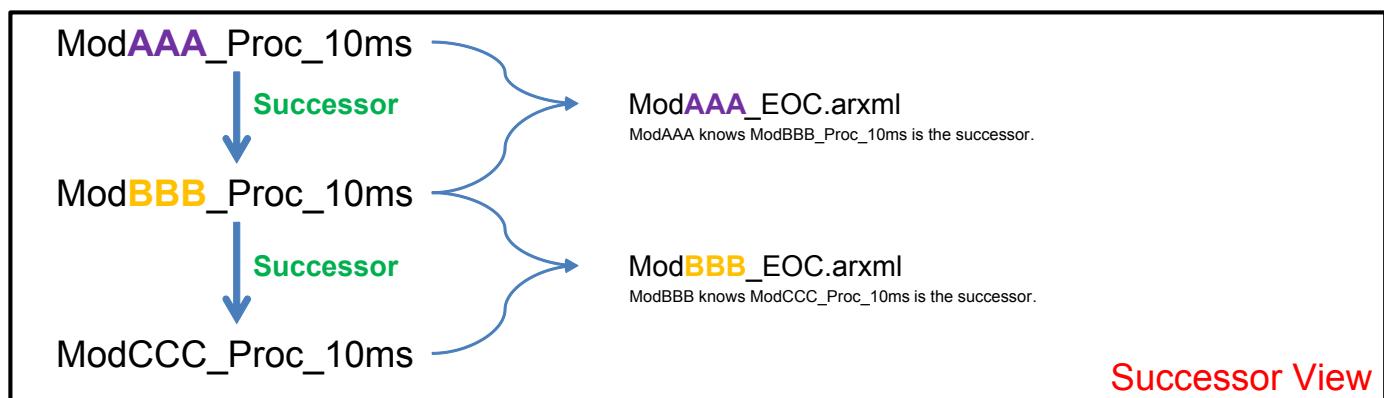
use case 3: Proc1 is autonomous, i.e. can be called from everywhere.

- ▶ [ModuleName]_EOC.arxml shall be provided but with no dependencies to other processes/runnables.

use case 4: If an **AbstractEvent** is legal removable cramp this **AbstractEvent** with a <VARIATION-POINT>.

- ▶ With a variation point you can define if a part exists or not.

Figure 76 EOC sequencing strategy, use case 1a, Successor View



EOC file examples, Successor View (only the essential parts)

```
ModAAA_EOC.arxml
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModAAA</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModAAA/ModAAA/IB_ModAAA/T_10ms</EVENT-REF>
      <SUCCESSOR-REF DEST="EOC-EVENT-REF">/RB/PCT_ModBBB/ModBBB_EcuTiming/EOC_T_10ms
        /EOCEventRef_ModBBB</SUCCESSOR-REF>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModBBB</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModBBB/ModBBB/IB_ModBBB/T_10ms</EVENT-REF>
```

ModBBB_EOC.arxml

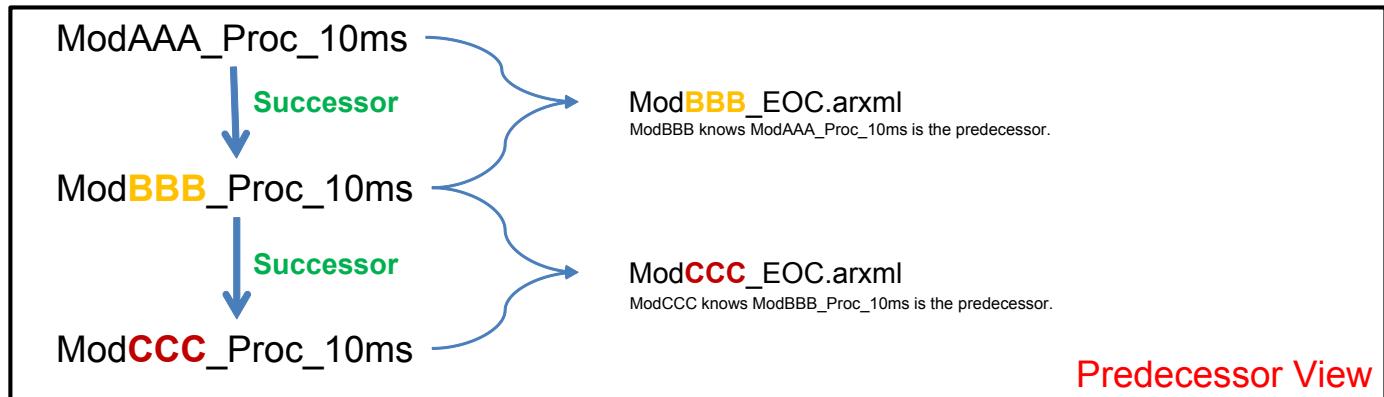
```
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModBBB</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModBBB/ModBBB/IB_ModBBB/T_10ms</EVENT-REF>
      <SUCCESSOR-REF DEST="EOC-EVENT-REF">/RB/PCT_ModCCC/ModCCC_EcuTiming/EOC_T_10ms
        /EOCEventRef_ModCCC</SUCCESSOR-REF>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModCCC</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModCCC/ModCCC/IB_ModCCC/T_10ms</EVENT-REF>
```

ModCCC_EOC.arxml

```
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModCCC</SHORT-NAME>
```

```
<EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModCCC/ModCCC/IB_ModCCC/T_10ms</EVENT-REF>
```

Figure 77 EOC sequencing strategy, use case 1b, Predecessor View

**EOC file examples, Predecessor View (only the essential parts)**

ModAAA_EOC.arxml

```
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModAAA</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModAAA/ModAAA/IB_ModAAA/T_10ms</EVENT-REF>
```

ModBBB_EOC.arxml

```
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModBBB</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModBBB/ModBBB/IB_ModBBB/T_10ms</EVENT-REF>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModAAA</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModAAA/ModAAA/IB_ModAAA/T_10ms</EVENT-REF>
      <SUCCESSOR-REF DEST="EOC-EVENT-REF">/RB/PCT_ModBBB/ModBBB_EcuTiming/EOC_T_10ms
        /EOCEventRef_ModBBB</SUCCESSOR-REF>
```

ModCCC_EOC.arxml

```
<EXECUTION-ORDER-CONSTRAINT> <SHORT-NAME>EOC_T_10ms</SHORT-NAME>
  <EXECUTION-ORDER-CONSTRAINT-TYPE>ORDINARY-EOC</EXECUTION-ORDER-CONSTRAINT-TYPE>
  <IS-EVENT>true</IS-EVENT>
  <ORDERED-ELEMENTS>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModCCC</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModCCC/ModCCC/IB_ModCCC/T_10ms</EVENT-REF>
    <EOC-EVENT-REF> <SHORT-NAME>EOCEventRef_ModBBB</SHORT-NAME>
      <EVENT-REF DEST="BSW-TIMING-EVENT">/RB/PCT_ModBBB/ModBBB/IB_ModBBB/T_10ms</EVENT-REF>
      <SUCCESSOR-REF DEST="EOC-EVENT-REF">/RB/PCT_ModCCC/ModCCC_EcuTiming/EOC_T_10ms
        /EOCEventRef_ModCCC</SUCCESSOR-REF>
```

Figure 78 EOC sequencing strategy, use case 2

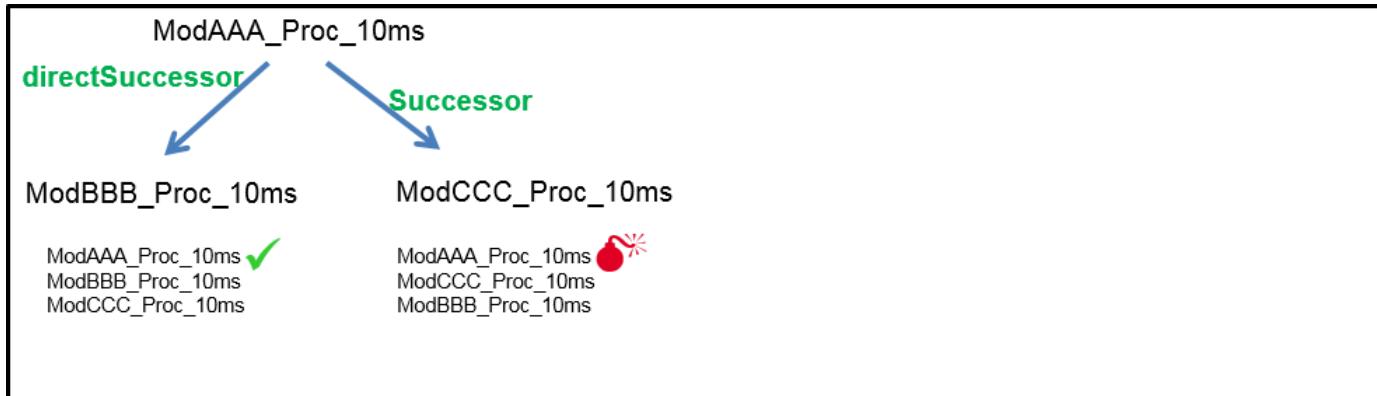
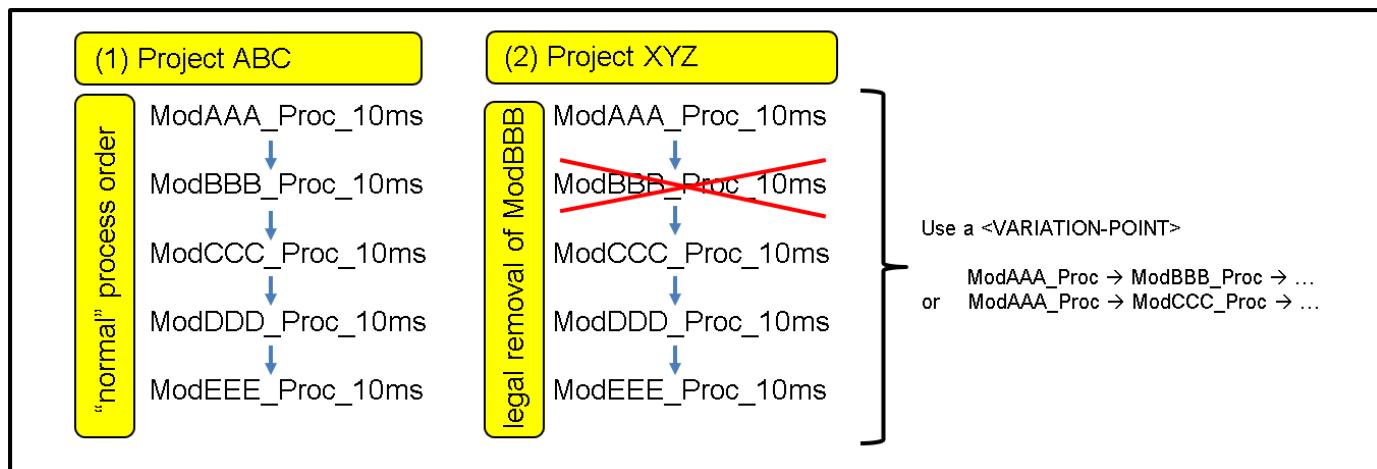


Figure 79 EOC sequencing strategy, use case 4 (ModBBB is legal removable!)



Rule EOC_007: Content of an <EXECUTION-ORDER-CONSTRAINT>

Instruction All AbstractEvents in an <EXECUTION-ORDER-CONSTRAINT> shall be compatible regarding their recurrence. So all <EOC-EVENT-REF> in an <EXECUTION-ORDER-CONSTRAINT> shall be of the same event type (TimingEvent, InitEvent, ...) and of the same event property (period: 10ms, 20ms, ...).

Table 61 AUTOSAR conform AbstractEvents (Event Types)

RTEEvents (ASW)	BswEvents (BSW)	Abbreviation
TimingEvent	BswTimingEvent	T
BackgroundEvent	BswBackgroundEvent	BG
DataReceivedEvent	BswDataReceivedEvent	DR
DataReceiveErrorEvent	-	DRE
DataSendCompletedEvent	-	DSC
DataWriteCompletedEvent	-	DWC
OperationInvokedEvent	BswOperationInvokedEvent	OI
AsynchronousServerCallReturnsEvent	BswAsynchronousServerCallReturnsEvent	ASCR
SwcModeSwitchEvent	BswModeSwitchEvent	MS
ModeSwitchedAckEvent	BswModeSwitchedAckEvent	MSA
SwcModeManagerErrorEvent	BswModeManagerErrorEvent	MME

RTEEvents (ASW)	BswEvents (BSW)	Abbreviation
ExternalTriggerOccurredEvent	BswExternalTriggerOccurredEvent	ETO
InternalTriggerOccurredEvent	BswInternalTriggerOccurredEvent	ITO
InitEvent	-	I
TransformerHardErrorEvent	-	THE

Rule EOC_008: Naming Convention for an <EOC-EVENT-REF>

Instruction

Class: NamingConvention

... for an <EOC-EVENT-REF>: **EOC_[EventType]_[EventProperty]**

The element <EOC-EVENT-REF> shall contain information about the event type and event property.

e.g.: EOC_T_10ms

- ▶ event type: T <BSW-TIMING-EVENT>
- ▶ event property: 10ms (period of <BSW-TIMING-EVENT>)

Hint Input for the Customer Integration Team to know in which task the referenced entities should be scheduled.

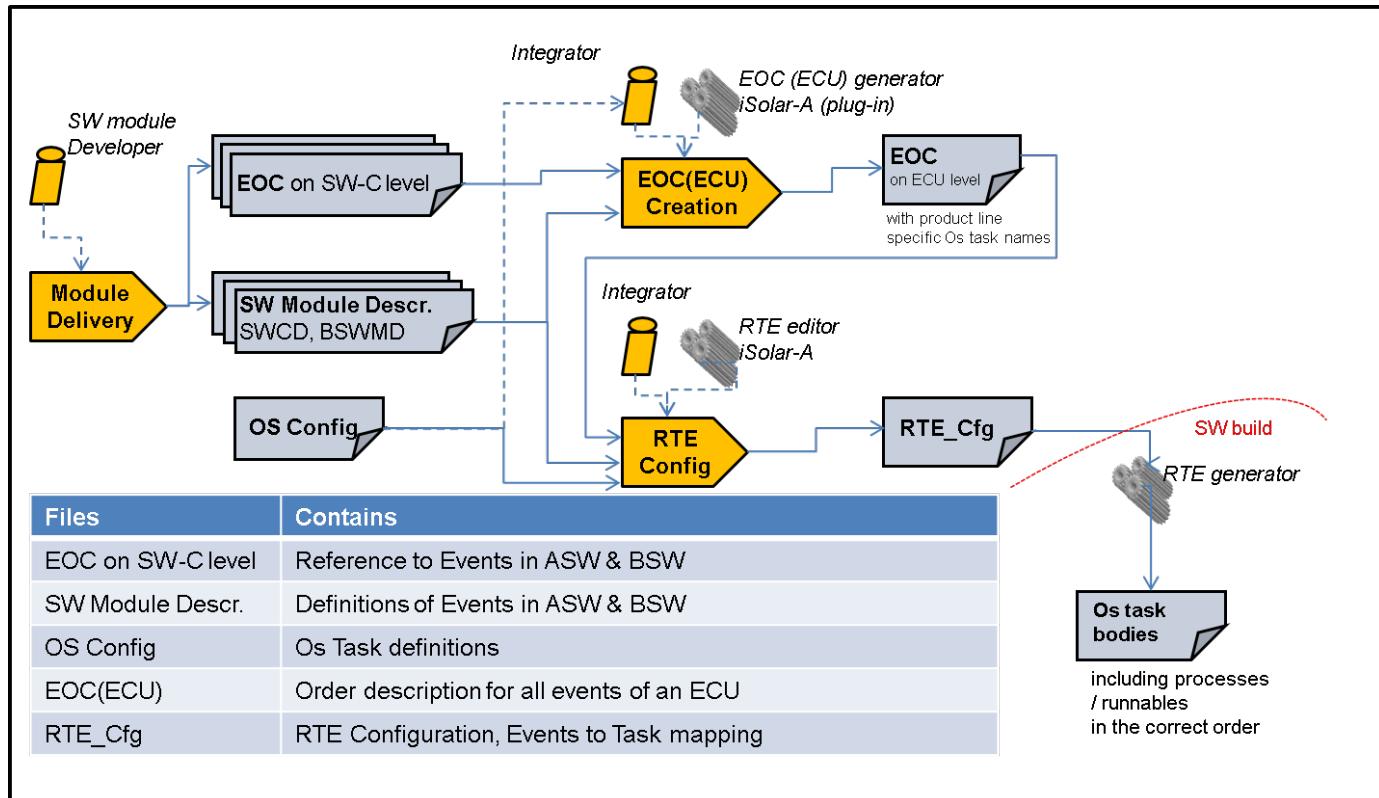
List of legal event types and their abbreviations see Table [/ARAbstractEvents "AUTOSAR conform AbstractEvents \(Event Types\)"](#).

The name of an event property correlates to the name of an Os task, like "10ms", "50ms", "ini".

9.3 Upcoming of ExecutionOrderConstraint

In future the combined information regarding AUTOSAR conforming scheduling and EOC could be used for RTE configuration.

Figure 80 Scheduling and EOC information as input for RTE configuration



10 Rule Set: AUTOSAR Package Structure

10.1 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 *ARPackages*.

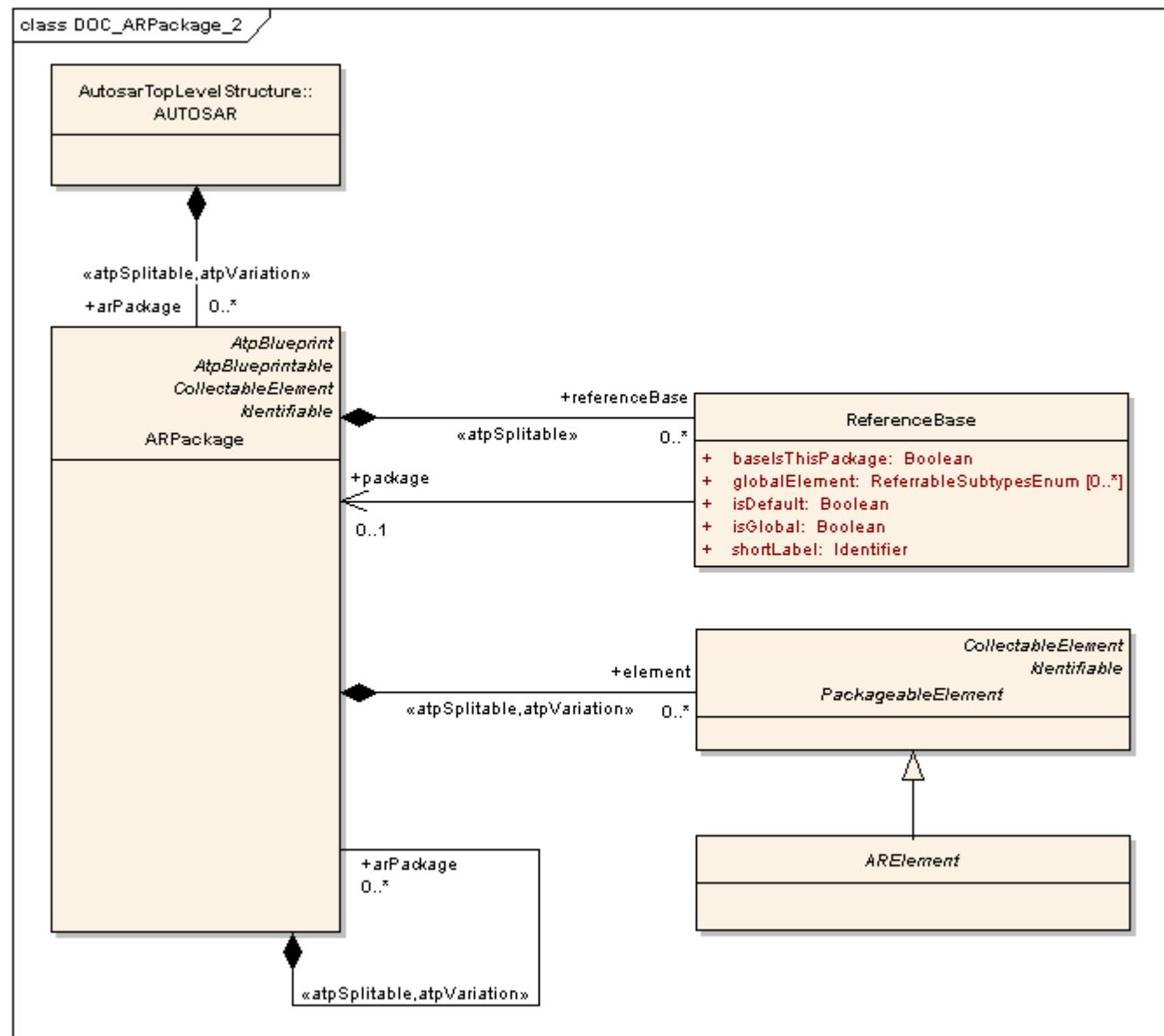
ARPackages define name spaces on arxml level. All the included *ARElements* shall have different *ShortNames*. E.g. if a *Unit* and a *CompuMethod* should have the same name you have to put them into different *ARPackages*.

ARPackages can be nested.

ARPackages can be split over several files.

The following figures show the *ARPackages* and its related objects for AUTOSAR 4.0.3.

Figure 81 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 *ARPackages* structure in a real .arxml file looks like.

```

05  <?xml version="1.0" encoding="UTF-8"?>
06  <AUTOSAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
07  xmlns="http://autosar.org/schema/r4.0"
08  xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
09    <ADMIN-DATA>
10      <USED-LANGUAGES><L-10 L="EN" xml:space="preserve">EN</L-10></USED-LANGUAGES>
11    </ADMIN-DATA>
12    <AR-PACKAGES>
13      <AR-PACKAGE>
14        <SHORT-NAME>AUTOSAR_CanIf</SHORT-NAME>
15        <AR-PACKAGES>
16          <AR-PACKAGE>
17            <SHORT-NAME>BswModuleDescriptions</SHORT-NAME>
18          </AR-PACKAGE>
19          <AR-PACKAGE>
20            <SHORT-NAME>DataConstraints</SHORT-NAME>
21          </AR-PACKAGE>
22          <AR-PACKAGE>
23            <SHORT-NAME>CompuMethods</SHORT-NAME>
24          </AR-PACKAGE>
25          <AR-PACKAGE>
26            <SHORT-NAME>Units</SHORT-NAME>
27          </AR-PACKAGE>
28          <AR-PACKAGE>
29            <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
30          </AR-PACKAGE>
31        </AR-PACKAGES>
32      </AR-PACKAGE>
33    </AR-PACKAGES>
34  </AUTOSAR>

```

10.1.1 Splitting ARPackages into different files

In figure 81 you see the stereotype «*atpSplittable*». 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.

Tool support is not easy and thus still not handled completely.

10.1.2 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 provides functionality for entering a reference and you don't have to type this deep path manually.

10.1.3 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 *ReferenceBase* representing an *ARPackage* path. In a reference you can then use a path relative to the reference base.

05 If the *ReferenceBase* changes you have to implement this change on one single position. Till AUTOSAR release 4.0.2 the *ReferenceBase* itself is not <>splitable<>. Starting with AUTOSAR release 4.0.3 the *ReferenceBase* itself is now <>splitable<>.

10 Example for using a *ReferenceBase*:

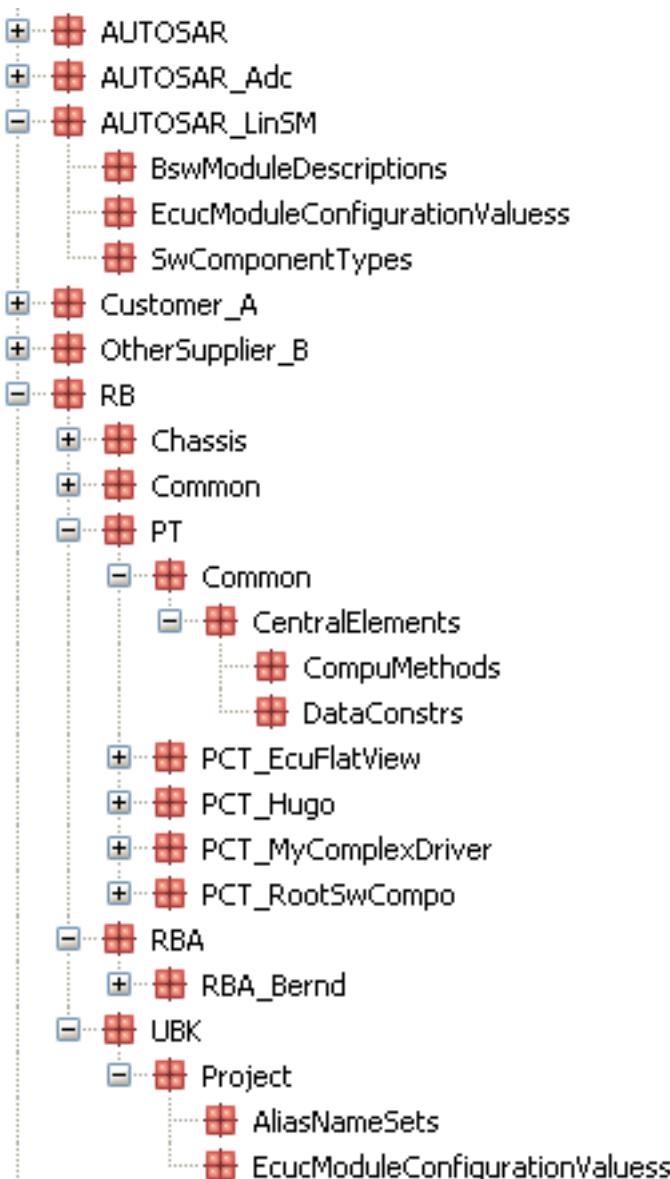
```
<UNIT-REF BASE="PT_CEL_Units" DEST="UNIT">Rpm</UNIT-REF>
```

15 10.2 Main Structure of ARPackages

Overview

20 This picture gives an overview of the nesting of ARPackages. Following chapters describe the detailed rules w.r.t the topics.

25 Figure 82 Overview of ARPackage structure



60 This shows the ARPackage structure of an ECU.

65 Of course these packages are spread over a lot of files.

05

Rule ARPac_10: Main Pattern for Package Structure

10

Instruction

10 **Scope:** RB deliverables, except AUTOSAR specified software (e.g. BSW modules, Application Interfaces (AISpecification))

15 **Class:** NamingConvention

15 The following structure should be followed:

20 /RB
 /{domain}
 [/{subdomain}]*
 /{block}
 /{kind}[_Blueprint|_Example]

25 where

- 25
- ▶ 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 [\[ARPac_14\]](#)

30 For an AUTOSAR specified BSW module there is another package structure to be used. See rule [\[ARPac_41\]](#).

30 For ECU Configuration values a special *ARPackage* structure is to be used ("/RB/UBK/Project/..."). See rule [\[ARPac_46\]](#)

35 .

35 For more information see [\[TPS_GST\]](#) for *TPS_GST_00083* and *TPS_GST_00017*.

40

Rule ARPac_11: Top Level *ARPackage*

45

Instruction

45 **Scope:** RB deliverables, except AUTOSAR specified software (e.g. BSW modules)

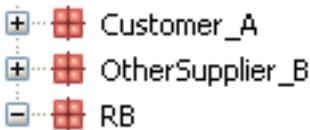
50 **Class:** NamingConvention

50 There shall be exactly one top level *ARPackage* in an .arxml file.

55 The top level package shall be named "RB".

55 Example:

55 Figure 83



55 **Note:** this figure shows the structure of *ARPackages* (which typically will be split over several files).

60 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.

65 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*.

05
Rule ARPac_12: Domain oriented ARPackages10
Instruction

10 **Scope:** RB deliverables, except AUTOSAR specified BSW modules

15 **Class:** NamingConvention

The package name on second level should be the name of the ECU-domain, as defined in [Table 62](#).

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.

20 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 62 Second Level ARPackage

short name	used for
PT	Power Train
AS	Active Safety
DA	Driver Assistance
PS	Passive Safety
--tbd--	For other ECU domains a short name will be defined.
UBK	an ARPackage for non-domain-specific objects
RBA	RB's first implementation of AUTOSAR BSW: CUBAS
Common	for objects which are used in all domains / sub domains

40 Example:

Figure 84



55 **Hint** Central Elements Guideline Artifact ([\[A_CEL\]](#)) defines detailed rules how to use ARPackages for central elements.

See rules [CEL_011](#) and [CEL_020](#).

60 Rule ARPac_13: Removed

65 Rule ARPac_14: Names for ARElement-kind based ARPackages

70 Instruction

Class: NamingConvention

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. A complete List is given in [Table 63 \[Recommended Packages\]](#)

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 63 Recommended Packages

Element kind	recommendedPackage
AclObjectSet	AclObjectSets
AclOperation	AclOperations
AclPermission	AclPermissions
AclRole	AclRoles
AliasNameSet	AliasNameSets
ApplicationArrayType	ApplicationDataTypes
ApplicationPrimitiveDataType	ApplicationDataTypes
ApplicationRecordDataType	ApplicationDataTypes
ApplicationSwComponentType	SwComponentTypes
BlueprintMappingSet	BlueprintMappingSets
BswImplementation	BswImplementations
BswModuleClientServerEntry	BswModuleEntries
BswModuleDescription	BswModuleDescriptions
BswModuleEntry	BswModuleEntries
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

05

10

15

20

25

30

35

40

45

50

55

60

65

70

Element kind	<i>recommendedPackage</i>
EcuTiming	TimingExtensions
EcucDefinitionCollection	EcucDefinitionCollections
EcucModuleConfigurationValues	EcucModuleConfigurationValue
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
FlexrayTpConfig	TpConfigs
Gateway	Gateways
HwCategory	HwCategories
HwElement	HwElements
HwType	HwTypes
ISignal	ISignals
ISignalGroup	ISignalGroups
ISignalPdu	Pdus
ISignalPduGroup	ISignalPduGroup
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

Element kind	<i>recommendedPackage</i>
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	PostBuildVariantCriterions
PostBuildVariantCriterionValueSet	PostBuildVariantCriterionValueSets
PredefinedVariant	PredefinedVariants
RapidPrototypingScenario	RapidPrototypingScenarios
SenderReceiverInterface	PortInterfaces
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

Element kind	<i>recommendedPackage</i>
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 *ARPackge* contains no *ARElements* you may omit it.

Rule ARPac_16: *ARPackge* for Common *ARElements*

Instruction

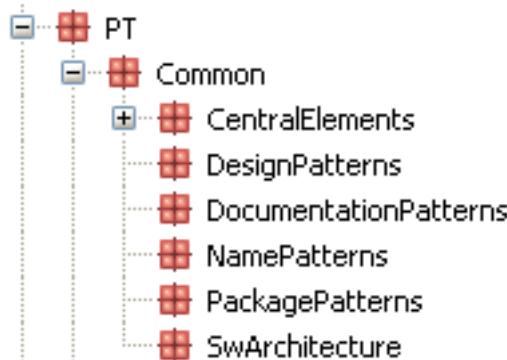
Class: NamingConvention

There are different kinds of common *ARElements*. They shall be covered by an *ARPackge* named "Common". The currently known common element types are: CentralElements, DesignPatterns, DocumentationPatterns, NamePatterns, PackagePatterns, SwArchitecture, NamingConventions.

The Common- *ARPackages* shall be used on that *ARPackge*-level for which the elements are common. This can be the level "RB" or the level of a domain or of a sub domain.

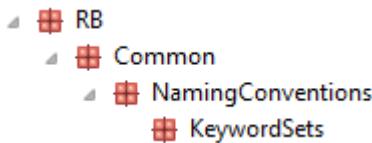
Examples for the Common- *ARPackge*:

Figure 85 Possible domain for common *ARPackages*



The meaning of these *ARPackages* can be found in Guideline Artifact [\[A_Blueprint\]](#).

Figure 86 Possible UBK common *ARPackages*



More information on KeywordSets can be found in [\[A_AppII\]](#) [\[A_AppIII\]](#).

Rule ARPac_17: Category of ARPackages

Instruction

The following *Categorys* 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. The *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 *Categorys* `ICS`, `EXAMPLE` shall NOT be used since the usage is not defined in our methods.

10.3 ARPackage for Basic Software

Rule ARPac_41: Top level ARPackages for AUTOSAR Specified BSW Modules

Instruction

Class: NamingConvention

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 64 \[BSWModuleNames\]](#) , taken from AUTOSAR specification 4.1.1.

Table 64 List of AUTOSAR specified BSW-Modules (AR 4.1.1)

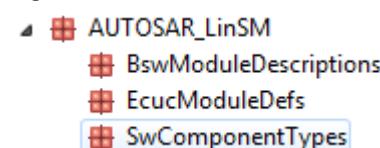
	list of BSW modules	source of this list
BSW module list	Adc, Com, BswM, SchM, Can, CanIf, CanNm, CanSM, CanTrcv, CanTp, ComM, CorTst, Csm, Dbg, Det, Dcm, Dem, Dlt, DoIP, Dio, EcuM, Ea, Eep, Eth, EthIf, EthTrcv, UdpNm, EthSM, Fls, Fee, FlsTst, Fr, FrIf, FrNm, FrSM, FrTrcv, FrArTp, FrTp, FiM, Gpt, Icu, 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 ([TR_BSWModuleList]) For more information see <i>TPS-GST_00017</i> in [TPS_GST]
virtual modules	AI Specification, Compiler, Comtype, EcuC, GenDef, Platform, Std	<i>TR_PDN_00001</i> in [TR_PDN]

Note: This rule is derived from *TPS_GST_00083* as defined in [\[TPS_GST\]](#).

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 87



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 ARPackets for non AUTOSAR Specified BSW Modules

Instruction

Scope: BSW modules not specified by AUTOSAR

Class: NamingConvention

Such modules shall be put into the top level ARPackage "RB".

They shall be put into the second level ARPackage for the respective 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 respective 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:

05
Figure 88

- ▲ RB [ARPackage]
 - ▷ Common [ARPackage]
 - ▲ PT [ARPackage]
 - ▷ Common [ARPackage]
 - ▲ PCT_MyComplexDriver [ARPackage]
 - BswModuleDescriptions [ARPackage]
 - EcucModuleDefs [ARPackage]
 - SwComponentTypes [ARPackage]
- ▲ RBA [ARPackage]
 - ▲ RBA_Bernd [ARPackage]
 - BswModuleDescriptions [ARPackage]
 - EcucModuleDefs [ARPackage]
 - SwComponentTypes [ARPackage]

10
20
25

Rule ARPac_44: Kind ARPackages for Basic Software Modules

Instruction

Class: NamingConvention

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.

30
35
30
Possible examples are:

Figure 89 Kind ARPackages for Basic Software Modules

- ▲ AUTOSAR_LinSM
 - BswModuleDescriptions
 - EcucModuleDefs
 - SwComponentTypes

40
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.

45

Rule ARPac_47: ARPackage for BSW Component Types

Instruction

Class: NamingConvention

50
Each *SwComponentType* (e.g. *ServiceSwComponentType*) which is defined as part of a BSW module shall be put into the *ARPackage* as defined in [/ARPac_41 \[ARPac_41\]](#) / [/ARPac_43 \[ARPac_43\]](#) and [/ARPac_44 \[ARPac_44\]](#).

Note: This rule is not applicable for *SwComponentTypes* which are part of ASW.

55
Example see: [/KindARPackagesForBSWModules Figure 89](#).

Rule ARPac_46: Software Module Configuration (for protected values)

Instruction

60
Protected configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following *ARPackage* structure:

```
/RB
  /UBK
    /Project
      /EcucModuleConfigurationValue
```

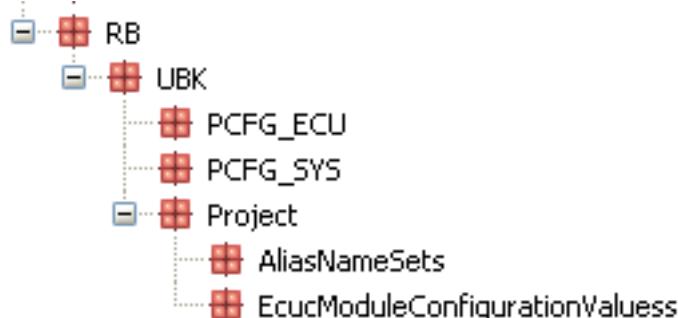
05
Note: the word "Project" is a keyword and shall not be replaced by any project's name. Reason: Tools will merge all configuration values coming from different files only if they are elements of the same *ARPackage*.

10
Protected configuration values shall **NOT** be put into the BSW module's *ARPackage* (e.g. /AUTOSAR_LinSM or /RB/RBA/RBA_Bernd).

15
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:

15
Figure 90



20
25
Note: the double "s" at the end of EcucModuleConfigurationValueess is not a misspelling. It is the result of the common rule to add a plural-s to the element kind.

30
35
Rule ARPac_45: Software Module Configuration (pre-configured or recommended values) --- currently **NOT** used

40
45
Instruction **Scope:** upcoming

Pre-configured or recommended ECUC values for BSW modules (AUTOSAR specified or not) shall be put into the *ARPackage* of the module itself and within that into a sub package EcucModuleConfigurationValueess.

Examples:

/AUTOSAR_LinSM/EcucModuleConfigurationValueess
/RB/RBA/rba_Bernd/EcucModuleConfigurationValueess

For pre-configured values the container's *ShortName* shall be *PreconfiguredValues*.

For recommended values the container's *ShortName* shall be *RecommendedValues*.

50
55
Note: Currently neither pre-configured nor recommended values are really defined. So, currently this rule will not be used.

10.4 Referencing Elements

10.4.1 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:

60
<UNIT-REF DEST="UNIT"/>/RB/PT/Common/CentralElements/Units/Rpm</UNIT-REF>

If you use a specific AUTOSAR authoring tool it typically provides functionality for entering a reference and you don't have to type this path manually.

65
No rules are defined in this chapter since it is defined by AUTOSAR how to use a reference.

10.4.2 Use of ReferenceBases

With AUTOSAR release 4 the mechanism of *ReferenceBases* 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>CUCEL_Units</SHORT-LABEL>
    <IS-DEFAULT>false</IS-DEFAULT>
    <IS-GLOBAL>false</IS-GLOBAL>
    <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
    <PACKAGE-REF DEST="AR-PACKAGE">
      /RB/RBA/Common/CentralElements/Units</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
```

Then, inside an *ARPackage* you can refer to an object in the following form:

```
<UNIT-REF BASE="CUCEL_Units" DEST="UNIT">Rpm</UNIT-REF>
```

Rule ARPac_71: Avoid Usage of Element ReferenceBases

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 is dangerous when someone changes the *PackageRef* of a *ReferenceBase* expecting that all references are affected.

However, this requires a functionality in authoring tools that, when creating a new reference, existing reference base really will be used.

Rule ARPac_77: Order of attributes in references

Instruction

The required attributes of a reference shall be in the following order: first *Base*, then *Dest*.

Example:

```
<UNIT-REF BASE="PT_CEL_Units" DEST="UNIT">Rpm</UNIT-REF>
```

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

Class: NamingConvention

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 [\[ARPac_14\]](#) (including the plural-s).

Examples: PTCEL_Units; CUCEL_RecordLayouts.

Rule ARPac_76: All attributes of *ReferenceBases* shall be specified explicitly

Instruction

All attributes of *ReferenceBases* shall be specified explicitly.

Attribute *IsDefault* shall always be set to false.

Attribute *IsGlobal* shall always be set to false.

Attribute *BaselsThisPackage* shall always be set to false.

Attribute *PackageRef* shall always be specified.

This is a strict definition. As soon as use cases are defined which require other settings this rule will be changed.

Example:

```
<REFERENCE-BASES>
  <REFERENCE-BASE>
    <SHORT-LABEL>CUCEL_Units</SHORT-LABEL>
    <IS-DEFAULT>false</IS-DEFAULT>
    <IS-GLOBAL>false</IS-GLOBAL>
    <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
    <PACKAGE-REF DEST="AR-PACKAGE">
      /RB/RBA/Common/CentralElements/Units</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
```

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 ArPackage as their usage in references. This could be the top level ArPackage eg. of a component or module.

Background of this rule is, that a delivery unit using *ReferenceBases* could change the whole behavior when the reference to other objects changes.

Blueprints for *ReferenceBases* for referencing Central Elements will be defined in central file(s) and might be copied to the files where they will be used. See *CEL_116* in CEL guideline V1.4.

11 Rule Set: Central Elements

Criteria for Central Elements

An element should be central (i.e. defined centrally) if it is expected to be used by a large number of product lines in the context of BBM wide software. An element should be central for a product line, if it is expected to be used by a large number of components or modules.

- ▶ Only if an element is about to be used by several components/modules or an element may be potentially used by many users, e.g. all OEMs and suppliers, then centralization is valuable.
- ▶ When creating a new central element it shall be checked whether a similar definition already exists. If an appropriate element is already present the existing one shall be used. Otherwise a new one shall be defined centrally.
- ▶ The central elements shall be generally and well-defined (check its uniqueness) to avoid uncontrolled growth as well as duplicates. The semantic definition of a central element shall be invariant and no project-specific variants shall be provided.
- ▶ The number of central elements shall be kept small and manageable.

Note: If an element shall be defined centrally it is assumed that these criteria are fulfilled. Otherwise it should not be defined centrally.

11.1 Centralization of Element Kinds

Rule CEL_120: Dealing with Central Elements

Instruction If a central element exists, it shall be used.

In exceptional cases elements might be needed earlier than being provided in a central package. In this case it is allowed to define the element locally first. In parallel a request for a central element shall be triggered and a first consensus with the provider of the central package should be found. The switch from the local element to the central element shall be done in the next refactoring step of the component.

These local elements shall contain a local ARPackage path, i.e. this path is local to the corresponding module and must not be referenced outside this module. Then, only the ARPackage path has to be adjusted pointing to the central element once it is supported.

All elements that are not explicitly mentioned to be defined as central elements within these central elements guidelines shall be defined locally only.

11.1.1 Tabular Overview of Central Elements

The following two tables provide an overview of all the elements that are exclusively defined centrally and of all the central elements where a local definition may also exist.

Table 65 List of exclusively centrally defined elements

<i>Central Elements (only central)</i>
<i>Unit</i>
<i>PhysicalDimension</i>
<i>SwAddrMethod</i>
<i>SwRecordLayout</i>
<i>KeywordSet</i>
<i>Compiler Abstraction Macros</i>
<i>Sw BaseType</i>

Table 66 List of centrally defined elements that may also be defined locally

<i>Central Elements (also local)</i>
<i>UnitGroup</i>
<i>CompuMethod</i>
<i>SwSystemconst</i>
<i>ImplementationDataType</i>
<i>ApplicationDataType</i>
<i>PortInterface</i>
<i>DataConstraint</i>
<i>DataTypeMappingSet</i>
<i>ReferenceBase</i>

11.2 How to use the Central Elements

The common ARPackage "CentralElements" covers all different kinds of central elements. For each kind of central ARElement a separate ARPackage shall be created (refer to rules ARPac_21, ARPac_22 in the ARPackage Guidelines).

Rule CEL_011: ARPackage Structure for Central Elements

Instruction

Scope: Bosch defined central elements

The path declaration for central elements shall be of the form:

/RB/{domain}/Common/CentralElements/{kind}s

A list of all possible domains can be found in Rule ARPac_12 of the ARPackage Guideline.

Note: AUTOSAR defined standardized objects which will use other package paths.

For instance the path declaration of central elements for BSW is /RB/RBA/Common/CentralElements/{kind}s.

Rule CEL_020: Referencing Central Elements

Instruction

Scope: ASW

An ASW product line shall not use central elements of another product line. An ASW product line may use BBM 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, /AUTOSAR_*/ and /RB/PT/Common/CentralElements.

E. g. ASW PT shall not use /RB/{Chassis}/Common/CentralElements.

CUBAS may access /AUTOSAR/, /AUTOSAR_* and /RB/RBA, but not /RB/Common/CentralElements.

Reason: Self-containment of CUBAS, because CUBAS should be delivered as separate product.

Rule CEL_117: Usage of ReferenceBases

Instruction

DerivedFrom: ARPac_73: A defined *ReferenceBase* shall always be used

05 If a centrally defined *ReferenceBase* exists, it shall be used in all references to that particular package (cf. rule ARPac_73 of the ARPackage guideline).

10 Reason: Avoid inconsistencies if the *ReferenceBase* path changes.

15 The following example illustrates establishing a *ReferenceBase* named "rba_CUCELCompuMethods" for an ARPackage with path /RB/RBA/Common/CentralElements/CompuMethods within an ARPackage Adc.

20 This *ReferenceBase* is copied from rba_CUCEL_ReferenceBases_Blueprint.arxml into the respective ARPackage Adc.

25 Figure 91 Example *ReferenceBase*

```
...
<AR-PACKAGE>
  <SHORT-NAME>Adc</SHORT-NAME>
  <REFERENCE-BASES>
    <REFERENCE-BASE>
      <SHORT-LABEL>rba_CUCELCompuMethods</SHORT-LABEL>
      <IS-DEFAULT>false</IS-DEFAULT>
      <IS-GLOBAL>false</IS-GLOBAL>
      <BASE-IS-THIS-PACKAGE>false</BASE-IS-THIS-PACKAGE>
      <PACKAGE-REF DEST="AR-PACKAGE">/RB/RBA/Common/CentralElements/CompuMethods</PACKAGE-REF>
    </REFERENCE-BASE>
  </REFERENCE-BASES>
  ...
</AR-PACKAGE>
...
```

30 Henceforth, the element with fully qualified path /RB/RBA/Common/CentralElements/CompuMethods/**Identical** can be referenced via /rba_CUCELCompuMethods/**Identical** within all sub-packages of ARPackage Adc.

35 Example:

```
<COMPU-METHOD-REF DEST="COMPU-METHOD" BASE="rba_CUCELCompuMethods">Identical</COMPU-METHOD-REF>
```

40 The following table lists all the centrally defined *ReferenceBases*.

45 Table 67 List of *ReferenceBases*

ReferenceBase	Full Path
Platform_ImplementationDataTypes	/AUTOSAR_Platform/ImplementationDataTypes
Platform_CompuMethods	/AUTOSAR_Platform/CompuMethods
Platform_SwBaseTypes	/AUTOSAR_Platform/SwBaseTypes
Std_ImplementationDataTypes	/AUTOSAR_Std/ImplementationDataTypes
Std_CompuMethods	/AUTOSAR_Std/CompuMethods
Comtype_ImplementationDataTypes	/AUTOSAR_Comtype/ImplementationDataTypes
Comtype_CompuMethods	/AUTOSAR_Comtype/CompuMethods
RBStd_ImplementationDataTypes	/RB/Common/CentralElements/ImplementationDataTypes
RB{domain}Std_ImplementationDataTypes	/RB/{domain}/Common/CentralElements/ImplementationDataTypes
AR_KeywordSets	/AUTOSAR_AISpecification/KeywordSets/{NameOfKeywordSet}
RB_KeywordSets	/RB/Common/NamingConventions/KeywordSets/{NameOfKeywordSet}
rba_CUCELSwAddrMethods	/RB/RBA/Common/CentralElements/SwAddrMethods
rba_CUCELCompuMethods	/RB/RBA/Common/CentralElements/CompuMethods

60 Note that currently no *ReferenceBases* are implemented within CUCEL.

65 Rule CEL_013: Package Path for AUTOSAR SwBaseTypes

70 **Instruction** The *SwBaseTypes* shall be implemented in the ARPackage /AUTOSAR_Platform/SwBaseTypes.

In the package CUCEL the following *SwBaseTypes* are provided. The following table lists all the *SwBaseTypes* defined by AUTOSAR.

Table 68 List of *SwBaseTypes*

Type	Description
boolean	boolean unsigned integer
uint8	8 bit unsigned integer
sint8	8 bit signed integer
uint16	16 bit unsigned integer
sint16	16 bit signed integer
uint32	32 bit unsigned integer
sint32	32 bit signed integer
uint64	64 bit unsigned integer
sint64	64 bit signed integer
float32	Single precision (32 bit) floating point number (maybe deactivated if µC has no floating point unit)
float64	Double precision (64 bit) floating point number (maybe deactivated if µC has no floating point unit)
uint8_least	At least 8 bit unsigned integer
sint8_least	At least 8 bit signed integer
uint16_least	At least 16 bit unsigned integer
sint16_least	At least 16 bit signed integer
uint32_least	At least 32 bit unsigned integer
sint32_least	At least 32 bit signed integer
void	void (no result provided to caller, but normal function termination)

For more details refer to the delivery notes of package CUCEL, where *SwBaseTypes* are provided by the SW module "Platform".

Rule CEL_012: Package Path for AUTOSAR StandardTypes

Instruction The StandardTypes shall be implemented as *ImplementationDataTypes* of the top-level ARPackage AUTOSAR_Std.

In the package CUCEL the following AUTOSAR StandardTypes are provided which are specified in *Document "Standard Types"* [TR_Std].

Table 69 List of AUTOSAR StandardTypes

Type	Description
Std_ReturnType	Type of a standard return type. Std_ReturnType is based on data type uint8.
Std_VersionInfoType	Type to request the version of a BSW module by using the module-specific GetVersionInfo() function. Std_VersionInfoType is a structure containing the following elements: <ul style="list-style-type: none">▶ uint16 vendorID▶ uint16 moduleID▶ uint8 sw_major_version▶ uint8 sw_minor_version▶ uint8 sw_patch_version

For more details refer to the delivery notes of package CUCEL, where AUTOSAR StandardTypes are provided by the SW module "Standard".

Table 71 List of AUTOSAR ComStackTypes

Type	Description
<i>PduIdType</i>	Used to define variables for unique identifiers for PDUs
<i>PduLengthType</i>	Used to define length information of a PDU
<i>PduInfoType</i>	Used to store the basic information about a PDU
<i>TpDataState</i>	Used to define variables to store the state of a Transport Protocol (TP)
<i>NotifResultType</i>	Used to define variables to store the result status of a notification (confirmation or indication)
<i>TpParameterType</i>	Used to specify the type of parameter of a Transport Protocol (TP)
<i>BufReq_ReturnType</i>	Used to store the result of a buffer request
<i>BusTrcvErrorType</i>	Used to return the bus status evaluated by a transceiver
<i>TpDataStateType</i>	Used to store the state of Transport Protocol buffer
<i>RetryInfoType</i>	Used to store the information about Transport Protocol buffer handling
<i>NetworkHandleType</i>	Used to store the identifier of a communication channel
<i>PNCHandleType</i>	Used to store the identifier of a partial network cluster

For more details refer to the delivery notes of package CUCEL where the AUTOSAR ComStackTypes are provided by the SW module "ComStack".

Rule CEL_016: Package Path for RB StandardTypes

Instruction The RB StandardTypes shall be implemented as *ImplementationDataTypes* of either ARPackage /RB/Common/CentralElements or /RB/{domain}/Common/CentralElements.

Currently no RB StandardTypes are specified within central elements deliveries.

Rule CEL_113: Package Path for CUBAS CompuMethods

Instruction Centrally defined *CompuMethods* shall be implemented in ARPackage /RB/RBA/Common/CentralElements/CompuMethods.

In the package CUCEL the following *CompuMethods* are provided.

Table 72 List of CUBAS CompuMethods

Type	Description
<i>Identical</i>	The internal value and the physical one are identical. No physical unit is available currently but it will be added as soon as it is made available by AUTOSAR. This <i>CompuMethod</i> shall only be used within CUBAS and not in ASW until the unit is available. For more details refer to the delivery notes of package CUCEL. The computation method is provided by the SW module "rba_CUCELSwCompuMethods".

In future more computation methods will be defined.

Rule CEL_114: Package Path for CUBAS SwAddrMethods

Instruction Centrally defined *SwAddrMethods* shall be implemented in ARPackage /RB/RBA/Common/CentralElements/SwAddrMethods.

In the package CUCEL the following *SwAddrMethods* are provided.

Table 73 List of AUTOSAR SwAddrMethods (and RB extensions)

Type	Description
INTERNAL_VAR_INIT	measurement points that are initialized with values after every reset
INTERNAL_VAR_CLEARED	measurement points that are cleared to zero after every reset
INTERNAL_VAR_POWER_ON_CLEARED	measurement points that are cleared to zero only after power on reset
CALIB	calibration parameters
CALIB_FAST	calibration parameters that are frequently used (this addressing method is an extension to AUTOSAR).
CALIB_SLOW	calibration parameters that are rarely used (this addressing method is an extension to AUTOSAR).
CODE	code (application code, BSW code, boot code)
CONFIG_DATA	constants with attributes that show that they reside in one segment for link-time module configuration
CONFIG_DATA_FAST	frequently used constants with attributes that show that they reside in one segment for link-time module configuration (This addressing method is an extension to AUTOSAR.)
CONFIG_DATA_SLOW	rarely used constants with attributes that show that they reside in one segment for link-time module configuration (This addressing method is an extension to AUTOSAR.)
CONST	global or static constants
CONST_FAST	global or static constants that are frequently used (This addressing method is an extension to AUTOSAR.)
CONST_SLOW	global or static constants that are rarely used (This addressing method is an extension to AUTOSAR.)
VAR_INIT	global or static variables that are initialized with values after every reset
VAR_CLEARED	global or static variables that are cleared to zero after every reset
VAR_POWER_ON_CLEARED	global or static variables that are cleared to zero only after power on reset or reset after trap
VAR_FAST_INIT	global or static variables that are frequently used and initialized with values after every reset
VAR_FAST_CLEARED	global or static variables that are frequently used and cleared to zero after every reset
VAR_FAST_POWER_ON_CLEARED	global or static variables that are frequently used and cleared to zero only after power on reset or reset after trap
VAR_SLOW_INIT	global or static variables that are rarely used and are initialized with values after every reset (This addressing method is an extension to AUTOSAR.)
VAR_SLOW_CLEARED	global or static variables that are rarely used and are cleared to zero after every reset (This addressing method is an extension to AUTOSAR.)
VAR_SLOW_POWER_ON_CLEARED	global or static variables that are rarely used and are cleared to zero only after power on reset or reset after trap (This addressing method is an extension to AUTOSAR.)

For more details refer to the delivery notes of package CUCEL, where *SwAddrMethods* are provided by the SW module "rba_CUCEL*SwAddrMethods*".

Rule CEL_131: Selecting "_FAST", "_SLOW" *SwAddrMethods*

Instruction *SwAddrMethods* with "_SLOW" in the name shall be used **for variables / parameters accessed by runnables slower than 20 ms** and other rarely called functions.

SwAddrMethods with "_FAST" in the name shall be used **for variables / parameters accessed by runnables faster than 5 ms and frequently used interrupts**.

In other cases *SwAddrMethods* without "_FAST" / "_SLOW" shall be used.

Rule CEL_115: Package Path for AUTOSAR KeywordSets

Instruction The ARPackagePath for AUTOSAR *KeywordSets* shall be of the form:

/AUTOSAR_AISpecification/KeywordSets/{NameOfKeywordSet} derived from /AUTOSAR/AISpecification/KeywordSets_Blueprint/KeywordList

The ARPackagePath for BBM *KeywordSets* shall be of the form:

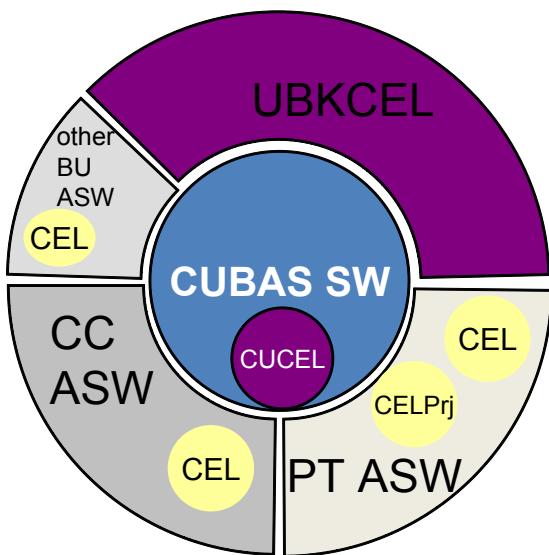
/RB/Common/NamingConventions/KeywordSets/{NameOfKeywordSet}

Currently no AUTOSAR or BBM *KeywordSets* are specified within central elements deliveries.

11.3 Delivery Aspects

CEL is the acronym for the term **Central E**lements which is used in the context of delivery units.

Figure 92 Central Packages/Delivery Units of BBM Common Software



Note: name of "UBKCEL" will be renamed somehow.

Rule CEL_018: Delivery Units

Instruction In order to refer to unique names within this guideline, a general definition of the delivery unit names was done here. Furthermore these names can be mapped to SCM-specific implementations.

The delivery unit for CUBAS central elements shall be CUCEL. CUCEL shall provide 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 AISpecification and for BBM (was UBK) shall be (tbd; was UBKCEL; will be renamed somehow).

The delivery unit for business unit specific central elements shall be the business unit specific CELs.

CUBAS software contains the delivery unit "CUCEL", which contains the central elements. This CUBAS specific central elements delivery unit contains the required central elements to develop and deliver self-contained CUBAS software. CUBAS software shall not depend on UBKCEL. UBKCEL contains the central elements for ASW of AUTOSAR WP 10.x and BBM . Hence CDG provides all BBM common central elements and AUTOSAR WP 10.x to the BBM customers. The scope of this guideline is not to define where a central element shall be located. This has to be decided by another instance (the so-called clearing board). The focus of this guideline is to establish rules for defining and handling central elements.

Rule CEL_019: Additional Central Elements in Business Unit

Instruction Business units may define additional delivery units for central elements.

Rule CEL_027: Backward Compatibility of CUCEL and other central elements deliveries

Instruction CUCEL and other central elements deliveries shall always be backward compatible.

05

12 Rule Set: CLF (Classification File)

10
This chapter is relevant for BSW modules which are developed with the CDG development environment using the CDG build.15

12.1 Introduction

20
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.25
Since several years the CDG tool environment follows the concept to adapt file and folder information. So it is possible to classify files and folders for specific Use Cases (Processors, Autosar, MSR, Documentation, etc.). These additional information 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.30
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.35

12.1.1 CLF projects overview

40
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.45
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.50
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).55
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.60
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".65

12.1.2 Tool Environment

70
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*.75
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.80
More details about the UI elements can be read in the specific parts of *Autosar Workbench* help.

05

12.1.3 References

The following documents have to be considered together with this document.

10 For CLF there is a reference available in the Intranet:

15 ▶ **CLF Documentation**

The CLF documentation can be found in the Inside-Wiki of CDG-SMT *Document "CLF Documentation" [CLFDocu]*

20

12.2 Guidelines

This chapter defines the guidelines for editing and using CLF

25

12.2.1 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.

30 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:

```
35 package Mcal
{
    class = PAC

    package-includes = Adc_Stc,
        Dio,
        Port
        rba_Gtm
        // more components separated by comma
}
```

45 CLF Name: Dio.clf:

```
package Dio
{
    class = COMP
}
```

50 Examples: ECU_CFG for PAC Mcal and COMP Dio

55 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
}
```

65 CLF Name: Dio_Cfg.clf:

```
05 package Dio_Cfg
{
    class = ECU_CFG
}
```

10 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

```
15 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
    }

    // ...
}
```

35 Hint It is recommended to name the top level CLF file identical to the project name.

40 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

45

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).

50 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).

45 Rule files shall reside in the top level folder of the project, aside MyProject.clf

55 The standard Rules set is provided by CDG-SMT/EMT, it can be found in the Inside.Wiki of CDG-SMT: [\[Document CLF Documentation / URL: https://inside-wiki.bosch.com/confluence/display/CDGSMT/CLF+documentation\]](#)

60 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
```

65

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 recommended to use only the top level variants instead of providing 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.

05 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.

10 PackageTwo.clf with project global variants:

```
package PackageTwo
{
    class = PAC

15    variant PROJECT_VARIANT_ONE
    {
        // variant specific includes
        package-includes = ComponentOne(PROJECT_VARIANT_ONE),
                           ComponentTwo(PROJECT_VARIANT_ONE)
    }

20    variant PROJECT_VARIANT_TWO
    {
        // variant specific includes
        package-includes = ComponentOne(PROJECT_VARIANT_TWO),
                           ComponentTwo(PROJECT_VARIANT_TWO)
    }

25    // ...
}
```

30 PackageFive.clf with package local variants:

```
package PackageFive
{
    class = PAC

    package-includes = ComponentSeven

    variant PACKAGEFIVE_VARIANT_A
    {
        // variant specific includes
        package-includes = ComponentEight(PACKAGEFIVE_VARIANT_A),
                           ComponentNine(PACKAGEFIVE_VARIANT_A)
    }

40    variant PACKAGEFIVE_VARIANT_B
    {
        // variant specific includes
        package-includes = ComponentEight(PACKAGEFIVE_VARIANT_B),
                           ComponentNine(PACKAGEFIVE_VARIANT_B)
    }

45    // ...
}
```

55

12.2.2 Naming Conventions

60 The following rules define naming conventions for CLF files and their structure.

Rule CLF_Naming_001: Domain

65 Instruction

Class: NamingConvention

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

Class: NamingConvention

The CLF file name and the containing package name shall be identical. CLF files and package names shall be unique across the whole project

Rule CLF_Naming_003: Packages and components

Instruction

Class: NamingConvention

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

Class: NamingConvention

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

Class: NamingConvention

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_CEG

Rule CLF_Naming_006: Structural components configuration

Instruction

Class: NamingConvention

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

Class: NamingConvention

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>

05 This helps to differentiate variant names from CLF package names which are usually written in CamelCase.

10 Example: IFX_DEV3_LQFP

15

12.3 Examples for Variants Usage

20 CLF variant handling can be used to enable or disable individual files or complete folders.

25

12.3.1 Simple enabling / disabling of folders

30 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.

35 Example: COMP rba_IoExtCj135

40 CLF Name: rba_IoExtCj135.clf

45

```
package rba_IoExtCj135
{
    class = COMP
    variant USE {}

    variant IGNORE {
        ignore-rule {
            path = **
        }
    }
}
```

50 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:

55

```
package IoExtDev
{
    class = PAC

    // use either USE or IGNORE as variant
    package-includes = rba_IoExtCj135(USE),
                        rba_IoExtCj721(IGNORE)
}
```

60

12.3.2 Using configuration variants

65 Usually there are many configuration variants, e.g. for different CPU devices, ECUs, customer projects, etc.

70 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.

75 Either the complete variant is passed to the next CLF package, e.g. IFX_DEV3_LQFP or JDP_DEV3_LQFP.

80 Or only the manufacturer type IFX or JDP, if there is no dependency to the individual microcontroller device.

85 Or no variant is passed at all if the component is independent of any hardware variant.

90 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
    }
}

```

05 CLF for configuration of BootCtrl: Conf\BootCtrl\BootCtrl_Cfg.clf

```

package BootCtrl_Cfg
{
    class = ECU_CFG

10    variant IFX_DEV3_LQFP
    {
        package-includes = rba_BootCtrl_Cfg(IFX_DEV3_LQFP)
    }

15    variant JDP_DEV3_LQFP
    {
        package-includes = rba_BootCtrl_Cfg(JDP_DEV3_LQFP)
    }

20}

```

CLF for configuration of rba_BootCtrl: Conf\BootCtrl\rba_BootCtrl\rba_BootCtrl_Cfg.clf

```

25 package rba_BootCtrl_Cfg
{
    class = ECU_CFG

    // rules for common folder
    package-includes = Project_Rules_Conf_Common

30    // 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
    }
}

```

45 CLF for configuration of BSMM: Conf\BSMM\BSMM_Cfg.clf

```

package BSMM_Cfg
{
    class = ECU_CFG

50    package-includes = BswM_Cfg
}

```

55 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

60    // rules for common folder
    package-includes = Project_Rules_Conf_Common

65}

```

05 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)
    }
}
```

30 Now to the included rule packages.

35 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.

40 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).

45 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

```
45 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
    }
}
```

65 Project_Rules_Conf_IFX_Common.clf

```
65 package Project_Rules_Conf_IFX_Common
{
```

```

05     rule {
10         class = CONFDATA
15         path = **/IFX_Common/*.arxml
20         path = **/IFX_Common/*_confdata.xml
     }

25     rule {
30         class = SWHDR
35         path = **/IFX_Common/**/*.h
     }

40 }

```

Project_Rules_Conf_JFP_Common.clf

```

package Project_Rules_Conf_JDP_Common
{
25     rule {
30         class = CONFDATA
35         path = **/JDP_Common/*.arxml
40         path = **/JDP_Common/*_confdata.xml
     }

45     rule {
50         class = SWHDR
55         path = **/JDP_Common/**/*.h
     }

60 }

```

Finally, as example for variant specific, Project_Rules_Conf_JFP_Common.clf

```

package Project_Rules_Conf_IFX_DEV3_LQFP
{
45     rule {
50         class = CONFDATA
55         path = **/IFX_DEV3_LQFP/*.arxml
60         path = **/IFX_DEV3_LQFP/*_confdata.xml
     }

65     rule {
70         class = SWHDR
75         path = **/IFX_DEV3_LQFP/*.h
     }

80     rule {
85         class = SWSRC
90         path = **/IFX_DEV3_LQFP/*.c
     }
}

```

So the rule of thumb is: Project_Rules_Conf_<VariantName>.clf

```

package Project_Rules_Conf_<VariantName>
{
65     rule {
70         class = CONFDATA
}

```

```

05      path = **/<VariantName>/*.arxml
       path = **/<VariantName>/*_confdata.xml
     }

10    rule {
       class = SWHDR
       path = **/<VariantName>/*.h
     }

15    rule {
       class = SWSRC
       path = **/<VariantName>/*.c
     }
   }

```

If necessary, other file classes can be added.

20

25

30

35

40

45

50

55

60

65

70

A Rules summary

Table 74 General Language Topics

Id	Rule Chapter 3.1.1
[CCode_001]	<p>Conformity to C Standard</p> <p>DerivedFrom: MISRA C:2012 Rule 1.1</p> <p>DerivedFrom: MISRA C:2012 Rule 1.2</p> <p>The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits. Language extensions should not be used.</p>
[CCode_002]	<p>Character Set, Escape Sequences and Trigraphs</p> <p>DerivedFrom: MISRA C:2012 Rule 4.2</p> <p>A basic set of characters and escape sequences conform to ISO C90 shall be used. Usage of trigraphs is not allowed.</p>
[CCode_003]	<p>Octal Constants and Escape Sequences</p> <p>DerivedFrom: MISRA C:2012 Rule 4.1</p> <p>DerivedFrom: MISRA C:2012 Rule 7.1</p> <p>Octal constants (other than zero) and octal escape sequences shall not be used. Hexadecimal escape sequences shall be terminated.</p>
[CCode_004]	<p>Usage and Handling of Bit Fields</p> <p>DerivedFrom: MISRA C:2012 Rule 6.1</p> <p>DerivedFrom: MISRA C:2012 Rule 6.2</p> <p>Bit-fields shall only be declared with an appropriate type ("unsigned int" or "signed int"). 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 and no certain order of the bits inside the bit field shall be assumed.</p> <p>Single-bit named bit fields shall not be of a signed type.</p>
[CCode_005]	<p>Documentation of Implementation-defined Behaviour</p> <p>DerivedFrom: MISRA C:2012 Dir 1.1</p> <p>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.</p>
[CCode_006]	<p>File Handles are not Allowed</p> <p>DerivedFrom: MISRA C:2012 Rule 22.3</p> <p>DerivedFrom: MISRA C:2012 Rule 22.4</p> <p>DerivedFrom: MISRA C:2012 Rule 22.5</p> <p>DerivedFrom: MISRA C:2012 Rule 22.6</p> <p>File handles and file streams shall not be used.</p>
[CCode_007]	<p>Check of Validity of Values Passed to Functions</p> <p>DerivedFrom: MISRA C:2012 Dir 4.11</p> <p>The validity of values passed to functions shall be checked.</p>

Table 75 MISRA and HIS Metrics Conformity

Id	Rule Chapter 3.1.2
[CCode_Misra-HIS_001]	<p>Conformity to MISRA Standard and HIS Metrics</p> <p>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.</p>
[CCode_Misra-HIS_002]	<p>Commenting MISRA Violations in C Code</p> <p>In technically reasonable, exceptional cases MISRA violations are permissible and shall be documented with a comment within the source file.</p>

Id	Rule <i>Chapter 3.1.2</i>
[CCode_Misra-HIS_003]	Commenting MISRA Violations for a Complete Module 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.
[CCode_Misra-HIS_004]	Commenting HIS Metric Violations in C Code In technically reasonable, exceptional cases HIS Metrics limit violations are permissible and shall be documented with a comment within the source file.

15 Table 76 Initializations

Id	Rule <i>Chapter 3.1.3</i>
[CCode_Inits_001]	Initialization of Variables DerivedFrom: MISRA C:2012 Rule 9.1 All variables shall not be read before they have been set.
[CCode_Inits_002]	Initialization of Enumerators DerivedFrom: MISRA C:2012 Rule 8.12 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.
[CCode_Inits_003]	Initializer Lists without Side Effects Scope: C99 based code DerivedFrom: MISRA C:2012 Rule 13.1 Initializer lists shall not contain persistent side effects.
[CCode_Inits_004]	Singular Initializations of Objects Scope: C99 based code DerivedFrom: MISRA C:2012 Rule 9.4 An element of an object shall not be initialized more than once.
[CCode_Inits_005]	Usage of Designated Initializers for Arrays Scope: DerivedFrom: MISRA C:2012 Rule 9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.

45 Table 77 Expressions

Id	Rule <i>Chapter 3.1.4</i>
[CCode_Expr_-001]	Precedence of Operators within Expressions DerivedFrom: MISRA C:2012 Rule 12.1 The precedence of operators within expressions should be made explicit.
[CCode_Expr_-002]	Value of Expressions DerivedFrom: MISRA C:2012 Rule 13.2 The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders
[CCode_Expr_-003]	Usage of Sizeof Operator DerivedFrom: MISRA C:2012 Rule 13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects.
[CCode_Expr_-004]	Logical and Conditional Operators without Side Effects DerivedFrom: MISRA C:2012 Rule 13.5 The right hand operand of a logical '&&' or ' ' operator and of a conditional operator '? :' shall not contain persistent side effects.

Id	Rule Chapter 3.1.4
[CCode_Expr_-005]	<p>Logical Operator Operands as Primary Expressions</p> <p>DerivedFrom: MISRA C:2012 Rule 12.1</p> <p>The operands of a logical '&&' or ' ' operators shall be primary-expressions or extra parentheses are recommended.</p>
[CCode_Expr_-006]	<p>Usage of Logical Operators</p> <p>DerivedFrom: MISRA C:2012 Rule 10.1</p> <p>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 '!='. </p>
[CCode_Expr_-007]	<p>Type of Bitwise Operators</p> <p>DerivedFrom: MISRA C:2012 Rule 10.1</p> <p>Bitwise operators shall not be applied to operands whose type is a signed integer.</p>
[CCode_Expr_-008]	<p>Usage of Shift Operator</p> <p>DerivedFrom: MISRA C:2012 Rule 12.2</p> <p>The right hand operand of a shift operator shall lie in the range: "zero" to "width in bits of essential type minus one" of the left hand operand.</p>
[CCode_Expr_-009]	<p>Usage of Unary Minus Operator</p> <p>DerivedFrom: MISRA C:2012 Rule 10.1</p> <p>The unary minus operator shall not be applied to an expression whose type is an unsigned integer.</p>
[CCode_Expr_-010]	<p>Usage of Comma Operator</p> <p>DerivedFrom: MISRA C:2012 Rule 12.3</p> <p>The comma operator shall not be used, except in the control expression of a "for" loop and within macros.</p>
[CCode_Expr_-011]	<p>Usage of Constant Unsigned Integer Expression</p> <p>DerivedFrom: MISRA C:2012 Rule 12.4</p> <p>Evaluation of constant expressions should not lead to unsigned integer wrap-around</p>
[CCode_Expr_-012]	<p>Usage of Increment and Decrement Operator</p> <p>DerivedFrom: MISRA C:2012 Rule 13.3</p> <p>A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator.</p>
[CCode_Expr_-013]	<p>Value of Floating Complex Expression</p> <p>Status: removed</p> <p>ReplacedBy: [CCode_Essential_008]</p>
[CCode_Expr_-014]	<p>Implicit Integer Type Conversions</p> <p>Implicit type conversions without a cast of integer types shall only be done from smaller to bigger types where no information is lost.</p>
[CCode_Expr_-015]	<p>Explicit Integer Type Casts</p> <p>Explicit type casts of integer types shall only be done if an explicit conversion to a narrower type is intended.</p>

Table 78 Control Statement Expressions

Id	Rule Chapter 3.1.5
[CCode_Control_-001]	<p>Boolean Expressions</p> <p>DerivedFrom: MISRA C:2012 Rule 13.4</p> <p>The result of an assignment operator should not be used.</p>

Id	Rule Chapter 3.1.5
[CCode_Control_002]	<p>Value Test Expression</p> <p>DerivedFrom: MISRA C:2012 Rule 14.4</p> <p>The controlling expression of an if-statement and the controlling expression of an iteration–statement shall have essentially Boolean type.</p>
[CCode_Control_003]	<p>Test of Floating-Point Expressions</p> <p>Floating-point expressions shall not be tested for equality or inequality. A check for equality or inequality to zero is allowed.</p>
[CCode_Control_004]	<p>For Loop Expression</p> <p>DerivedFrom: MISRA C:2012 Rule 14.2</p> <p>A for loop shall be well-formed: The three expressions of a for-statement shall be concerned only with loop control.</p>
[CCode_Control_005]	<p>For Loop Iteration</p> <p>DerivedFrom: MISRA C:2012 Rule 14.2</p> <p>A for loop shall be well-formed: Numeric variables being used within a for-loop for iteration counting shall not be modified in the body of the loop.</p>
[CCode_Control_006]	<p>Non Floating Loop Counters</p> <p>DerivedFrom: MISRA C:2012 Rule 14.1</p> <p>A loop counter shall not have essentially floating type.</p>
[CCode_Control_007]	<p>Non Invariant Controlling Expressions</p> <p>DerivedFrom: MISRA C:2012 Rule 14.3</p> <p>Controlling expressions shall not be invariant.</p>

Table 79 Control Flow

Id	Rule Chapter 3.1.6
[CCode_Cntr-Flow_001]	<p>Unreachable Code</p> <p>DerivedFrom: MISRA C:2012 Rule 2.1</p> <p>A project shall not contain unreachable code.</p>
[CCode_Cntr-Flow_002]	<p>Duplication of Code</p> <p>The duplication of code should be avoided.</p>
[CCode_Cntr-Flow_003]	<p>Non-Null Statements</p> <p>DerivedFrom: MISRA C:2012 Rule 2.2</p> <p>All non-null statements shall either a) have at least one side-effect however executed, or b) cause control flow to change.</p>
[CCode_Cntr-Flow_004]	<p>Null Statement</p> <p>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.</p>
[CCode_Cntr-Flow_005]	<p>Goto Statement</p> <p>DerivedFrom: MISRA C:2012 Rule 2.6</p> <p>DerivedFrom: MISRA C:2012 Rule 15.1</p> <p>The 'goto' statement shall not be used. Additionally goto labels shall not be defined.</p>
[CCode_Cntr-Flow_006]	<p>Iteration Statement</p> <p>DerivedFrom: MISRA C:2012 Rule 15.4</p> <p>There should be no more than one break statement used to terminate any iteration statement.</p>
[CCode_Cntr-Flow_007]	<p>If ... Else</p> <p>DerivedFrom: MISRA C:2012 Rule 15.7</p> <p>All if ... else if constructs shall be terminated with an else statement.</p>

Table 80 Switch Statements

Id	Rule Chapter 3.1.7
[CCode_Switch_-001]	<p>Prohibition of Declarations or Definitions between Case and Default Clauses</p> <p>DerivedFrom: MISRA C:2012 Rule 16.1</p> <p>The case and default clauses in the body of a switch statement shall not be preceded by declarations or definitions.</p>
[CCode_Switch_-002]	<p>Position of Switch Labels</p> <p>DerivedFrom: MISRA C:2012 Rule 16.2</p> <p>A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.</p>
[CCode_Switch_-003]	<p>Usage of Break Statement</p> <p>DerivedFrom: MISRA C:2012 Rule 16.3</p> <p>An unconditional break statement shall terminate every switch-clause even if it is not used because of optimized programming.</p>
[CCode_Switch_-004]	<p>Usage of Default Label</p> <p>DerivedFrom: MISRA C:2012 Rule 16.4</p> <p>DerivedFrom: MISRA C:2012 Rule 16.5</p> <p>Every switch statement shall have a default label.</p> <p>The default label shall appear as either the first or the last switch label of a switch statement.</p>
[CCode_Switch_-005]	<p>Condition for Switch Expressions</p> <p>DerivedFrom: MISRA C:2012 Rule 16.7</p> <p>A switch-expression shall not have essentially Boolean type.</p>
[CCode_Switch_-006]	<p>Usage of Switch Statements</p> <p>DerivedFrom: MISRA C:2012 Rule 16.6</p> <p>Every switch statement shall have at least two switch-clauses.</p>
[CCode_Switch_-007]	<p>Conformance of Switch Statements</p> <p>DerivedFrom: MISRA C:2012 Rule 16.1</p> <p>All switch statements shall be well-formed.</p>

Table 81 Structures and Unions

Id	Rule Chapter 3.1.8
[CCode_Struct_-001]	<p>Usage of Unions</p> <p>DerivedFrom: MISRA C:2012 Rule 19.1</p> <p>DerivedFrom: MISRA C:2012 Rule 19.2</p> <p>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.</p>
[CCode_Struct_-002]	<p>Typedef Declarations for Structures and Unions</p> <p>All structure and unions should base on specific typedef declarations.</p>
[CCode_Struct_-003]	<p>Completion of Structure and Union Types</p> <p>DerivedFrom: MISRA C:2012 Rule 1.3</p> <p>All structure and union types shall be complete at the end of a translation unit.</p>
[CCode_Struct_-004]	<p>Avoidance of Alignment Gaps inside Structures</p> <p>Elements of structures shall be arranged in that way to avoid alignment gaps.</p>

Table 82 Preprocessing Directives

Id	Rule Chapter 3.1.9
[CCode_Propo_001]	Handling of #include Statements DerivedFrom: MISRA C:2012 Rule 20.1 #include directives should only be preceded by preprocessor directives or comments.
[CCode_Propo_002]	Prohibition of Non-Standard Characters in #include Directives Class: NamingConvention DerivedFrom: MISRA C:2012 Rule 20.2 The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.
[CCode_Propo_003]	#include followed by a File Name DerivedFrom: MISRA C:2012 Rule 20.3 The #include directive shall be followed by either a <filename> or "filename" sequence.
[CCode_Propo_004]	Prohibition of #undef DerivedFrom: MISRA C:2012 Rule 20.5 #define should not be used.
[CCode_Propo_005]	Handling of Preprocessor Operator "defined" DerivedFrom: MISRA C:2012 Rule 1.3 The "defined" preprocessor operator shall only be used in one of the two standard forms.
[CCode_Propo_006]	Correctness of Preprocessing Directives DerivedFrom: MISRA C:2012 Rule 20.13 A line whose first token is # shall be a valid preprocessing directive.
[CCode_Propo_007]	Hold Preprocessor Directives together DerivedFrom: MISRA C:2012 Rule 20.14 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
[CCode_Propo_008]	Defined Identifiers in control Expressions DerivedFrom: MISRA C:2012 Rule 20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.
[CCode_Propo_009]	Evaluation of #if and #elif Preprocessing Directives DerivedFrom: MISRA C:2012 Rule 20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.

Table 82 Preprocessing Directives

Table 83 Macros

Id	Rule Chapter 3.1.10
[CCode_Macro_001]	Handling of C Macros 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.
[CCode_Macro_002]	#undef of C Macros Status: removed
[CCode_Macro_003]	Handling of Function-like Macros DerivedFrom: MISRA C:2012 Rule 20.7 Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses unless they are used as operands of # or ##.
[CCode_Macro_004]	Calling of Function-like Macros DerivedFrom: MISRA C:2012 Rule 1.3 A function-like macro shall not be invoked without all of its arguments.

Id	Rule Chapter 3.1.10
[CCode_Macro_005]	<p>Arguments of Function-like Macros</p> <p>DerivedFrom: MISRA C:2012 Rule 20.6</p> <p>Tokens that look like a preprocessing directive shall not occur within a macro argument.</p>
[CCode_Macro_006]	<p>Macros are no Replacements for Keywords</p> <p>DerivedFrom: MISRA C:2012 Rule 20.4</p> <p>A macro shall not be defined with the same name as a keyword.</p>
[CCode_Macro_007]	<p>Handling of # and ## Operators within Macros</p> <p>DerivedFrom: MISRA C:2012 Rule 20.11</p> <p>A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.</p>
[CCode_Macro_008]	<p>Handling of Macro Parameters used as Operands to # or ## Operators</p> <p>DerivedFrom: MISRA C:2012 Rule 20.12</p> <p>A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators.</p>

Table 84 Essential Type Model

Id	Rule Chapter 3.1.11
[CCode_Essential_001]	<p>Operands Conformity to Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.1</p> <p>Operands shall not be of an inappropriate essential type.</p>
[CCode_Essential_002]	<p>Expression of Essentially Character Types</p> <p>DerivedFrom: MISRA C:2012 Rule 10.2</p> <p>Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.</p>
[CCode_Essential_003]	<p>Possible Value Assignments</p> <p>DerivedFrom: MISRA C:2012 Rule 10.3</p> <p>The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.</p>
[CCode_Essential_004]	<p>Operands of Operators and Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.4</p> <p>Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.</p>
[CCode_Essential_005]	<p>Value of Expressions and Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.5</p> <p>The value of an expression should not be cast to an inappropriate essential type.</p>
[CCode_Essential_006]	<p>Composite Expression and Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.6</p> <p>The value of a composite expression shall not be assigned to an object with wider essential type.</p>
[CCode_Essential_007]	<p>Composite Expression as Operand and Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.7</p> <p>If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type.</p>
[CCode_Essential_008]	<p>Composite Expression as Operand and Essential Type</p> <p>DerivedFrom: MISRA C:2012 Rule 10.8</p> <p>The value of a composite expression shall not be cast to a different essential type category or a wider essential type.</p>

05 Table 85 Main Rule

Id	Rule <i>Chapter 3.2.1</i>
[Abstr_Main_001]	<p>Main Rule of Compiler Abstraction</p> <p>DerivedFrom: MISRA C:2012 Rule 1.3</p> <p>The source code of software modules (at least above the µC Abstraction Layer (MCAL)) shall be neither processor-dependent nor compiler-dependent. No reliance shall be placed on undefined or unspecified behaviour of the compiler.</p>

15 Table 86 Prohibition of Language and Compiler Specifics

Id	Rule <i>Chapter 3.2.2</i>
[Abstr_Main_002]	<p>Prohibition of Compiler Specific Headers and Libraries</p> <p>DerivedFrom: MISRA C:2012 Dir 1.1</p> <p>DerivedFrom: MISRA C:2012 Rule 1.3</p> <p>DerivedFrom: MISRA C:2012 Rule 21.4</p> <p>DerivedFrom: MISRA C:2012 Rule 21.5</p> <p>DerivedFrom: MISRA C:2012 Rule 21.6</p> <p>DerivedFrom: MISRA C:2012 Rule 21.7</p> <p>DerivedFrom: MISRA C:2012 Rule 21.8</p> <p>DerivedFrom: MISRA C:2012 Rule 21.9</p> <p>DerivedFrom: MISRA C:2012 Rule 21.10</p> <p>DerivedFrom: MISRA C:2012 Rule 21.11</p> <p>DerivedFrom: MISRA C:2012 Rule 21.12</p> <p>Compiler specific header files, libraries and intrinsic functions shall not be used.</p>
[Abstr_Main_003]	<p>Prohibition of Compiler Specific Keywords</p> <p>Compiler specific keywords and internal defines shall not be used.</p>
[Abstr_Main_004]	<p>Prohibition of Redefinition of Reserved Identifiers, Macros and Functions</p> <p>DerivedFrom: MISRA C:2012 Rule 21.1</p> <p>DerivedFrom: MISRA C:2012 Rule 21.2</p> <p>#define and #undef shall not be used on a reserved identifier or reserved macro name. A reserved identifier or macro name shall not be declared.</p>
[Abstr_Main_005]	<p>Prohibition of Restrict Type Qualifier</p> <p>Scope: C99 based code</p> <p>DerivedFrom: MISRA C:2012 Rule 8.14</p> <p>The restrict type qualifier shall not be used.</p>

50 Table 87 Abstraction of Addressing Keywords

Id	Rule <i>Chapter 3.2.3 [Abstr_AddrKeywords]</i>
[Abstr_NearFar_-001]	<p>Prohibition of Usage of Addressing Keywords</p> <p>Direct use of compiler and platform specific keywords for addressing of data and code like "_near" or "_far" is not allowed.</p>
[Abstr_NearFar_-002]	<p>AUTOSAR 16 bit Microcontroller Abstraction</p> <p>To encapsulate 16 bit microcontroller specifics the AUTOSAR standardized #defines shall be used.</p>
[Abstr_NearFar_-003]	<p>Usage of Module Name of near/far Addressing Keywords</p> <p>For declarations the module name of the defining module has to be used for the near/far addressing keywords.</p>

Table 88 Abstraction of Memory Mapping

Id	Rule <i>Chapter 3.2.4 [Abstr_MemMap]</i>
[Abstr_MemMap_-001]	Usage of Memory Mapping Concept instead of #pragma Keywords 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.
[Abstr_MemMap_-002]	Structure of Memory Mapping Concept 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 "{S-WC}_MemMap.h" for ASW components).
[Abstr_MemMap_-003]	Exception of Memory Mapping Concept: Function Local Variables 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.
[Abstr_MemMap_-004]	Exception of Memory Mapping Concept: Inline Functions For inline functions a memory mapping is not needed therefore memory mapping concept shall not be used.
[Abstr_MemMap_-005]	No Usage inside Function Bodies Memory mapping header files shall not be included inside the body of a function.
[Abstr_MemMap_-006]	Usage of Module Name of Memory Mapping Keywords For declarations the module name of the defining module has to be used for the memory mapping keywords.

Table 89 Abstraction of Inline Functions

Id	Rule <i>Chapter 3.2.5</i>
[Abstr_Inline_001]	Usage of LOCAL_INLINE DerivedFrom: MISRA C:2012 Rule 8.10 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.
[Abstr_Inline_002]	Inlines Without RTE APIs Within inline functions no RTE APIs shall be called.

Table 90 Handling of Assembler Instructions

Id	Rule <i>Chapter 3.2.6</i>
[Abstr_Asm_001]	Usage of Assembler Code DerivedFrom: MISRA C:2012 Dir 4.2 DerivedFrom: MISRA C:2012 Dir 4.3 Assembler instructions inside C code shall not be used unless they are encapsulated and isolated for special cases. All usage of assembly language should be documented, too.

Table 91 Prohibition of Dynamic Memory

Id	Rule <i>Chapter 3.2.7</i>
[Abstr_DynMem_-001]	Prohibition of Dynamic Memory Allocation DerivedFrom: MISRA C:2012 Dir 4.12 DerivedFrom: MISRA C:2012 Rule 21.3 DerivedFrom: MISRA C:2012 Rule 22.2 Dynamic memory allocation shall not be used.
[Abstr_DynMem_-002]	Usage of Memory Areas An area of memory shall not be reused for unrelated purposes.

Table 92 Pointer Type Conversions

Id	Rule Chapter 3.2.8
[Abstr_PtrConv_001]	Prohibited Function Pointer Conversions DerivedFrom: MISRA C:2012 Rule 11.1 Conversions shall not be performed between a pointer to a function and any other type.
[Abstr_PtrConv_002]	Prohibited Object Pointer Conversions DerivedFrom: MISRA C:2012 Rule 11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type.
[Abstr_PtrConv_003]	Prohibition of Casts between Pointer and Void and Arithmetic Types DerivedFrom: MISRA C:2012 Rule 11.6 A cast shall not be performed between pointer to void and an arithmetic type.
[Abstr_PtrConv_004]	Prohibition of Casts between Object Pointer Types DerivedFrom: MISRA C:2012 Rule 11.3 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.
[Abstr_PtrConv_005]	Preservation of Pointer Qualifications DerivedFrom: MISRA C:2012 Rule 11.8 A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.
[Abstr_PtrConv_006]	Preservation of Conversion between a Pointer to Object and Integer Type DerivedFrom: MISRA C:2012 Rule 11.4 A conversion should not be performed between a pointer to object and an integer type.
[Abstr_PtrConv_007]	Preservation of Conversion from Pointer to Void to Pointer to Object DerivedFrom: MISRA C:2012 Rule 11.5 A conversion should not be performed from pointer to void into pointer to object.
[Abstr_PtrConv_008]	Preservation of Pointer Qualifications DerivedFrom: MISRA C:2012 Rule 11.7 A cast shall not be performed between pointer to object and a non-integer arithmetic type.

Table 93 Pointer Arithmetic and Arrays

Id	Rule Chapter 3.2.9
[Abstr_PtrArith_001]	Allowed Form of Pointer Arithmetic DerivedFrom: MISRA C:2012 Rule 18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.
[Abstr_PtrArith_002]	Allowed Form of Pointer Subtraction DerivedFrom: MISRA C:2012 Rule 18.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array.
[Abstr_PtrArith_003]	Allowed Form of Comparison of Pointers DerivedFrom: MISRA C:2012 Rule 18.3 >, >=, <, <= shall not be applied to pointer types except where they point to the same array.
[Abstr_PtrArith_004]	Validity of Pointer Objects DerivedFrom: MISRA C:2012 Rule 18.6 The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist.
[Abstr_PtrArith_005]	Maximum Number of Pointer Indirection DerivedFrom: MISRA C:2012 Rule 18.5 Declarations should contain no more than two levels of pointer nesting.

Id	Rule Chapter 3.2.9
[Abstr_PtrArith_006]	<p>Prohibition of flexible array members</p> <p>DerivedFrom: MISRA C:2012 Rule 18.7</p> <p>Flexible array members shall not be declared.</p>
[Abstr_PtrArith_007]	<p>Prohibition of Variable-length Array Types</p> <p>DerivedFrom: MISRA C:2012 Rule 18.8</p> <p>Variable-length array types shall not be used.</p>

Table 94 Ensure Usability of BSW Modules in C++ Environments

Id	Rule Chapter 3.2.10
[Abstr_CPP_001]	<p>Don't use C++ Keywords not Defined as Keywords in C</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"> ▶ 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
[Abstr_CPP_002]	<p>Type of Character Literal</p> <p>Externally usable macros and inline functions shall not depend on the type of character literal.</p>
[Abstr_CPP_003]	<p>Use Cast to Assign String Literals</p> <p>Externally usable macros and inline functions shall use a cast to non-const type when a string literal is assigned to a pointer.</p>
[Abstr_CPP_004]	<p>Don't use Double Definitions</p> <p>Externally usable macros and inline functions shall not use repeated type definitions.</p>
[Abstr_CPP_005]	<p>Don't Refer to Structs Defined in Structs from Outside</p> <p>Externally usable macros and inline functions shall not refer to structs, enumerations or enumerator names outside the struct in which those are defined.</p>
[Abstr_CPP_006]	<p>Declare const Objects with External Linkage in C code Explicitly extern</p> <p>Externally usable macros and inline functions shall declare <i>const</i> objects with external linkage in C code explicitly <i>extern</i>.</p>
[Abstr_CPP_007]	<p>Use Cast to Convert void* to a Pointer Type</p> <p>Externally usable macros and inline functions shall use a cast to convert a <i>void*</i> to a pointer type.</p>
[Abstr_CPP_008]	<p>Only Pointers to non-const and non-volatile Objects may be Implicitly Converted to void*</p> <p>Within externally usable macros and inline functions only pointers to non-const and non-volatile objects may be implicitly converted to <i>void*</i>.</p>
[Abstr_CPP_009]	<p>const Objects shall be Initialized</p> <p>Externally usable macros and inline functions shall initialize <i>const</i> objects.</p>

Id	Rule Chapter 3.2.10
[Abstr_CPP_010]	Don't Use auto as a Storage Class Specifier Externally usable macros and inline functions shall not use auto as a storage class specifier.
[Abstr_CPP_011]	Assign only Values of the same Type to Enumeration Objects Within externally usable macros and inline functions only values of the same type shall be assigned to enumeration objects.
[Abstr_CPP_012]	Don't let the Code Depend on the Type and Size of Enumeration Objects Within externally usable macros and inline functions code shall not depend on the type and size of enumeration objects.
[Abstr_CPP_013]	Use a Cast or Separate Implementations to Copy Volatile Structs Externally usable macros and inline functions code shall use a cast when copying volatile structs to non-volatile structs.
[Abstr_CPP_014]	Don't use __STDC__ Within externally usable macros and inline functions the value of __STDC__ shall not be used.
[Abstr_CPP_015]	Encapsulate C code Conflicting with C++ and Provide C++ Substitution Code 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.

Table 95 Prohibition of Open Source Software

Id	Rule Chapter 3.2.11
[Abstr_OSS_001]	Prohibition of Open Source Software Open Source Software (OSS) shall not be used within BSW modules.

Table 96 Base Integer and Float Data Types

Id	Rule Chapter 3.3.1
[CCode_Types_-001]	Platform Integer and Float Data Types DerivedFrom: MISRA C:2012 Dir 4.6 The 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.
[CCode_Types_-002]	Prohibition of Plain C Data Types The usage of plain C data types "int", "short" and "long" is forbidden.
[CCode_Types_-003]	Usage of Char Data Type The plain "char" data type shall only be used for the storage and use of character values or data.
[CCode_Types_-004]	Handling of 64 Bit Data Types 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 be used in exceptional cases.
[CCode_Types_-005]	Handling of Own Defined Data Types 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 97 Optimized Integer Data Types

Id	Rule Chapter 3.3.2
[CCode_Types_-006]	Common Optimized Integer Data Types DerivedFrom: MISRA C:2012 Dir 4.6 The 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.

Id	Rule <i>Chapter 3.3.2</i>
[CCode_Types_007]	<p>Handling of Optimized Integer Data Types 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.</p>

Table 98 Standard Symbols

Id	Rule <i>Chapter 3.3.3</i>
[CCode_Symbols_001]	<p>TRUE and FALSE 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.</p>
[CCode_Symbols_002]	<p>CPU Specific Symbols The following CPU specific common symbols are available: CPU_TYPE and CPU_BYTE_ORDER.</p>
[CCode_Symbols_003]	<p>Common Symbols The following other common symbols may be used: E_OK, E_NOT_OK, STD_HIGH, STD_LOW, STD_ACTIVE, STD_IDLE, STD_ON, STD_OFF.</p>
[CCode_Symbols_004]	<p>Standard Version Info Type Type "Std_VersionInfoType" shall be used to request the version of a BSW module.</p>
[CCode_Symbols_005]	<p>Usage of Null Pointer DerivedFrom: MISRA C:2012 Rule 11.9 For null pointers "NULL_PTR" shall be used.</p>

Table 99 Specific Types and Symbols for Communication Software

Id	Rule <i>Chapter 3.3.4</i>
[CCode_Com-StackTypes_001]	<p>Type for PDU Identifiers Type "PduldType" has to be used to define variables for unique identifiers for PDUs.</p>
[CCode_Com-StackTypes_002]	<p>Handling of Types for PDU Identifiers Variables of type "PduldType" shall be zero-based and consecutive, in order to be able to perform table-indexing within a software module.</p>
[CCode_Com-StackTypes_003]	<p>Type for Length Information of a PDU Type "PduLengthType" shall be used to define length information of a PDU which is provided in number of bytes.</p>
[CCode_Com-StackTypes_004]	<p>Type for Basic Information of a PDU Type "PdulInfoType" has to be used to store the basic information about a PDU.</p>
[CCode_Com-StackTypes_005]	<p>Type for Result of a Buffer Request Type "BufReq_ReturnType" shall be used to define variables to store the result of a buffer request.</p>
[CCode_Com-StackTypes_006]	<p>Type for Result Status of a Notification Type "NotifResultType" shall be used to define variables to store the result status of a notification (confirmation or indication).</p>
[CCode_Com-StackTypes_007]	<p>Naming of Return Codes of a Notification Class: NamingConvention Return codes of a notification shall be named as follows: NTFRSLT_E_<Communication System Abbreviation>_<Error Code Name>.</p>
[CCode_Com-StackTypes_008]	<p>Type for Bus Status Evaluated by a Transceiver Type "BusTrcvErrorType" shall be used to define variables to return the bus status evaluated by a transceiver.</p>
[CCode_Com-StackTypes_009]	<p>Naming of Return Codes of a Bus Status Notification Class: NamingConvention Return codes of a bus status notification shall be named as follows: BUSTRCV_E_<Communication System Abbreviation>_<Error Code Name>.</p>

Id	Rule <i>Chapter 3.3.4</i>
[CCode_Com-StackTypes_010]	Type for State of a Transport Protocol Buffer Type "TpDataStateType" shall be used to define variables to store the state of a Transport Protocol (TP) buffer.
[CCode_Com-StackTypes_011]	Type for Transport Protocol Buffer Handling Type "RetryInfoType" shall be used to define variables to store the information about TP buffer handling.
[CCode_Com-StackTypes_012]	Type for Identifiers of Communication Channels Type "NetworkHandleType" shall be used to define variables to store the identifier of a communication channel.
[CCode_Com-StackTypes_013]	Type to Specify a Parameter Type "TpParameterType" shall be used in "ChangeParameter" interfaces to specify the parameter to which the value has to be changed.

Table 100 Module Specific Add Ons

Id	Rule <i>Chapter 3.3.5</i>
[SpecificTypes_5]	Handling of Additional Types and Symbols 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 101 Rule Set: Naming Convention

Id	Rule <i>Chapter 3.4</i>
[CDGNaming_-001]	Definition of a Component Prefix Class: NamingConvention DerivedFrom: MISRA C:2012 Rule 5.8 Every name with a global visibility shall be prefixed with a component prefix.
[CDGNaming_-002]	Macros Class: NamingConvention DerivedFrom: MISRA C:2012 Dir 4.5 Macro names shall be defined in the following form: <COMPONENT PREFIX>{_IDENTIFIER}_<UPPER-TEXT>.
[CDGNaming_-003]	C-Identifiers with File Scope Class: NamingConvention DerivedFrom: MISRA C:2012 Dir 4.5 C-identifiers with file scope shall be defined in the following form: <Component prefix>{_identifier}<pp><DescriptiveText>{_Typesuffix}.
[CDGNaming_-004]	Typedefs as Unique Identifiers Class: NamingConvention DerivedFrom: MISRA C:2012 Rule 5.6 A typedef name shall be a unique identifier.
[CDGNaming_-005]	Tag Names as Unique Identifiers Class: NamingConvention DerivedFrom: MISRA C:2012 Rule 5.7 A tag name shall be a unique identifier.

Id	Rule Chapter 3.4
[CDGNaming_-006]	<p>C-Identifiers with Block Scope</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Dir 4.5</p> <p>C-identifiers with block scope or function prototype scope shall be defined in the following form: { s_ }<pp><DescriptiveText>_<Typesuffix>.</p>
[CDGNaming_-007]	<p>Usage of Object of Function Identifier</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Rule 5.9</p> <p>No object or function identifier with static storage duration should be reused.</p>
[CDGNaming_-008]	<p>Spelling of Identifiers in Different Name Spaces</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Dir 4.5</p> <p>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.</p>
[CDGNaming_-009]	<p>Identifiers of Inner and Outer Scope</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Rule 5.3</p> <p>An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.</p>
[CDGNaming_-010]	<p>Significance of Identifiers</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Rule 5.1</p> <p>DerivedFrom: MISRA C:2012 Rule 5.2</p> <p>DerivedFrom: MISRA C:2012 Rule 5.4</p> <p>DerivedFrom: MISRA C:2012 Rule 5.5</p> <p>Identifiers (internal and external) shall not rely on the significance of more than 60 characters. Identifiers shall be distinct.</p>
[CDGNaming_-011]	<p>File and Directory Names</p> <p>Class: NamingConvention</p> <p>File and directory names shall be defined in a specified form.</p>
[CDGNaming_-012]	<p>SymbolicNameValue</p> <p>Class: NamingConvention</p> <p>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</p> <ul style="list-style-type: none"> ▶ 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.
[CDGNaming_-013]	<p>Usage of U Suffixes for Unsigned Integer Constants</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Rule 7.2</p> <p>A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.</p>
[CDGNaming_-014]	<p>Usage of L as Suffix for Constants</p> <p>Class: NamingConvention</p> <p>DerivedFrom: MISRA C:2012 Rule 7.3</p> <p>The lowercase character "l" shall not be used in a literal suffix.</p>

Id	Rule <i>Chapter 3.4</i>
[CDGNaming_-015]	<p>Usage of f as Suffix for Float Constants</p> <p>Class: NamingConvention Float constants of single precision shall be suffixed with a "f "character.</p>

Table 102 Basis Set of Module Files

Id	Rule <i>Chapter 3.5.1</i>
[BSW_Files_001]	<p>Basic Set Of Module Files</p> <p>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.</p>
[BSW_Files_002]	<p>Additional Module C Files</p> <p>If a module provides several functions and processes additional module source files "<Module>_<Sub>.c" may be used. Name <Sub> can be chosen freely.</p>
[BSW_Files_003]	<p>Header for Callback Functions</p> <p>Declarations of callback functions shall be grouped and out-sourced in a separate header file "<Module>_Cbk.h".</p>
[BSW_Files_004]	<p>Files for ECU Configuration</p> <p>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 the 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.</p>
[BSW_Files_005]	<p>Header for Module Specific Types</p> <p>Module specific types and symbols can be defined in header "<Module>_Types.h".</p>
[BSW_Files_006]	<p>Header for Exclusive Interfaces for another Module</p> <p>Interfaces which are provided exclusively for one module should be separated into a dedicated header file "<Module>_<User>.h".</p>
[BSW_Files_007]	<p>Additional Module Headers</p> <p>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.</p>
[BSW_Files_008]	<p>Private Module Header</p> <p>Elements which shall not be exported via module header file "<Module>.h" shall be placed in private header "<Module>_Prv.h".</p>

Table 103 Header Include Concept

Id	Rule <i>Chapter 3.5.2</i>
[BSW_HeaderInc_-001]	<p>Definition of BSW Header Include Concept</p> <p>The header include concept explained within the following rules shall be to be considered.</p>
[BSW_HeaderInc_-002]	<p>Include Order of Module Header</p> <p>"<Module>.h" header has to be included as first header in every c file (functional and configuration) of a module.</p>

Id	Rule Chapter 3.5.2
[BSW_HeaderInc_003]	<p>Include Order after Module Header</p> <p>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").</p> <p>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.</p>
[BSW_HeaderInc_004]	<p>Include Order of Types Header inside Module Header</p> <p>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".</p>
[BSW_HeaderInc_005]	<p>Include Order of other Headers inside Module Header</p> <p>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").</p> <p>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.</p>
[BSW_HeaderInc_006]	<p>Include Order of Private Headers</p> <p>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.</p>
[BSW_HeaderInc_007]	<p>Module Export Headers</p> <p>"<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.</p>
[BSW_HeaderInc_008]	<p>Import of Headers from Other Modules</p> <p>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").</p>
[BSW_HeaderInc_009]	<p>Protection against Multiple Inclusion</p> <p>Scope: All headers except MemMap headers</p> <p>DerivedFrom: MISRA C:2012 Rule 20.9</p> <p>Precautions shall be taken in order to prevent the contents of a header file being included more than once.</p>
[BSW_HeaderInc_010]	<p>Preprocessor Check for AUTOSAR Headers</p> <p>A pre-processor check shall be performed for all included AUTOSAR based header files (Inter Module Check).</p>
[BSW_HeaderInc_011]	<p>Correct Spelling of Headers within #include Directives</p> <p>The name of a header within an #include directive shall be correctly spelled, i.e. mind upper and lower case. The name of an included header shall be syntactically identical to the file name of the header.</p>

Table 104 BSW Service Module with and without RTE

Id	Rule <i>Chapter 3.5.3</i>
[BSW_ServiceRT_E_001]	<p>BSW Service Module with and without RTE: Conditions for RTE headers</p> <p>Scope: BSW service modules which provide a SWCD file and have a close connection to the <i>RTE</i> generator</p> <p>To support that a BSW service module can work with and without a <i>RTE</i> generator the following <i>RTE</i> specific files shall be provided with the following content:</p> <ul style="list-style-type: none"> ▶ 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 <i>RTE</i> is available, but in case of an existing <i>RTE</i> generator this file is anyway mandatory
[BSW_ServiceRT_E_002]	<p>BSW Service Module with and without RTE: Conditions for Module Header</p> <p>Scope: BSW service modules which provide a SWCD file and have a close connection to the <i>RTE</i> generator</p> <p>To support that a BSW service module can work with and without a <i>RTE</i> generator the module header / the module types header shall fulfill the following conditions:</p> <ul style="list-style-type: none"> ▶ 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 <i>RTE</i> 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"

Table 105 Design of APIs (Application Programming Interfaces)

Id	Rule <i>Chapter 3.5.4</i>
[BSW_APIDesign_001]	<p>Return Type of Initialization Functions</p> <p>Status: removed</p> <p>ReplacedBy: [BSW_ProcISR_001]</p>
[BSW_APIDesign_002]	<p>Design of Main Processes</p> <p>Status: removed</p> <p>ReplacedBy: [BSW_ProcISR_002]</p>
[BSW_APIDesign_003]	<p>Callback Functions</p> <p>Callback functions shall avoid return types other than void if possible.</p>
[BSW_APIDesign_004]	<p>Prohibition of Function Pointers as Parameter</p> <p>Function pointers shall not be used as a parameters of an API.</p>
[BSW_APIDesign_005]	<p>Separation of Error and Status Information</p> <p>All software modules shall strictly separate error and status information in return values and also in internal variables.</p>
[BSW_APIDesign_006]	<p>Design of Instances of BSW Modules</p> <p>Instances of BSW modules shall be accessed index based if they are characterized by same vendor, same functionality and same hardware device.</p>
[BSW_APIDesign_007]	<p>Unit of Time Parameters</p> <p>The unit of time for specification and configuration of BSW modules shall be preferably in physical time unit, not ticks.</p>
[BSW_APIDesign_008]	<p>Usage of Standard Return Type</p> <p>APIs with a connection to <i>RTE</i> have to use "Std_ReturnType" as standard API return type.</p>

Id	Rule Chapter 3.5.4
[BSW_APIDesign - 009]	<p>Test of Error Information</p> <p>DerivedFrom: MISRA C:2012 Dir 4.7</p> <p>If a function returns error information, then that error information shall be tested.</p>
[BSW_APIDesign - 010]	<p>Stable Number of Parameters</p> <p>DerivedFrom: MISRA C:2012 Rule 17.1</p> <p>Functions shall not be defined with a variable number of arguments.</p>
[BSW_APIDesign - 011]	<p>Prohibition of Recursive Function Calls</p> <p>DerivedFrom: MISRA C:2012 Rule 17.2</p> <p>Functions shall not call themselves, either directly or indirectly. Recursive function calls are not allowed.</p>
[BSW_APIDesign - 012]	<p>Number of Arguments identical to Number of Parameters</p> <p>DerivedFrom: MISRA C:2012 Rule 17.3</p> <p>The number of arguments passed to a function shall match the number of parameters.</p>
[BSW_APIDesign - 013]	<p>Explicit Return Statement</p> <p>DerivedFrom: MISRA C:2012 Rule 17.4</p> <p>All exit paths from a function with non-void return type shall have an explicit return statement with an expression.</p>
[BSW_APIDesign - 014]	<p>Call of Functions via Identifier</p> <p>A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.</p>
[BSW_APIDesign - 015]	<p>Structure Passed to Functions as Pointer</p> <p>A structure as parameter for a function should be passed as pointer.</p>
[BSW_APIDesign - 016]	<p>Handling of Unused Parameters</p> <p>DerivedFrom: MISRA C:2012 Rule 2.7</p> <p>Unused parameters shall be handled with a void cast.</p>
[BSW_APIDesign - 017]	<p>Change of an Interface leads to a New Interface</p> <p>To ensure compatibility a change of an interface or a changed interpretation of interface parameters shall always lead to a definition of a new interface.</p>
[BSW_APIDesign - 018]	<p>APIs are Functions Called by Other Components</p>
[BSW_APIDesign - 019]	<p>APIs shall be executable on all cores.</p>
[BSW_APIDesign - 020]	<p>Appropriate Number of Array Elements in Function Arguments</p> <p>DerivedFrom: MISRA C:2012 Rule 17.5</p> <p>The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.</p>
[BSW_APIDesign - 021]	<p>No static Keyword between [] of an Array Parameter</p> <p>Scope: C99 based code</p> <p>DerivedFrom: MISRA C:2012 Rule 17.6</p> <p>The declaration of an array parameter shall not contain the static keyword between the [].</p>
[BSW_APIDesign - 022]	<p>Usage of Non-Void Return Types</p> <p>DerivedFrom: MISRA C:2012 Rule 17.7</p> <p>The value returned by a function having non-void return type shall be used.</p>
	<p>No Modification of Function Parameters</p>
	<p>DerivedFrom: MISRA C:2012 Rule 17.8</p>
	<p>A function parameter should not be modified.</p>

Table 106 Design of Processes and Interrupt Service Routines

Id	Rule <i>Chapter 3.5.5</i>
[BSW_ProcISR_-001]	Return Value of Initialization Processes The return type of initialization processes shall be void.
[BSW_ProcISR_-002]	Parameters of Initialization Processes 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.
[BSW_ProcISR_-003]	Interface of Scheduled Processes Scheduled processes shall have no parameters and no return value.
[BSW_ProcISR_-004]	Internal Design of Scheduled Processes Scheduled processes shall be designed to run in parallel with APIs and ISRs on other cores.
[BSW_ProcISR_-005]	Interface of Interrupt Service Routines Interrupt Service Routines shall have no parameters and no return value.
[BSW_ProcISR_-006]	Internal Design of Interrupt Service Routines Interrupt Service Routines shall be executable on any core.
[BSW_ProcISR_-007]	No Dem/FIM Calls Within Interrupt Service Routines Interrupt Service Routines shall not call Dem (Diagnostic Event Manager) or FIM (Function Inhibition Manager) services. In general it should be avoided to call services within an Interrupt Service Routine.

Table 107 Definition and Declaration of Functions and Objects

Id	Rule <i>Chapter 3.5.6</i>
[BSW_FuncObj_-001]	Definitions only in C Files There shall be no definitions of objects or functions (except inline functions) in a header file. Definitions shall take place in a c file.
[BSW_FuncObj_-002]	Explicit Functions and Objects DerivedFrom: MISRA C:2012 Rule 8.1 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.
[BSW_FuncObj_-003]	Void Functions DerivedFrom: MISRA C:2012 Rule 8.2 Function with no parameters shall be declared and defined with the parameter list void.
[BSW_FuncObj_-004]	Number of Definitions of Objects and Functions DerivedFrom: MISRA C:2012 Rule 8.6 An identifier (object or function) with external linkage or global visibility shall have exactly one external definition in one file.
[BSW_FuncObj_-005]	Scope of Functions Functions shall be declared at file scope and not on block scope.
[BSW_FuncObj_-006]	Prototype Declarations DerivedFrom: MISRA C:2012 Rule 8.2 DerivedFrom: MISRA C:2012 Rule 8.4 DerivedFrom: MISRA C:2012 Rule 8.5 DerivedFrom: MISRA C:2012 Rule 17.3 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.

Id	Rule Chapter 3.5.6
[BSW_FuncObj_007]	<p>Equality of Declaration and Definition of a Function</p> <p>DerivedFrom: MISRA C:2012 Rule 8.2</p> <p>DerivedFrom: MISRA C:2012 Rule 8.3</p> <p>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.</p>
[BSW_FuncObj_008]	<p>Static Objects and Functions</p> <p>DerivedFrom: MISRA C:2012 Rule 8.8</p> <p>The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.</p>
[BSW_FuncObj_009]	<p>Constant Pointer Parameters</p> <p>DerivedFrom: MISRA C:2012 Rule 8.13</p> <p>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.</p>
[BSW_FuncObj_010]	<p>Compatible Types of multiple declared Objects of Functions</p> <p>If objects or functions are declared more than once their types shall be compatible.</p>
[BSW_FuncObj_011]	<p>Global Constants</p> <p>Global visible constant data shall be indicated with read-only purposes by explicitly assigning the const keyword.</p>
[BSW_FuncObj_012]	<p>Static Objects and Functions</p> <p>DerivedFrom: MISRA C:2012 Rule 8.7</p> <p>Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.</p>
[BSW_FuncObj_013]	<p>Global Constants</p> <p>DerivedFrom: MISRA C:2012 Rule 7.4</p> <p>A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char".</p>

Table 108 Code Re-entrancy

Id	Rule Chapter 3.5.7
[BSW_Reentrancy_002]	<p>Services and APIs are Multicore Re-entrant</p> <p>All services and APIs of a module shall be multicore re-entrant.</p>
[BSW_Reentrancy_003]	<p>Scheduled Functions (Processes) are Multicore Re-entrant</p> <p>Status: removed</p> <p>ReplacedBy: Rules of chapter 3.5.5</p>
[BSW_Reentrancy_004]	<p>Atomic data access</p> <p>Assumption about atomic data access shall be documented in the chapter for integration in the product documentation.</p>

Table 109 Providing Version Information

Id	Rule Chapter 3.5.8
[BSW_VersionInfo_001]	<p>Provision Version Information</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>Each AUTOSAR based BSW module shall provide information to identify vendor, module and SW version.</p>

Id	Rule <i>Chapter 3.5.8</i>
[BSW_VersionInfo_002]	<p>Module Vendor Identifier</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>A module vendor identifier "<MODULE>_VENDOR_ID" shall be provided in the module header file.</p>
[BSW_VersionInfo_003]	<p>Module Identifier</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>A module identifier "<MODULE>_MODULE_ID" shall be provided in the module header file.</p>
[BSW_VersionInfo_004]	<p>Software Version Number and AUTOSAR Specification Version Number</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>A software version number and an AUTOSAR specification version number shall be provided in the module header file.</p> <p>The software version number shall be updated even before a new version of the software will be released.</p> <p>The AUTOSAR specification version number shall be updated even if the module supports a new version of an AUTOSAR specification.</p>
[BSW_VersionInfo_005]	<p>Provision of GetVersionInfo Interface</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>Each AUTOSAR based BSW module shall provide a function <Module>_GetVersionInfo to read out published parameters.</p>
[BSW_VersionInfo_006]	<p>Usage of GetVersionInfo Interface</p> <p>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.</p>

Table 110 BSW Scheduler and Exclusive Areas

Id	Rule <i>Chapter 3.5.9</i>
[BSW_Sched_001]	<p>Suppressing and Resuming Interrupts</p> <p>DerivedFrom: MISRA C:2012 Dir 4.13</p> <p>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.</p>

Table 111 Common File Style

Id	Rule <i>Chapter 3.6.1</i>
[CCode_Style_-001]	<p>Unique Style</p> <p>Each C file, header and script source file shall follow common settings to get an unique style.</p>
[CCode_Style_-002]	<p>Standard File Header</p> <p>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.</p>
[CCode_Style_-003]	<p>Special Comment Blocks</p> <p>To structure source files special comment blocks may be used to cluster header includes, definitions, declaration and code.</p>
[CCode_Style_-004]	<p>Line Length</p> <p>The line length shall not exceed 120 characters.</p>
[CCode_Style_-005]	<p>Indentation</p> <p>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.</p>
[CCode_Style_-006]	<p>Definition of Variables</p> <p>Only one variable shall be defined in a single line. A multi line or comma separated definition is not allowed.</p>

Id	Rule Chapter 3.6.1
[CCode_Style_-007]	<p>Initialization of Arrays and Structures</p> <p>DerivedFrom: MISRA C:2012 Rule 9.2</p> <p>DerivedFrom: MISRA C:2012 Rule 9.3</p> <p>Braces shall be used to indicate and match the structure in a non-zero initialization of arrays, unions and structures.</p>
[CCode_Style_-008]	<p>Operators and Whitespaces</p> <p>Mathematical, logical, comparison and conditional operators shall be embedded in whitespaces. This rule does not apply to brackets and operators like !, ++ and --.</p>
[CCode_Style_-009]	<p>Nested Preprocessor Commands</p> <p>Nested blocks of preprocessor commands should use a special form of indentation.</p>
[CCode_Style_-010]	<p>Display History Information</p> <p>To display history information from the SCM system a special comment block shall be placed at end of a file.</p>
[CCode_Style_-011]	<p>Display Header File Name</p> <p>The name of a header file should be named within a comment at the end of each header file.</p>
[CCode_Style_-012]	<p>New Line at End of File</p> <p>Each c file and header shall end with a new line to avoid compiler warnings.</p>
[CCode_Style_-013]	<p>Avoiding of trailing spaces</p> <p>Trailing spaces shall be avoided.</p>

Table 112 Block Style

Id	Rule Chapter 3.6.2
[CCode_BlockStyle_001]	<p>Common Block Style Layout</p> <p>DerivedFrom: MISRA C:2012 Rule 15.6</p> <p>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 body of an iteration-statement or a selection-statement shall be a compound-statement.</p>
[CCode_BlockStyle_002]	<p>Block Brackets are Single Characters in a Line</p> <p>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.).</p>
[CCode_BlockStyle_003]	<p>Horizontal Position of Brackets of Blocks</p> <p>"{" and "}" shall be placed at the same horizontal position.</p>
[CCode_BlockStyle_004]	<p>Instruction Block with One Instruction is a Block</p> <p>DerivedFrom: MISRA C:2012 Rule 15.6</p> <p>If an instruction block only contains one instruction, the block shall nevertheless be embraced with "{" and "}".</p>

Table 113 Comments

Id	Rule Chapter 3.6.3
[CCode_Comments_001]	<p>Usage of Comments</p> <p>Code which is not self-explanatory shall be extended by a meaningful comment. Also workarounds and rule violations shall be commented, too.</p>
[CCode_Comments_002]	<p>Allowed Comment Marker</p> <p>It is allowed to use "/* ... */" or "://" as comment marker styles to write a comment.</p>

Id	Rule <i>Chapter 3.6.3</i>
[CCode_Comments_003]	<p>Non-Nested Comments</p> <p>DerivedFrom: MISRA C:2012 Rule 1.3</p> <p>DerivedFrom: MISRA C:2012 Rule 3.1</p> <p>Comments shall not be nested and comment styles shall not be mixed.</p>
[CCode_Comments_004]	<p>Sections of Code Should Not be Commented Out</p> <p>DerivedFrom: MISRA C:2012 Dir 4.4</p> <p>Sections of code should not be "commented out".</p>
[CCode_Comments_005]	<p>Language of Comments is English</p> <p>Comments shall be written in English.</p>
[CCode_Comments_006]	<p>Position of Comments</p> <p>It is allowed to write a comment behind code but it is not allowed to write code behind a comment.</p>
[CCode_Comments_007]	<p>Markers for Requirements Tracing</p> <p>A special marker may be used within comments to trace requirements from AUTOSAR: TRACE[<AUTOSAR SWS Trace ID>].</p>
[CCode_Comments_008]	<p>Publishable Contents of Comments</p> <p>All contents of comments shall be publishable. That means that comments ...</p> <ul style="list-style-type: none"> ▶ 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"
[CCode_Comments_009]	<p>No Line-Splicing within // Comments</p> <p>DerivedFrom: MISRA C:2012 Rule 3.2</p> <p>Line-splicing shall not be used in // comments.</p>

Table 114 Authoring of ECUC artifacts

Id	Rule <i>Chapter 4.1.1</i>
[ECUC_001]	<p>AUTOSAR conformance</p> <p>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).</p> <p>All AUTOSAR ECUC artifacts shall be conforming to the corresponding AUTOSAR specifications.</p>
[ECUC_002]	<p>AUTOSAR version</p> <p>On the head of the main development branch, all AUTOSAR ECUC artifacts shall be based on AUTOSAR release 4.0.2 or higher.</p>

Table 115 ECUC Values: Content Responsibility

Id	Rule <i>Chapter 4.2.1</i>
[ECUC_V001]	<p>Content responsibility</p> <p>Customer projects are responsible for ECUC values files unless this responsibility has explicitly been delegated.</p> <p>Project-specific ECUC values files are named <Comp>_EcucValues.arxml¹⁰.</p>

¹⁰ See also the guideline on file naming rules for an overview of all file naming conventions.

Id	Rule <i>Chapter 4.2.1</i>
[ECUC_V002]	Protected configuration
	Protected ECUC values shall not be changed by any developer who receives these files as a deliverable ¹¹ .
	Protected ECUC values files are named <Comp>_Prot_EcucValues.arxml.
[ECUC_V003]	Configuration examples
	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.
[ECUC_V004]	Release conditions
	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 116 ECUC Values: Editing, Checking, and Reviewing

Id	Rule <i>Chapter 4.2.2</i>
[ECUC_V005]	Editing
	ECUC values shall be edited with BCT.
[ECUC_V006]	Formal checks
	ECUC values have to be formally checked before delivery by processing it in a suitable project context.
[ECUC_V007]	Reviewing
	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 117 ECUC Values: Testing

Id	Rule <i>Chapter 4.2.3</i>
[ECUC_V008]	Testing
	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 118 ECUC Values: Content-related Procedures

Id	Rule <i>Chapter 4.2.4</i>
[ECUC_V009]	Container short names
	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.

¹¹ 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 Chapter 4.2.4
[ECUC_V010]	<p>ArPackage usage</p> <p>All ECU configuration values shall be within the ArPackage hierarchy /RB/UBK/Project/EcucModuleConfigurationValues.</p> <p>See the guideline rules for ArPackage usage for details.</p>
[ECUC_V011]	<p>EcucValueCollection usage</p> <p>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.</p>

Table 119 ECUC ParamDefs: Content Responsibility

Id	Rule Chapter 4.3.1
[ECUC_PD001]	<p>Content responsibility</p> <p>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.arxml.</p>

Table 120 ECUC ParamDefs: Editing, Checking, and Reviewing

Id	Rule Chapter 4.3.2
[ECUC_PD002]	<p>Editing</p> <p>ECUC parameter definitions shall be changed in a backwards-compatible way whenever possible. Incompatible changes shall be documented.</p>
[ECUC_PD003]	<p>Formal checks</p> <p>ECUC parameter definitions shall be formally checked for validity against the corresponding version of the AUTOSAR schema before delivery.</p>
[ECUC_PD004]	<p>Reviewing</p> <p>After a successful formal check, the parameter definitions shall be reviewed against the requirements which triggered their creation or update.</p>

Table 121 ECUC ParamDefs: Content-related Procedures

Id	Rule Chapter 4.3.4
[ECUC_PD005]	<p>Vendor-specific derivation</p> <p>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.</p>
[ECUC_PD006]	<p>One module definition per file</p> <p>No ECUC parameter definition file shall contain the definition of more than one <ECUC-MODULE-DEF>.</p>
[ECUC_PD007]	<p>English language</p> <p>The only language to be used is English.</p>
[ECUC_PD008]	<p>EcucDefinitionCollection usage</p> <p>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.</p>

Id	Rule Chapter 4.3.4
[ECUC_PD009]	<p>Naming conventions for short names</p> <p>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".</p>
[ECUC_PD010]	<p>Order of parameters</p> <p>The <SHORT-NAME>s of all items inside a container shall be in the following order:</p> <ol style="list-style-type: none"> 1. all parameters in alphabetical order 2. all references in alphabetical order 3. all sub-containers in alphabetical order.
[ECUC_PD011]	<p>Long name of parameters</p> <p>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]".</p>
[ECUC_PD012]	<p>Description of parameters</p> <p>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.</p>
[ECUC_PD013]	<p>Specification of parameter origin</p> <p>The <ORIGIN> of any parameter not defined by AUTOSAR shall be set to "RB", possibly followed by a version and/or date specifier.</p>
[ECUC_PD014]	<p>ArPackage usage</p> <p>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.</p>
[ECUC_PD015]	<p>Anticipation of parameters from future AUTOSAR releases</p>
[ECUC_PD016]	<p>Multiplicity handling</p> <p>If the multiplicities of a standardized ECUC parameter shall be changed in a vendor-specific derivation, the following rules apply:</p> <ul style="list-style-type: none"> ▶ 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.
[ECUC_PD017]	<p>Default values for mandatory parameters</p> <p>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.</p>

Table 122 ECUC Processors: Content Responsibility and Language

Id	Rule Chapter 4.4.1
[ECUC_P001]	<p>Content responsibility</p> <p>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.</p>
[ECUC_P002]	<p>Language</p> <p>The only allowed languages for ECUC processors are oAW (openArchitectureWare) and Perl. No other languages are allowed.</p> <p>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:</p> <ul style="list-style-type: none"> ▶ MemStack ▶ IoStack ▶ MCAL layer except MCAL SW components belonging to the ComStack ▶ Calibration Software ▶ ECUSec ▶ Boot Control ▶ BSW Library. <p>CDG packages shall not contain a mix of ECUC processor languages.</p>

Table 123 ECUC Processors: Editing, Checking, and Reviewing

Id	Rule Chapter 4.4.2
[ECUC_P003]	<p>Reviewing</p> <p>Before delivery, ECUC processors shall be reviewed against the corresponding requirements and existing content-related procedures (see below).</p>

Table 124 ECUC Processors: Testing

Id	Rule Chapter 4.4.3
[ECUC_P004]	<p>Testing</p> <p>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.</p> <p>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.</p> <p>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.</p>

Table 125 ECUC Processors: Documentation

Id	Rule Chapter 4.4.4
[ECUC_P005]	<p>Documenting</p> <p>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.</p>

Table 126 ECUC Processors: General Content-related Procedures

Id	Rule Chapter 4.4.5
[ECUC_P006]	<p>Allowed output</p> <p>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 configuration¹².</p> <p>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.</p> <p>For generated code and meta information, the same quality demands and coding guidelines apply as for manually created code.</p>
[ECUC_P007]	<p>Stable output</p> <p>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:</p> <ul style="list-style-type: none"> ▶ 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.
[ECUC_P008]	<p>Allowed input</p> <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>
[ECUC_P009]	<p>Independence from input origin</p> <p>ECUC processors are not allowed to react differently in case of</p> <ul style="list-style-type: none"> ▶ manually created input ECUC values resp. ▶ forwarded ECUC values.
[ECUC_P010]	<p>Independence from container short names in ECUC Values</p> <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>
[ECUC_P011]	<p>(Removed)</p> <p>Removed.</p>
[ECUC_P012]	<p>Naming conventions for input files</p> <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"> ▶ the corresponding oAW template shall be called Hugo_Cfg_h.xpt and ▶ the corresponding Perl template shall be called Hugo_Cfg.ht.

¹² 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>
[ECUC_P013]	<p>Naming conventions for output files</p> <p>The following naming conventions apply for files generated by ECUC processors:</p> <ul style="list-style-type: none"> ▶ Files containing PreCompile-time information shall be named <code><Comp>_Cfg[_<Sub>].<ext></code> ▶ Files containing link time information shall be named <code><Comp>_Lcfg[_<Sub>].<ext></code> ▶ Files containing PostBuild-time information shall be named <code><Comp>_PBcfg[_<Sub>].<ext></code> ▶ Files defining AUTOSAR interfaces to ASW shall be named <code><Comp>_Cfg[_<Sub>]_SWCD.xml</code> ▶ Files defining (fragments of) BSW module descriptions shall be named <code><Comp>_Cfg[_<Sub>]_BSWMD.xml</code> ▶ Report files shall be named <code><Comp>_Report.txt</code> ▶ Export files shall be named <code><Comp>_Export.xml</code>
[ECUC_P014]	<p>Clear separation of actions</p> <p>ECUC processors fulfil the following tasks (not necessarily all of them):</p> <ul style="list-style-type: none"> ▶ 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. <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>
[ECUC_P015]	<p>Validation responsibility</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>
[ECUC_P016]	<p>Problem reporting</p> <p>Problem reporting in ECUC processors shall meet the following expectations:</p> <ul style="list-style-type: none"> ▶ 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.
[ECUC_P017]	<p>Logging</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>
[ECUC_P018]	<p>Generating configuration documentation and reports</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>
[ECUC_P019]	<p>Handling of PostBuild data sets</p> <p>The following conventions apply for multiple configuration containers (used for post-build selectable configuration, marked as <MULTIPLE-CONFIGURATION-CONTAINER> set to <i>true</i> in the ECUC parameter definition):</p> <ul style="list-style-type: none"> ▶ Configuration: <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – 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.

Table 127 ECUC Processors: oAW-specific Procedures

Id	Rule <i>Chapter 4.4.6 [OP]</i>
[ECUC_OP_001]	<p>File naming conventions</p> <p>OAW files and methods corresponding to processor actions shall be named according to the following scheme (see also rule ECUC_P014).</p> <p>The <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders in a Perl processor.</p> <p>A oAW processor for a component <Comp> shall be a set of oAW scripts with the following naming conventions:</p> <p>Workflow Control (MWE, Model Workflow Environment) files:</p> <ul style="list-style-type: none"> – for configuration forwarding: <Comp>_Forward[<Suffix>].mwe – for configuration validation: <Comp>_Validate[<Suffix>].mwe – for code generator: <Comp>_Generate[<Suffix>].mwe – for ID generation: <Comp>_Prepare[<Suffix>].mwe <p>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.</p> <p>Forwarder scripts:</p>



Id	Rule Chapter 4.4.6 [OP]
	<p style="text-align: center;">△</p> <ul style="list-style-type: none"> - configuration forwarder files: <Comp>_Forward[<Suffix>].ext <p>Generate ID scripts:</p> <ul style="list-style-type: none"> - ID generator files: <Comp>_Prepare<Suffix>.chk - ID generator extension files: <Comp>_Prepare[<Suffix>].ext <p>Validator scripts:</p> <ul style="list-style-type: none"> - configuration validation files: <Comp>_Validate[<Suffix>].chk - configuration validation extension scripts: : <Comp>_Validate[<Suffix>].ext <p>Generator scripts:</p> <ul style="list-style-type: none"> - configuration generator template (Xpand) files: <Comp>_Generate[<Suffix>].xpt - configuration generator extension (Xtend) files: <Comp>_Generate[<Suffix>].ext - C code file generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_c.xpt - C header file generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_h.xpt - RTE configuration generator template (Xpand) files: <Comp>_[<Suffix>]_Cfg_Bswmd_<Suffix>.xpt and <Comp>_[<Suffix>]_Cfg_Swcd_[<Suffix>].xpt <p>The file name part <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders or prepare scripts within the same component.</p>
[ECUC_OP_002]	<p>Subdirectories</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"> - for generate ID and configuration forwarders: <i>scripts\forwarder</i> - for code and other file generators: <i>scripts\generator</i> - for configuration validators: <i>scripts\validator</i> - for utility extensions: <i>scripts\util</i>
[ECUC_OP_003]	<p>Forward scripts need separate action</p> <p>Forward scripts shall have a separate action defined with complete input and output data declaration in BAMF.</p>
[ECUC_OP_004]	<p>Id Generator scripts need separate action</p> <p>ID Generators shall have separate actions defined with complete input and output data declaration in BAMF.</p>
[ECUC_OP_005]	<p>Generator scripts need separate action</p> <p>Generator scripts shall have a separate actions defined with complete input data and output artifact declaration in BAMF.</p>

Table 128 ECUC Processors: Perl-specific Procedures

Id	Rule Chapter 4.4.7
[ECUC_PP001]	<p>File naming conventions</p> <p>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.</p>
[ECUC_PP002]	<p>Usage of the conf_process API</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>
[ECUC_PP003]	<p>Usage of global variables</p> <p>Global variables in Perl (i.e. variables in file scope) are evil. Try to avoid them whenever somehow possible.</p>

Id	Rule Chapter 4.4.7
[ECUC_PP004]	<p>Conventions for injection markers</p> <p>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.</p>
[ECUC_PP005]	<p>Action naming conventions</p> <p>Perl subroutines corresponding to processor actions shall be named according to the following scheme:</p> <ul style="list-style-type: none"> ▶ subroutines preparing the configuration input for further processing¹³ 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>]. <p>The <Suffix> is optional and shall only be used to distinguish several generators resp. forwarders in a Perl processor.</p>
[ECUC_PP006]	<p>Restriction of output file locations and names</p> <p>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.</p>
[ECUC_PP007]	<p>Dynamic registration of generated files</p> <p>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</p> <ul style="list-style-type: none"> ▶ shall be registered using the RegisterDynamicArtifact API of conf_process and ▶ shall match exactly one BAMF specification of created artifacts which uses wildcard operators. <p>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.</p>
[ECUC_PP008]	<p>No WhoAmI or Register in AUTOSAR BSW</p> <p>The legacy Perl subroutines WhoAmI() and Register() shall not be used in AUTOSAR BSW.</p>

Table 129 BuildAction Declarations

Id	Rule Chapter 4.5.1
[ECUC_B001]	<p>BuildActions names shall be unique</p> <p>All BuildActions to be executed within a given context (that is in a full program stand at max) shall have unique names.</p>
[ECUC_B002]	<p>Naming conventions for BuildActions</p> <p>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.</p>
[ECUC_B003]	<p>Declaring Startup-/Teardown Action References</p> <p>Startup-/Teardown Action References should be declared within just one BAMF file.</p>

¹³ 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>
[ECUC_B004]	Usage of Startup-/Teardown-/Followup and Predecessor Action References
	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.
[ECUC_B005]	Invocation Parameter keys
	A BuildAction instance shall not have two or more invocation parameters with the same key.

Table 130 Artefact Declarations

Id	Rule <i>Chapter 4.5.2.1</i>
[ECUC_BIOA001]	Artefact Declaration Composition
	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.
[ECUC_BIOA002]	Artefact Declaration Characteristics
	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]_
[ECUC_BIOA003]	Uniqueness of Output Declarations
	One BuildAction instance shall never contain duplicate output element declarations which will be applied to one and the same artefact.

Table 131 Model Reference Declarations

Id	Rule <i>Chapter 4.5.2.2</i>
[ECUC_BIOM001]	Required declaration of accessed Model Data
	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.
[ECUC_BIOM002]	Model Reference Identifiers
	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>
[ECUC_BIOM003]	Model Reference Category
	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 132 Action Specific Declarations

Id	Rule <i>Chapter 4.5.3</i>
[ECUC_BAS001]	<p>Artifact marked as processor</p> <p>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.</p>
[ECUC_BAS002]	<p>Generated Artefacts for Perl Actions</p> <p>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 stick to these centrally configured directories.</p>
[ECUC_BAS003]	<p>Generated Artefacts for oAW Actions</p> <p>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.</p>
[ECUC_BAS004]	<p>Perl Actions Required Invocation Parameters</p> <p>The following parameters shall be present for any Perl Action and be filled with a valid value:</p> <p>MODULE_NAME STEP_NAME</p> <p>The value for the parameter MODULE_NAME shall be the same as the first part of the BuildAction name</p> <p>The value for the parameter STEP_NAME shall be the same as the last part of the BuildAction name</p>
[ECUC_BAS005]	<p>oAW Actions Required Invocation Parameters</p> <p>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.</p>
[ECUC_BAS006]	<p>Setup Action Required Invocation Parameters</p> <p>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.</p>

Table 133 Code Layout and Comments

Id	Rule <i>Chapter 5.1 [PerlCoding_CodeLayout]</i>
[Perl_001]	<p>Sections of Perl files</p> <p>Every source file contains the following sections</p> <ul style="list-style-type: none"> ▶ a file header (with copyright information and a short description) ▶ the actual source code ▶ a file footer (with the history information from the SCM system)
[Perl_002]	<p>Comments</p> <p>Comments in the code shall be written in English</p>
[Perl_003]	<p>Comment of a subroutine</p> <p>Every function in the Perl source is preceded by a comment block, specifying a synopsis of the defined function</p>

Id	Rule <i>Chapter 5.1 [PerlCoding_CodeLayout]</i>
[Perl_004]	<p>Indentation and spacing style</p> <p>Use the following indentation style:</p> <ul style="list-style-type: none"> ▶ 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
[Perl_005]	<p>Line length</p> <p>The line length of 120 characters shall not be exceeded.</p>

Table 134 Naming Conventions

Id	Rule <i>Chapter 5.2 [PerlCoding_NamingConv]</i>
[Perl_006]	<p>Packages and filenames</p> <p>Class: NamingConvention</p> <p>Perl script associated files shall start with the module name and are written in camel case (e.g. mixed upper/lower case).</p>
[Perl_007]	<p>Constants</p> <p>Class: NamingConvention</p> <p>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.</p>
[Perl_008]	<p>Naming of global and local variables</p> <p>Class: NamingConvention</p> <ul style="list-style-type: none"> ▶ 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: <ul style="list-style-type: none"> – "_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).
[Perl_009]	<p>References</p> <p>Class: NamingConvention</p> <p>The names of reference variables are suffixed with</p> <ul style="list-style-type: none"> ▶ "_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
[Perl_010]	<p>Names of subroutines</p> <p>Class: NamingConvention</p> <p>Subroutines names are written in camel case, and start with an upper-case letter.</p>

Id	Rule <i>Chapter 5.2 [PerlCoding_NamingConv]</i>
[Perl_0101]	<p>File handles</p> <p>Class: NamingConvention File handles (typeglobs) are written in upper case letters.</p>
[Perl_011]	<p>Names for hashes and arrays</p> <p>Class: NamingConvention Name of arrays may be in plural and hashes in singular.</p>

Table 135 Coding Conventions

Id	Rule <i>Chapter 5.3 [PerlCoding_CodingConventions]</i>
[Perl_012]	<p>Function Calls The first thing to do in a function is to fetch its parameters</p>
[Perl_020]	<p>Reading Configuration Data of Other Modules 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".</p>
[Perl_021]	<p>Consistency of the Name of a Perl Module File and the Package Name within the Perl Module Class: NamingConvention The specified package name within a perl module file (.pm file) shall be set identical to the file name of the perl module file if the perl module file is used as processor in a build action.</p>

Table 136 Programming Tips

Id	Rule <i>Chapter 5.4</i>
[Perl_014]	<p>Defensive Programming Make a habit of defensive programming</p> <ul style="list-style-type: none"> ▶ 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.
[Perl_015]	<p>Make regular expressions readable</p> <ul style="list-style-type: none"> ▶ 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).
[Perl_016]	<p>Make dereferenciation to references readable</p> <ul style="list-style-type: none"> ▶ The usage of the arrow operator allows a more compact notation of dereferenciation. ▶ The storage of intermediate results in blocks makes complicated pointer constructs more readable.
[Perl_017]	<p>Handling of multi-line strings For the definition of multi-line strings should preferred here documents.</p>
[Perl_018]	<p>Local overwriting of global variables Avoid the overwriting of global variables and furthermore of Perl system variables.</p>

05 Table 137 Templates

Id	Rule <i>Chapter 5.5</i>
[Perl_019]	Removed, template file is available

10 Table 138 Rules for ApplicationDataType of Category Value

Id	Rule <i>Chapter 7.1.3.1.1 [ApplDataType_Val_Rule]</i>
[Data_003]	<p>Removed</p> <p>Status: removed</p>
[Data_004a]	<p>Where to Define ApplicationDataType</p> <p>DerivedFrom: ARPac_14</p> <p>DerivedFrom: CEL_061</p> <p>DerivedFrom: Blueprint_011</p> <ol style="list-style-type: none"> 1. An <i>ApplicationDataType</i> that is standardized by AUTOSAR shall be used whenever applicable. 2. An <i>ApplicationDataType</i> that is normalized as central element shall be used whenever applicable. 3. An <i>ApplicationDataType</i> that is already defined by another <i>SwComponentType</i> may be reused. For details refer to <i>Rule Static_0108 [A_StaticView]</i>. 4. Module/Component specific <i>ApplicationDataTypes</i> shall be defined in the context (name space) of the module / component (that means in component's / module's ArPackage, sub- ArPackage "ApplicationDataTypes").
[Data_091]	<p>Referencing a CompuMethod from an ApplicationDataType</p> <p>CompuMethod shall be referenced by an <i>ApplicationDataType</i> through <i>SwDataDefProps</i> using the attribute <COMPU-METHOD-REF>.</p>
[Data_114]	<p>Data type for Application Primitive Data Type</p> <p>Scope: DGS</p> <p>DerivedFrom: DP0050A-2</p> <p>Primitive ApplicationDataTypes shall be mapped to AUTOSAR specified <i>ImplementationDataTypes</i> (uint 8, sint 16 etc.) which are available as central elements.</p>
[Data_119]	<p>DataConstraint are mandatory for ApplicationDataType that are mapped to Float</p> <p>ApplicationDataType shall have a DataConstraint, if it is mapped to <i>ImplementationDataType</i> of type float.</p>
[Data_142]	<p>Definition of SwDataDefProps for ApplicationPrimitiveDataType</p> <p><i>SwDataDefProps</i> for an <i>ApplicationPrimitiveDataType</i> shall be defined as follows:</p> <ul style="list-style-type: none"> ▶ A <i>SwDataDefProps</i> shall contain a <i>SwCalibrationAccess</i>, <i>CompuMethod</i> and <i>DataConstraint</i>. ▶ A <i>SwDataDefProps</i> may contain <i>DisplayFormat</i>, <i>SwImplPolicy</i> and <i>SwRecordLayout</i>. ▶ Default value for <i>SwCalibrationAccess</i> shall be READ-WRITE and for <i>SwImplPolicy</i> shall be STANDARD.

50 Table 139 Rules for ApplicationDataType of Category Boolean

Id	Rule <i>Chapter 7.1.3.2.1 [ApplDataType_Boolean_Rule]</i>
[Data_155]	<p>CompuMethods referenced from ApplicationDataType of Category BOOLEAN</p> <p><i>ApplicationDataType</i> of Category BOOLEAN shall refer to <i>CompuMethods</i> of Category TEXTTABLE only</p>
[Data_173]	<p>Definition of SwDataDefProps for ApplicationPrimitiveDataType of Category Boolean</p> <p><i>SwDataDefProps</i> shall be a mandatory element for <i>ApplicationPrimitiveDataType</i> of Category Boolean as mandatory element similar to <i>Short-Name</i> and <i>Category</i>.</p> <p><i>SwDataDefProps</i> for an <i>ApplicationPrimitiveDataType</i> of category Boolean shall be defined as follows:</p> <ul style="list-style-type: none"> ▶ A <i>SwDataDefProps</i> shall contain a <i>SwCalibrationAccess</i>, <i>CompuMethod</i>. ▶ A <i>SwDataDefProps</i> may contain <i>DisplayFormat</i>, <i>SwImplPolicy</i> and <i>SwRecordLayout</i>.

Table 140 Rules for ApplicationDataType of Category String

Id	Rule Chapter 7.1.3.4.1 [ApplDataType_String_Rule]
[Data_130]	<p>Removed</p> <p>Status: removed</p>

Table 141 Rules for ApplicationDataType of Category Array

Id	Rule Chapter 7.1.4.1.1 [ApplDataType_Array_Rule]
[Data_140]	<p>Defining Array Size</p> <p><i>MaxNumberOfElements</i> describes the size of ARRAY, as such the attribute <i>MaxNumberOfElements</i> shall be defined for <i>ApplicationArrayType</i></p> <p><i>ArraySizeSemantics</i> should not be defined in the <i>ApplicationArrayType</i> (Could be defined in the <i>ImplementationDataType</i>)</p> <p><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.</p>
[Data_176]	<p>Defining LongNames, Annotations and Description of an Array</p> <p><i>LongName</i>, <i>Annotations</i> and <i>Descr</i> shall be defined at the parent level in case of simple Array and nested Array.</p> <p>In case of an Array & Structure combination, <i>LongNames</i>, shall be defined at Parent Element and may also be defined at <i>ArrayElement</i> level.</p>

Table 142 Rules for ApplicationDataType of Category Structure

Id	Rule Chapter 7.1.4.2.1 [ApplDataType_Struct_Rule]
[Data_123]	<p>Removed</p> <p>Status: removed</p>
[Data_141]	<p>Definition of SwDataDefProps at SubElement Level</p> <p>The <i>SwDataDefProps</i> of the <i>Elements</i> of an <i>ApplicationArrayType</i> or <i>ApplicationRecordType</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.</p> <p>The definition of <i>SwDataDefProps</i> at parent element shall contain <i>SwCalibrationAccess</i>.</p>

Table 143 Rules for ImplementationDataType

Id	Rule Chapter 7.1.5.1 [ImplDataType_Rule]
[Data_004b]	<p>Where to Define ImplementationDataType</p> <ol style="list-style-type: none"> Primitive Implementation data types (e.g. sint16) shall not be defined locally as they are always central and Standardized and delivered as Central Elements. 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 AUTOSAR Standardized implementation data types for these objects in a future version. For further details refer to Document "Central Elements" [A_CEL] Other Complex Implementation data types (e.g. Structures) are defined as Module/Component specific in BSWMD/SWCD. <i>ImplementationDataTypes</i> referencing TEXTTABLE CompuMethods (for enums) shall be provided as central elements if they are product line specific, or they shall be defined module specific.
[Data_050]	<p>Reuse of ImplementationDataTypes</p> <p>The reuse of <i>ImplementationDataTypes</i> is encouraged. Developer shall reuse the <i>ImplementationDataType</i> that are provided centrally or create module specific <i>ImplementationDataType</i>. <i>ImplementationDataType</i> defined for one specific module shall not be reused.</p>

Table 144 Rules for ImplementationDataType of Category Array

Id	Rule Chapter 7.1.5.3.1 [ImplDataType_Array]
[Data_139]	<p>Usage of ArraySize and ArraySizeSemantics <i>ImplementationDataTypes</i> shall have <i>ArraySize</i> defined in <i>ImplementationDataTypeElement</i> which defines the size of Array. <i>ArraySizeSemantics</i> shall be FIXED-SIZE.</p>

Table 145 Rules for DataTypeMappingSet

Id	Rule Chapter 7.1.6.1 [DataTypeMapping_Rule]
[Data_127]	<p>Where to Define DataTypeMappingSet DerivedFrom: CEL_064 <i>DataTypeMappingSet</i> may be defined centrally or decentral.</p> <ul style="list-style-type: none"> ▶ If the referenced <i>ApplicationDataType</i> and the referenced <i>ImplementationDataType</i> are defined centrally, then <i>DataTypeMappingSet</i> is also defined centrally. ▶ Mapping between <i>ApplicationDataType</i> and <i>ImplementationDataType</i> shall be defined locally (Component/ Module Specific Mapping) if the <i>ApplicationDataType</i> and <i>ImplementationDataType</i> are also defined locally.
[Data_129]	<p>Mapping Between ApplicationDataTypes and ImplementationDataTypes Every <i>ApplicationDataType</i> shall be mapped to an <i>ImplementationDataType</i> through a <i>DataTypeMapping</i>. For all <i>ApplicationDataTypes</i> defined by a component corresponding mapping to <i>ImplementationDataType</i> shall also be specified in the same component.</p>

Table 146 Rules for SwCalibAccess

Id	Rule Chapter 7.2.2.1.1 [SwCalibAccess_Rule]
[Data_040]	<p>Instantiating a ParameterDataPrototype with SwCalibrationAccess and SwImplPolicy A calibration parameter is instantiated with a <i>ParameterDataPrototype</i> class that aggregates a <i>SwDataDefProps</i> with properties</p> <ul style="list-style-type: none"> ▶ <i>swCalibrationAccess</i> = <i>readWrite</i> and <i>swImplPolicy</i> = <i>standard</i> when the <i>ParameterDataPrototype</i> shall be visible in the A2L file and calibratable (changeable). ▶ <i>swCalibrationAccess</i> = <i>readOnly</i> and <i>swImplPolicy</i> = <i>standard</i> when the <i>ParameterDataPrototype</i> shall be visible in the A2L file and not calibratable (changeable). ▶ <i>swCalibrationAccess</i> = <i>notAccessible</i> and <i>swImplPolicy</i> = <i>standard</i> when the <i>ParameterDataPrototype</i> shall not be written to A2L file at all.
[Data_143]	<p>Calibration Access for Measurements Measurement object shall only have READ-ONLY as Calibration Access</p>

Table 147 Rules for SwImplPolicy

Id	Rule Chapter 7.2.2.2.1 [SwImplPolicy_Rule]
[Data_170]	<p>Handling of SwImplPolicy attribute at ParameterDataPrototype DerivedFrom: TPS_SWCT_02000 <i>SwImplPolicy</i> shall not be defined for <i>ParameterDataPrototype</i></p>
[Data_171]	<p>Handling of SwImplPolicy attribute at VariableDataPrototype DerivedFrom: TPS_SWCT_02000 <i>SwImplPolicy</i> shall not be defined for <i>VariableDataPrototype</i></p>

Table 148 Rules for ParameterDataPrototype of Category Value

Id	Rule <i>Chapter 7.2.3.1.1 [ParmDataPrototype_Value]</i>
[Data_009]	Usage of Shared Calibration Parameters and Per Instance Calibration Parameters 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.
[Data_010]	Removed
[Data_011]	Sharing Calibration Parameters between instances of different "SwComponentType" <ul style="list-style-type: none"> ▶ For the <i>Calibration parameters</i> to be visible (shared) in other <i>SwComponentTypes</i>, a dedicated <i>ParameterSwComponentType</i> has to be used. ▶ The <i>ParameterSwComponentType</i> has no <i>InternalBehavior</i> and employs exclusively <i>PPortPrototypes</i> of type <i>ParameterInterface</i>. ▶ Every <i>SwComponentType</i> requiring access to shared Calibration Parameters will have an <i>RPortPrototype</i> typed by a <i>ParameterInterface</i>.
[Data_031]	Initial Value for ParameterDataPrototypes Every <i>ParameterDataPrototype</i> shall have an initial value specified . This value shall be an implementation based one.
[Data_051]	Reference to an ImplementationDataType <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 an <i>ImplementationDataType</i>
[Data_051a]	Reference to an ImplementationDataType Scope: BSW <i>ImplementationDataType</i> shall be referenced directly by a <i>DataPrototype</i> , if and only if the <i>DataPrototype</i> cannot be an A2L object. ¹⁴
[Data_174]	Categories of DataPrototype and corresponding ApplicationDataType shall be same A <i>DataPrototype</i> shall have same category as the <i>ApplicationDataType</i> to which it is referring.
[Data_178]	Usage of Constant Memorys are not allowed for specifying CalParams <i>RTE-Generator</i> shall not create any C-Code for <i>Constant Memorys</i> . As such neither in ASW nor in BSW <i>Constant Memorys</i> shall be used.

Table 149 Rules for VariableDataPrototype of Category Value

Id	Rule <i>Chapter 7.2.6.1.1 [VarDataPrototype_Value_Rule]</i>
[Data_179]	Possibility to defineVariableDataPrototypes in ASW and BSW <ul style="list-style-type: none"> ▶ In ASW, <i>VariableDataPrototypes</i> shall defined in the role of <i>EXPLICIT-INTER-RUNNABLE-VARIABLE</i> and <i>IMPLICIT-INTER-RUNNABLE-VARIABLE</i>. ▶ In BSW, <i>VariableDataPrototypes</i> shall be defined in the role of <i>Static Memorys</i>.

Table 150 Rules for SwDataDefPropsUsage

Id	Rule <i>Chapter 7.3.1 [SwDataDefProps_Rule]</i>
[Data_135]	Usage of Attributes of SwDataDefProps Properties of <i>SwDataDefProps</i> shall be defined according to table " <i>Usage of Attributes of SwDataDefProps</i> " .
[Data_136]	Not Used Attributes of SwDataDefProps Properties of <i>SwDataDefProps</i> listed in table " <i>Not Used Attributes of SwDataDefProps</i> " shall not be used.
[Data_137]	Usage of InstantiationDataDefProps <i>InstantiationDataDefProps</i> shall not be used.

¹⁴ A *DataPrototype* that appear in an A2L file is termed as A2L object.

Id	Rule <i>Chapter 7.3.1 [SwDataDefProps_Rule]</i>
[Data_175]	<p>Semantics of displayFormat</p> <p><i>displayFormat</i> shall be of form "<code>%d.pf</code>", where</p> <ul style="list-style-type: none"> ▶ <i>d</i> represents minimum overall number of digits. This includes flag and decimal point. ▶ <i>.</i> represents decimal point ▶ <i>p</i> represents the number of digits after decimal point ▶ <i>f</i> represents type character for float representation.

15 Table 151 Rules for *DataConstraint*

Id	Rule <i>Chapter 7.5.3 [DataConstr_Rule]</i>
[Data_157]	<p>Attributes which are allowed or mandated for DataConstraints</p> <ul style="list-style-type: none"> ▶ <i>DataConstraints</i> shall contain <i>ShortName</i> and <i>DataConstrRule</i> as mandatory attributes. ▶ <i>LongName</i> may be an optional attribute. ▶ <i>AdminData</i> should not be used.
[Data_158]	<p>Making a reference to DataConstraint</p> <p><i>DataConstraints</i> shall be referred from the <i>SwDataDefProps</i> of <i>ApplicationDataType</i> only.</p>
[Data_159]	<p>Semantics of DataConstrRule</p> <p>DerivedFrom: <i>constr_2561</i></p> <p>A <i>DataConstrRule</i> shall contain <i>physConstr</i> and <i>constrLevel</i>. The <i>constrLevel</i> shall always set to "0", representing "hard limits"¹⁵</p>
[Data_160]	<p>Attributes that composes PhysConstr</p> <p>A <i>PhysConstr</i> shall contain an <i>UpperLimit</i>, a <i>LowerLimit</i>.</p> <p>A <i>PhysConstr</i> should make a reference to <i>Unit</i>. It shall only be omitted if consistent usage of the <i>PhysConstr</i> can be guaranteed, e.g. in case of generated code.</p>
[Data_161]	<p>Where to Define DataConstraint</p> <p>DerivedFrom: <i>CEL_063</i></p> <p>Developers shall use <i>DataConstraint</i> defined as part of central elements if it is available.</p>
[Data_162]	<p>Reuse of DataConstraints</p> <p>Existing <i>DataConstraints</i> shall be reused if possible.</p>
[Data_169]	<p>Allowed type ofDataConstraint</p> <p><i>DataConstraint</i> of the type <i>PhysConstr</i> is the only allowed type. These constraint shall take value of type float.</p>
[Data_172]	<p>Naming convention for DataConstraints</p> <p>Class: <i>NamingConvention</i></p> <p>The naming convention for <i>DataConstraint</i> shall follow the following syntax: <i>Constr_[Referenced Unit]_Limits</i> where " <i>Constr</i>" is the prefix for <i>DataConstraint</i> followed by the short name of the unit referenced and finally limits in alphanumeric form.</p>

¹⁵ Hard limits imply that calibration tools will not allow to violate these constraints.

Table 152 Rules for CompuMethods

Id	Rule <i>Chapter 7.6.1 [CompuMethod_Rule]</i>
[Data_090]	<p>When to use which CompuMethod Category</p> <p>Developer shall use:</p> <ul style="list-style-type: none"> ▶ <i>CompuMethod</i> of Category <i>IDENTICAL</i> if identical (one to one) relation exist between physical and internal values. ▶ <i>CompuMethod</i> of Category <i>LINEAR</i> if linear relation exists between physical and internal values and if identical relation is not applicable. ▶ <i>CompuMethod</i> of Category <i>RAT_FUNC</i> if numerical relation exists between physical and internal values and linear relation is not applicable. ▶ <i>CompuMethod</i> of Category <i>TEXTTABLE</i> if text values are to be assigned to the internal values.
[Data_110]	<p>Unit reference by different categories of CompuMethods</p> <p>RelatedTo: <i>constr_1175</i> in TPS SWCT "Software Component Template"</p> <ol style="list-style-type: none"> 1. <i>CompuMethod</i> of Category <i>IDENTICAL</i>, <i>LINEAR</i>, <i>RAT_FUNC</i> shall refer to a <i>Unit</i>. If there is no <i>Unit</i> available, then the <i>Unit</i> with <i>ShortName</i> "NoUnit" shall be referenced. 2. <i>CompuMethod</i> of Category <i>TEXTTABLE</i> shall also refer to a <i>Unit</i> but it will be typically the <i>Unit</i> named "NoUnit".
[Data_111]	<p>Removed</p> <p>Status: removed</p>
[Data_113]	<p>Removed</p> <p>Status: removed</p>
[Data_116]	<p>Removed</p> <p>Status: removed</p>
[Data_124]	<p>Where to DefineCompuMethod</p> <p>DerivedFrom: <i>CEL_051</i></p> <p>DerivedFrom: <i>CEL_052</i></p> <p><i>CompuMethods</i> should be coming from central elements. If a suitable <i>CompuMethod</i> exists as central element it shall be used instead of creating a local one. Arithmetic <i>CompuMethods</i> (These includes Categories <i>IDENTICAL</i>, <i>LINEAR</i>, <i>RAT_FUNC</i>) should be centralized. Textual <i>CompuMethods</i> (of Category <i>TEXTTABLE</i>) should be centralized if they are reusable.</p>
[Data_132]	<p>Allowed Categories forCompuMethod</p> <p>Not all the categories defined by AUTOSAR are supported in UBK yet. Developer shall use the categories <i>IDENTICAL</i>, <i>LINEAR</i>, <i>RAT_FUNC</i>, <i>TEXTTABLE</i> for their developmental activities.</p>
[Data_133]	<p>Removed</p> <p>Status: removed</p>
[Data_147]	<p>Attributes which are allowed or mandated for all categories of CompuMethods</p> <p>RelatedTo: <i>constr_1021</i></p> <ul style="list-style-type: none"> ▶ Each <i>CompuMethod</i> shall have a <i>ShortName</i>. ▶ Each <i>CompuMethod</i> shall have a <i>Category</i>. ▶ Each <i>CompuMethod</i> should specify conversion directions (depending on <i>Category</i>): <ul style="list-style-type: none"> – <i>compuPhysToInternal</i> – <i>compuInternalToPhys</i> ▶ Each <i>CompuMethod</i> may specify <i>LongName</i>. ▶ Each <i>CompuMethod</i> shall specify <i>Unit</i> depending upon the <i>Category</i> of <i>CompuMethod</i>. ▶ Each <i>CompuMethod</i> may specify <i>Desc</i>, <i>AdminData</i>, <i>Introduction</i>, <i>Annotation</i> ▶ Each <i>CompuMethod</i> should not specify <i>DisplayFormat</i>.

Id	Rule <i>Chapter 7.6.1 [CompuMethod_Rule]</i>
[Data_167]	<p>Using a fractional number as part of a ShortName</p> <p>Class: NamingConvention</p> <p>Whenever a fractional number is to be used as part of a ShortName it shall be done using the following format:</p> $m\{number\} (p\{number\}) ? (Em\{number\}) ?$ <p>where</p> <ul style="list-style-type: none"> ▶ number: [1-9] ([0-9]) * ▶ "p" represents decimal point (".") ▶ "E" represents exponent ▶ "m" represents a minus "-"

Table 153 Rules for Identical CompuMethod

Id	Rule <i>Chapter 7.6.2.1 [Rules_IdenticalCompu]</i>
[Data_146]	<p>Attributes which are allowed or mandated for CompuMethod of Category IDENTICAL</p> <ul style="list-style-type: none"> ▶ A CompuMethod of Category IDENTICAL shall have a ShortName ▶ Category shall be set to IDENTICAL ▶ A CompuMethod of Category IDENTICAL shall reference a Unit. ▶ A CompuMethod of Category IDENTICAL should not specify physConstr or internalConstr ▶ A CompuMethod of Category IDENTICAL shall not contain compuScales
[Data_163]	<p>Naming Convention for CompuMethod of Category IDENTICAL</p> <p>Class: NamingConvention</p> <p>The name of a CompuMethod of category IDENTICAL shall follow the syntax:</p> $<\text{shortName of the Unit}>+\text{Identcl}$ <p>where</p> <ul style="list-style-type: none"> ▶ <shortName of the Unit>: is the short name of the referenced Unit ▶ Identcl: Keyword to represent Identical CompuMethod.

Table 154 Rules for LINEAR CompuMethod

Id	Rule <i>Chapter 7.6.3.1 [Rules_LinearCompu]</i>
[Data_092]	<p>Floating numbers to be used for Numerical CompuMethod</p> <p>The values within compuScales of CompuMethods shall be entered as float. Float numbers shall always be specified with decimal point ".".</p> <p>e.g. "5.0" instead of "5"</p>
[Data_115]	<p>Conversion Direction for CompuMethod of Category LINEAR</p> <p>CompuMethod of category LINEAR shall be defined using compuPhysToInternal.</p>
[Data_148]	<p>Restrictions on compuScale in LinearCompuMethod</p> <p>A LINEAR CompuMethod shall specify one compuScale. Its compuNumerator shall contain two Vs, the first one specifying the offset and the second one specifying the linear conversion factor. Its compuDenominator shall be set to 1.0.</p>
[Data_149]	<p>Attributes which are allowed or mandated for LinearCompuMethod</p> <p>Attributes for a LinearCompuMethod:</p> <ul style="list-style-type: none"> ▶ A LinearCompuMethod shall have ShortName and a reference to an Unit. ▶ Category shall be set to LINEAR ▶ A Linear CompuMethod may specify a LongName. ▶ Attributes physConstr and internalConstr shall not be specified.

Id	Rule <i>Chapter 7.6.3.1 [Rules_LinearCompu]</i>
[Data_164]	<p>Naming Convention for LinearCompuMethod</p> <p>Class: NamingConvention</p> <p>The name of a <i>CompuMethod</i> of category <i>LINEAR</i> shall follow the syntax: <code><shortName of Unit>+Lnr+<sequenceNumber></code></p> <p>where</p> <ul style="list-style-type: none"> ▶ <code><shortName of Unit></code>: is the short name of the referenced <i>Unit</i> ▶ <code>Lnr</code>: is the keyword used to represent Linear CompuMethod ▶ <code>{sequenceNumber}</code>: A positive integer to make the name unique in the given name space.

Table 155 Rules for RatFunc CompuMethod

Id	Rule <i>Chapter 7.6.4.1 [Rules_RatFunc]</i>
[Data_117]	<p>Removed</p> <p>Status: removed</p>
[Data_150]	<p>Restriction on having the number of V elements</p> <p>A <i>RAT_FUNC CompuMethod</i> shall specify one <i>compuScale</i>. Its <i>compuNumerator</i> shall contain one <i>V</i> element and its <i>compuDenominator</i> shall contain two <i>V</i> elements.</p>
[Data_156]	<p>Attributes which are allowed or mandated for RatFuncCompuMethod</p> <p>Attributes for RatFunc CompuMethod are:</p> <ul style="list-style-type: none"> ▶ A <i>RAT_FUNC CompuMethod</i> shall specify a <i>ShortName</i> and a reference to <i>Unit</i>. ▶ <i>Category</i> shall be set to <i>RAT_FUNC</i> ▶ <i>compuPhysToInternal</i> is the only conversion direction allowed. ▶ A <i>RAT_FUNC CompuMethod</i> shall specify interval of the values (Lower-Limit Interval and Upper--Limit Interval) that are specified for <i>compuNumerator</i> and <i>compuDenominator</i>.
[Data_165]	<p>Naming Convention for CompuMethod of category RAT_FUNC</p> <p>Class: NamingConvention</p> <p>The name of a <i>CompuMethod</i> of category <i>RAT_FUNC</i> shall follow the syntax: <code>Rat({_n{Nominator}})*({_d{Denominator}})*{_Unit}({_Counter})?</code></p> <p>where</p> <ul style="list-style-type: none"> ▶ <code>{Nominator}</code>: A fractional number with decimal point which correlates to the nominator, according to their order. It shall be converted to a string as defined in [Data_167]. ▶ <code>{Denominator}</code>: A fractional number with decimal point which correlates to the denominator, according to their order. It shall be converted to a string as defined in [Data_167]. ▶ <code>{Unit}</code>: is the short name of the referenced <i>Unit</i> ▶ <code>{Counter}</code>: A positive integer to make the name unique in the given name space. <p>For the representation of a fractional number refer to rule [Data_167]</p>

Table 156 Rules for TextTable CompuMethod

Id	Rule <i>Chapter 7.6.5.2 [Rules_Texttable]</i>
[Data_118]	<p>Attributes which are allowed or mandated for TEXTTABLE CompuMethod</p> <p><i>compuInternalToPhys</i> is the only allowed direction of the conversion for <i>CompuMethod</i> of category <i>TEXTTABLE</i>.</p> <p><i>compuInternalToPhys</i> shall contain <i>compuScales</i> consisting of <i>UpperLimit</i> and <i>LowerLimit</i>, both with <i>INTERVAL-TYPE="CLOSED"</i> and both shall specify the same numerical internal value.</p> <p>The result of conversion shall be placed in <i>VT</i>. All <i>VTs</i> of a single <i>CompuMethod</i> shall be unique. <i>CompuDefaultValue</i> should not be used.</p>

Id	Rule <i>Chapter 7.6.5.2 [Rules_Texttable]</i>
[Data_145]	<p>Removed</p> <p>Status: removed</p>
[Data_166]	<p>Naming Convention for CompuMethod of category TEXTTABLE</p> <p>Class: NamingConvention</p> <p>The name of a <i>CompuMethod</i> of category <i>TEXTTABLE</i> shall follow the syntax: <code>Txt (_{TextValue}) * (_Counter) ?</code> where</p> <ul style="list-style-type: none"> ▶ {TextValue}: The text Values according their order. ▶ {Counter}: A positive integer to make the name unique in the given name space.
[Data_181]	<p>Vt shall either be a valid C-Identifier or Symbol shall be defined</p> <p>Content of <i>Vt</i> elements shall follow C-Identifier syntax.</p> <p>If <i>Vt</i> is explicitly designed to have no C-Identifier syntax developer shall define <i>Symbol</i> additionally. In this case <i>Symbol</i> shall have C-identifier syntax.</p>

Table 157 Rules

Id	Rule <i>Chapter 7.8.1 [SwAddrMethods_Rule]</i>
[Data_122]	<p>Removed</p> <p>Status: removed</p>

Table 158 Rules for Units and PhysicalDimension

Id	Rule <i>Chapter 7.9.3 [Units_PhysDim_Rule]</i>
[Data_112]	<p>Attributes which are allowed or mandated for for Unit</p> <ol style="list-style-type: none"> 1. Every <i>Unit</i> shall contain <i>ShortName</i>. 2. Every <i>Unit</i> shall contain <i>factorSiToUnit</i>. 3. Every <i>Unit</i> shall contain <i>OffsetSiToUnit</i>. 4. Every <i>Unit</i> shall contain reference to <i>PhysicalDimension</i>. 5. Every <i>Unit</i> shall contain <i>DisplayName</i>.
[Data_121]	<p>Removed</p> <p>Status: removed</p>
[Data_125]	<p>Where to Define Units and Physical Dimensions</p> <p>DerivedFrom: <i>CEL_001</i></p> <p>DerivedFrom: <i>CEL_031</i></p> <p><i>Units</i> and Physical Dimensions shall be defined as central elements. Developers shall only make a reference to these <i>Units</i> and Physical Dimension and shall not create them.</p>
[Data_126]	<p>Removed</p> <p>Status: removed</p>
[Data_152]	<p>Making a reference to Units</p> <p>Units shall be referenced only from <i>CompuMethods</i> and <i>PhysConstrs</i> and not from <i>SwDataDefProps</i></p>
[Data_153]	<p>Accuracy of factorSiToUnit and OffsetSiToUnit</p> <p><i>factorSiToUnit</i> and <i>OffsetSiToUnit</i> 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.</p>
[Data_154]	<p>PhysicalDimension for Absolute and Relative Units</p> <p>Distinct <i>PhysicalDimensions</i> shall be defined for absolute and relative <i>Units</i> that have a non-zero value for <i>offsetSiToUnit</i>.</p>

Id	Rule <i>Chapter 7.9.3 [Units_PhysDim_Rule]</i>
[Data_168]	<p>Naming Convention for Units</p> <p>Class: NamingConvention</p> <p>The name of a <i>Unit</i> shall be derived from its content of <i>DisplayName</i> as</p> <ul style="list-style-type: none"> ▶ If the unit is a formula containing "x to the power of 2" the short name contain "Sqrd". ▶ If the unit is a formula containing "x to the power of 3" the short name shall contain "Cubd". ▶ If the unit is a formula containing "x to the power of number >3" the short name shall contain To-PwrOf <number>. ▶ If the unit is a formula containing the division the short name shall contain "Per".
[Data_177]	<p>Optional/Mandatory Exponent values in PhysicalDimension</p> <p>In a <i>PhysicalDimension</i>, exponent values may not be defined if a default value is available in AUTOSAR. However there are <i>PhysicalDimensions</i> for which default exponent values are not defined in AUTOSAR. In this case user shall specify the value for the Exponent.</p>
[Data_180]	<p>Naming Convention for PhysicalDimensions</p> <p>Class: NamingConvention</p> <p>The name of a <i>PhysicalDimension</i> shall be of the form:</p> <p><BaseDimensionKeyword1[-]{exp}><BaseDimensionKeyword2[-]{exp}>.....n where,</p> <p>BaseDimensionKeyword: BaseDimension Key word represents the key word representing each of base quantities like length, mass, time etc. These keywords are standardised by Keyword data base.</p> <p>exp is the exponent of the dimension. Exponent can be negative as well.</p>

Table 159 Rules forUnitGroup

Id	Rule <i>Chapter 7.10.1 [UnitGroup_Rule]</i>
[Data_151]	<p>Usage of UnitGroups</p> <p><i>UnitGroups</i> shall not be used by developers as they are not supported by tool chain currently.</p>

Table 160 Common Aspects of BSWMD

Id	Rule <i>Chapter 8.1</i>
[BSWMD_Common_001]	<p>BSWMD Standard File Header.</p> <p>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.</p>
[BSWMD_Common_002]	<p>Headline of a BSWMD ARXML File</p> <p>Each BSWMD ARXML file shall start with a headline containing the definition of the used AUTOSAR schema.</p>
[BSWMD_Common_003]	<p>ARPackage Hierarchy within BSWMD Files</p> <p>BSWMD files shall be conform to the ARPackage structure. That means that</p> <ul style="list-style-type: none"> ▶ 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. <p>The <ModulePrefix> represents the module prefix of the BSW module (but without a VendorId and a VendorApilnfix).</p> <p>In the BSWMD file the ARPackage structure shall be derived from the top level hierarchy.</p>
[BSWMD_Common_004]	<p>Comments Within BSWMD Files</p> <p>Within a BSWMD file comments may be set using the comment marker <!-- ... -->.</p>
[BSWMD_Common_005]	<p>Order of Tag Sequences Within BSWMD Files</p> <p>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.</p>

Table 161 BswModuleDescription

Id	Rule Chapter 8.2.1.1
[BSWMD_Module-Desc_001]	<p>ShortName of ARPackage Child Element BswModuleDescription</p> <p>Class: NamingConvention</p> <p>DerivedFrom: [ARpac_14] p. 397</p> <p>The ShortName of the ARPackage for the <i>BswModuleDescription</i> shall be set to "BswModuleDescriptions".</p>
[BSWMD_Module-Desc_002]	<p>ShortName of the BswModuleDescription</p> <p>Class: NamingConvention</p> <p>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).</p>
[BSWMD_Module-Desc_003]	<p>LongName of the BswModuleDescription</p> <p>Class: NamingConvention</p> <p>The LongName of the <i>BswModuleDescription</i> shall contain a meaningful description.</p>
[BSWMD_Module-Desc_004]	<p>Category of BswModuleDescription</p> <p>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> (default), <i>BSW_CLUSTER</i> (avoid this setting) or <i>LIBRARY</i>.</p>
[BSWMD_Module-Desc_005]	<p>BSW Module Identifier</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>Within the <i>BswModuleDescription</i> a module identifier shall be set within the tag <MODULE-ID>.</p>

Table 162 BswInternalBehavior

Id	Rule Chapter 8.2.1.2
[BSWMD_IntBehav_001]	<p>ShortName of the BswInternalBehavior</p> <p>Class: NamingConvention</p> <p>The ShortName of the <i>BswInternalBehavior</i> shall be set to "BswInternalBehavior".</p>
[BSWMD_IntBehav_002]	<p>Reference to DataTypeMapping</p> <p>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:</p> <ul style="list-style-type: none"> ▶ The <i>DataTypeMapping</i> shall be set using the tag <DATA-TYPE-MAPPING-REF> containing the destination type "DATA-TYPE-MAPPING-SET" ▶ The reference shall contain a correctly and fully specified ARPackage path to the <i>DataTypeMapping</i> ▶ The <i>DataTypeMapping</i> shall be enclosed from the tags <DATA-TYPE-MAPPING-REFS>

Table 163 BswImplementation

Id	Rule Chapter 8.2.1.3
[BSWMD_Impl_001]	<p>ShortName of the ARPackage Child Element BswImplementation</p> <p>Class: NamingConvention</p> <p>DerivedFrom: [ARpac_14] p. 397</p> <p>The ShortName of the ARPackage for the <i>BswImplementation</i> shall be set to "BswImplementations".</p>
[BSWMD_Impl_002]	<p>ShortName of the BswImplementation</p> <p>Class: NamingConvention</p> <p>The ShortName of the <i>BswImplementation</i> shall be identical to the module prefix of the BSW module.</p>

Id	Rule Chapter 8.2.1.3
[BSWMDImpl-003]	<p>Specification of a VendorId and a VendorApiInfix</p> <p>Scope: Only relevant for BSW driver modules which are based on AUTOSAR specifications and are providing multiple instances (number of implementation > 1)</p> <p>A vendor identification shall be specified with tag <VENDOR-ID>.</p> <p>Additionally a specific name shall be specified as VendorApiInfix using the tag <VENDOR-API-IN-FIX>.</p>
[BSWMDImpl-004]	<p>Provision of a Code Descriptor</p> <p>For each BswImplementation a code descriptor shall be provided. The following ARXML snippet shall be used:</p> <pre data-bbox="343 595 970 900"><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></pre>
[BSWMDImpl-005]	<p>Documentation of Programming Language</p> <p>Within the <i>BswImplementation</i> the programming language shall be specified. The related tag <PROGRAMMING-LANGUAGE> shall be set to "C".</p>
[BSWMDImpl-006]	<p>Provision of Software Version Number</p> <p>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.</p> <p>The software version number shall be updated directly before a new version of the software is released.</p>
[BSWMDImpl-007]	<p>Provision of AUTOSAR Release Version Number</p> <p>Scope: Software based on an AUTOSAR SWS</p> <p>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.</p> <p>The AUTOSAR version information shall be updated when the module supports a new version of an AUTOSAR specification.</p>
[BSWMDImpl-008]	<p>Reference to BswInternalBehavior</p> <p>A reference to the <i>BswInternalBehavior</i> (with a correctly and fully specified ARPackage path) shall be specified within the <i>BswImplementation</i> using the tag <BEHAVIOR-REF> containing the destination type "BSW-INTERNAL-BEHAVIOR".</p>

Table 164 Splitting of BSWMD Files

Id	Rule Chapter 8.2.2
[BSWMDSplit-001]	<p>Main Rule of Splitting of BSWMD Files</p> <p>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).</p>

Id	Rule <i>Chapter 8.2.2</i>
[BSWMD_Split_002]	<p>Possibility to Split BswInternalBehavior from BswModuleDescription</p> <p>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.</p> <p>If a split is done it shall be ensured that the structure of the corresponding BSWMD files is correct:</p> <ul style="list-style-type: none"> ▶ 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 any parts of the <i>BswInternalBehavior</i>. ▶ 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 <i>ShortName</i> of the <i>BswModuleDescription</i> shall be repeated.
[BSWMD_Split_003]	<p>BswInternalBehavior is not Splitable</p> <p>It is not allowed to split the <i>BswInternalBehavior</i> into different files. All attributes and sub-elements of the <i>BswInternalBehavior</i> shall be only specified in a single BSWMD file.</p>
[BSWMD_Split_004]	<p>Splitting of BswImplementation to a Single BSWMD File</p> <p>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.</p>
[BSWMD_Split_005]	<p>Contents of BswImplementation are not Splitable</p> <p>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.</p>
[BSWMD_Split_006]	<p>Splitting of Other BSWMD Specific ARPackages to Single BSWMD Files</p> <p>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.</p>

Table 165 Definition of Measurement Variables and Measurement Points

Id	Rule <i>Chapter 8.3.1.1</i>
[BSWMD_MCSupport_001]	<p>Definition of Measurement Variables in BSWMD</p> <p>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.</p>
[BSWMD_MCSupport_002]	<p>Definition of Measurement Variables in C Files</p> <p>The representation of measurement variables or measurement points 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 the definitions in the module BSWMD file (regarding data type and name of the measurement variable). The measurement variable or measurement points shall be defined as global variable and not as static variable.</p>
[BSWMD_MCSupport_004]	<p>Naming Convention for Measurement Variables</p> <p>Class: NamingConvention</p> <p>The ShortName of a measurement variable shall be conform to the following convention: <ComponentPrefix>_<pp><DescriptiveText>.</p>
[BSWMD_MCSupport_008]	<p>Naming Convention for Measurement Points</p> <p>Class: NamingConvention</p> <p>The ShortName of a measurement point shall be conform to the following convention: <ComponentPrefix>_<pp><DescriptiveText>_MP.</p>

Table 166 Definition of Calibration Parameters

Id	Rule <i>Chapter 8.3.1.2</i>
[BSWMD_MCSup-port_003]	<p>Definition of Calibration Parameters in BSWMD</p> <p>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.</p> <p>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.</p>
[BSWMD_MCSup-port_005]	<p>Naming Convention for Calibration Parameters</p> <p>Class: NamingConvention</p> <p>The ShortName of a calibration parameter shall be conform to the following naming convention: <ComponentPrefix>_<pp><DescriptiveText>_<EX></p>
[BSWMD_MCSup-port_006]	<p>Accessing a Calibration Parameter</p> <p>Class: NamingConvention</p> <p>To access a calibration parameter the following interface shall be used: SchM_CData_<ComponentPrefix>[_<vi>_<ai>]<Name>().</p>
[BSWMD_MCSup-port_007]	<p>Consideration of a Neutral Behavior of Calibration Related BSW Modules in Case of Development</p> <p>Scope: BSW modules which are relevant for calibration</p> <p>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:</p> <ul style="list-style-type: none"> ▶ New functionalities should be encapsulated with a new calibration parameter to be able to switch off 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.

Table 167 Mandatory Elements for Scheduling

Id	Rule <i>Chapter 8.3.2.1</i>
[BSWMD_Schedule_001]	<p>Mandatory Elements for the Scheduling Use Case</p> <p>To be able to schedule a BSW function the following elements shall be specified and related to each other: <i>BswModuleEntry</i>, <i>BswSchedulableEntity</i>, <i>BswEvent</i>.</p>
[BSWMD_Schedule_002]	<p>Dependency Between BswEvent and BswSchedulableEntity</p> <p>If a BSW function has to be scheduled in different tasks, or is called based on different conditions every time, a particular <i>BswEvent</i> shall be defined. The dependency between a <i>BswEvent</i> and a <i>BswSchedulableEntity</i> is a 'n to 1' relationship.</p>

Table 168 Mapping Between BSW and a Corresponding SWC

Id	Rule <i>Chapter 8.3.2.3</i>
[BSWMD_SWC_-001]	<p>Definition of SwcBswRunnableMapping</p> <p>For each <i>SwcBswRunnable</i> Mapping, a SwcBswRunnableMappings element shall be defined, a reference to a <i>BswSchedulableEntity</i> shall be defined in BswEntityRef and a reference to the Runnable Entity shall be defined in a SwcRunnableRef.</p> <p>A reference to a Bsw Internal Behavior shall be defined in BswBehaviorRef.</p> <p>And a reference to the Swc Internal Behavior shall be defined in a SwcRunnableRef.</p> <p>All these element shall be defined in a SwcBswMapping.</p> <p>BswModuleEntry shall be defined. Mapping between BSW and SWC shall be defined with the SwcBswMapping element in an ARXML file according to AUTOSAR specifications.</p> <p>The Runnable mappings shall be defined in a SwcBswRunnableMappings.</p>

Id	Rule <i>Chapter 8.3.2.3</i>
[BSWMD_SWC_002]	<p>Referencing a SwcBswMapping in a BSWMD ARXML file 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.</p>
[BSWMD_SWC_003]	<p>Definition of SynchronizedModeGroups between SWC and BSW 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.</p>
[BSWMD_SWC_004]	<p>Definition of SynchronizedTriggers between SWC and BSW 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.</p>

Table 169 Common Attributes of a BswModuleEntry

Id	Rule <i>Chapter 8.3.4.1.1</i>
[BSWMD_Entry_001]	<p>ShortName of the ARPackage Kind Element BswModuleEntry Class: NamingConvention DerivedFrom: [ARpac_14] p. 397 The ShortName of the ArPackage for the <i>BswModuleEntry</i> shall be set to "BswModuleEntries".</p>
[BSWMD_Entry_002]	<p>ShortName of the BswModuleEntry Class: NamingConvention The ShortName of the <i>BswModuleEntry</i> shall be set identical to the name of the C-function (but without a VendorId and a VendorApInfix).</p>
[BSWMD_Entry_003]	<p>Serviceld of the BswModuleEntry 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.</p>
[BSWMD_Entry_004]	<p>Reentrancy of a BswModuleEntry The tag <IS-REENTRANT> shall be used to describe the reentrancy of the specified service. The following two settings are possible: <i>true</i> or <i>false</i></p>
[BSWMD_Entry_005]	<p>Synchrony of a BswModuleEntry The tag <IS-SYNCHRONOUS> shall be used to describe the synchrony of the specified service. The following two settings are possible: <i>true</i> or <i>false</i></p>
[BSWMD_Entry_006]	<p>Call Type of the BswModuleEntry With the tag <CALL-TYPE> the type of call associated with this specified service shall be set. The following settings are possible: <i>CALLBACK</i>, <i>INTERRUPT</i>, <i>REGULAR</i> or <i>SCHEDULED</i>.</p>
[BSWMD_Entry_007]	<p>Execution Context of the BswModuleEntry With the tag <EXECUTION-CONTEXT> the execution context required or guaranteed for the call associated with this specified service shall be set. The following settings are possible: <i>HOOK</i>, <i>INTERRUPT-CAT-1</i>, <i>INTERRUPT-CAT-2</i>, <i>TASK</i> or <i>UNSPECIFIED</i>.</p>

Id	Rule <i>Chapter 8.3.4.1.1</i>
[BSWMD_Entry_008]	<p>SwServiceImplPolicy of the BswModuleEntry With the tag <SW-SERVICE-IMPL-POLICY> the implementation policy of the specified service shall be documented. The following settings are possible: <i>INLINE</i>, <i>INLINE-CONDITIONAL</i>, <i>MACRO</i> or <i>STANDARD</i>.</p>
[BSWMD_Entry_009]	<p>LongName of the BswModuleEntry The tag item <LONG-NAME> shall be used to specify a headline as explanation of the <i>BswModuleEntry</i>.</p>
[BSWMD_Entry_010]	<p>Description of the BswModuleEntry The tag item <DESC> shall be used to specify a short description of a <i>BswModuleEntry</i>.</p>
[BSWMD_Entry_011]	<p>Introduction of the BswModuleEntry The tag item <INTRODUCTION> should be used to specify a more detailed explanation of a <i>BswModuleEntry</i>.</p>

Table 170 Definition of a Return Value and Arguments of a BswModuleEntry

Id	Rule <i>Chapter 8.3.4.1.2</i>
[BSWMD_Entry-RetArg_001]	<p>ShortName of a Return Value of a BswModuleEntry Class: NamingConvention The ShortName of a return value of a <i>BswModuleEntry</i> shall be set to 'ReturnValue'.</p>
[BSWMD_Entry-RetArg_002]	<p>ShortName of an Argument of a BswModuleEntry Class: NamingConvention 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.</p>
[BSWMD_Entry-RetArg_003]	<p>Category of a Return Value or an Argument of a BswModuleEntry With the tag <CATEGORY> the kind of the return value or the argument shall be specified (value, data pointer, function pointer). The following categories are possible: <i>TYPE_REFERENCE</i>, <i>DATA_REFERENCE</i> or <i>FUNCTION_REFERENCE</i>. Dependent on a specific category appropriate SwDataDefProps shall be specified.</p>
[BSWMD_Entry-RetArg_004]	<p>Direction of a Return Type or an Argument of a BswModuleEntry 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. The 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.</p>
[BSWMD_Entry-RetArg_005]	<p>Definition of a Value as Return Value or Argument of a BswModuleEntry To specify a value as return value or argument of a <i>BswModuleEntry</i> the following settings shall be made:</p> <ul style="list-style-type: none"> ▶ A reference to an ImplementationDataType using the tag <IMPLEMENTATION-DATA-TYPE-REF> shall be set within the <i>SwDataDefProps</i> tag structure ▶ The destination type of the <i>ImplementationDataTypeRef</i> shall be set to 'IMPLEMENTATION-DATA-TYPE' ▶ A correctly and complete specified path to a centrally or locally defined <i>ImplementationDataType</i> shall be set ▶ The category of the return value or argument shall be set to 'TYPE-REFERENCE' (conform to rule [BSWMD_EntryRetArg_003])

Id	Rule Chapter 8.3.4.1.2
[BSWMD_Entry- RetArg_006]	<p>Definition of a Data Pointer as Return Value or Argument of a BswModuleEntry</p> <p>To specify a data pointer as return value or argument of a <i>BswModuleEntry</i> the following settings shall be made:</p> <ul style="list-style-type: none"> ▶ A reference to an <i>ImplementationDataType</i> using the tag <IMPLEMENTATION-DATA-TYPE-REF> 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 ▶ The destination type of the <i>ImplementationDataTypeRef</i> shall be set to 'IMPLEMENTATION-DATA-TYPE' ▶ A correctly and complete path to a centrally or locally defined <i>ImplementationDataType</i> shall be set ▶ The category of the return value or argument shall be set to 'DATA-REFERENCE' (conform to rule [BSWMD_EntryRetArg_003])
[BSWMD_Entry- RetArg_007]	<p>Definition of a Void Pointer as Return Value or Argument of a BswModuleEntry</p> <p>To specify a void pointer as return value or argument of a <i>BswModuleEntry</i> the following settings shall be made:</p> <ul style="list-style-type: none"> ▶ A reference to a <i>SwBaseType</i> using the tag <BASE-TYPE-REF> 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 ▶ The destination type of the <i>ImplementationDataTypeRef</i> 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])
[BSWMD_Entry- RetArg_008]	<p>Definition of a Function Pointer as Return Value or Argument of a BswModuleEntry</p> <p>To specify a function pointer as return value or argument of a <i>BswModuleEntry</i> the following settings shall be made:</p> <ul style="list-style-type: none"> ▶ A reference to a <i>BswModuleEntry</i> using the tag <FUNCTION-POINTER-SIGNATURE-REF> shall be set within <i>SwPointerTargetProps</i> which are part of the <i>SwDataDefProps</i> tag structure of the return value or argument definition ▶ The destination type of the <i>FunctionPointerSignatureRef</i> shall be set to 'BSW-MODULE-ENTRY' ▶ A correctly and complete path to a <i>BswModuleEntry</i> shall be set ▶ The category of the return value or argument shall be set to 'FUNCTION-REFERENCE' (conform to rule [BSWMD_EntryRetArg_003])
[BSWMD_Entry- RetArg_009]	<p>Definition of an Implementation Policy for Return Values and Arguments of a BswModuleEntry</p> <p>Within the <i>SwDataDefProps</i> an implementation policy shall be specified using the tag <SW-IMPL-POLICY>.</p> <p>The following values are possible: STANDARD or CONST.</p> <p>The tag <SW-IMPL-POLICY> is mandatory on each level of <i>SwDataDefProps</i>.</p> <p>The implementation policy 'CONST' is only relevant in the 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>Definition of an Additional Native Type Qualifier for Return Values and Arguments of a BswModuleEntry</p> <p>Within the <i>SwDataDefProps</i> an additional type qualifier (e.g. "volatile") may be specified using the tag <ADDITIONAL-NATIVE-TYPE-QUALIFIER>.</p>
[BSWMD_Entry- RetArg_011]	<p>Definition of a Target Category for Data Pointers</p> <p>Scope: 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 <TARGET-CATEGORY>. The category of the referenced data shall be set.</p> <p>The following values are possible: VALUE, ARRAY or STRUCTURE</p>
[BSWMD_Entry- RetArg_012]	<p>Definition of Return Values and Arguments if the BswModuleEntry is a Macro</p> <p>If the <i>BswModuleEntry</i> represents a macro the return type and the arguments shall be defined without <i>SwDataDefProps</i>. The category tag shall be set to MACRO.</p>

Id	Rule <i>Chapter 8.3.4.1.2</i>
[BSWMD_Entry- RetArg_013]	<p>BswModuleEntry with no Return Value In case of an empty return type ("void" in C) no return value shall be specified within the <i>BswModuleEntry</i>. The <RETURN-TYPE> tag structure shall not be set.</p>
[BSWMD_Entry- RetArg_014]	<p>BswModuleEntry with no Argument In case of an empty argument list ("void" in C) no arguments shall be specified within the <i>BswModuleEntry</i>. The <ARGUMENT> tag structure shall not be set.</p>
[BSWMD_Entry- RetArg_015]	<p>LongName of the BswModuleEntry The tag item <LONG-NAME> may be used to specify a headline as explanation of the return value or the argument of a <i>BswModuleEntry</i>.</p>
[BSWMD_Entry- RetArg_016]	<p>Description of the BswModuleEntry The tag item <DESC> should be used to specify a short description of the return value or the argument of a <i>BswModuleEntry</i>.</p>
[BSWMD_Entry- RetArg_017]	<p>Introduction of the BswModuleEntry The tag item <INTRODUCTION> may be used to specify a more detailed explanation of the return value or the argument of a <i>BswModuleEntry</i>.</p>

Table 171 Export of a BswModuleEntry

Id	Rule <i>Chapter 8.3.4.1.3</i>
[BSWMD_EntryEx- port_001]	<p>Export of a BswModuleEntry of Type 'INTERRUPT', 'REGULAR' or 'SCHEDULED' BswModuleEntries of call type 'INTERRUPT', 'REGULAR' or 'SCHEDULED' shall be declared as provided entry to be exported outside of a BSW module. This shall be done in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <PROVIDED-ENTRYS> ▶ As next tag layer the tag item <BSW-MODULE-ENTRY-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several BswModuleEntries have to be exported. ▶ The reference to a BswModuleEntry shall be set within the tag item <BSW-MODULE-ENTRY--REF> ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY' ▶ A correctly and complete path to a BswModuleEntry shall be set
[BSWMD_EntryEx- port_002]	<p>Export of a BswModuleEntry of Type 'CALLBACK' BswModuleEntries of call type 'CALLBACK' shall be declared as outgoing callback to be exported outside of a BSW module. This shall be done in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <OUTGOING-CALLBACKS> ▶ As next tag layer the tag item <BSW-MODULE-ENTRY-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several BswModuleEntries have to be exported. ▶ The reference to a BswModuleEntry shall be set within the tag item <BSW-MODULE-ENTRY--REF> ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY' ▶ A correctly and complete path to a BswModuleEntry shall be set

Table 172 Properties of a Code Fragment (BswModuleEntity)

Id	Rule <i>Chapter 8.3.4.2</i>
[BSWMD_Entity_- 001]	<p>Kind of BswModuleEntity For the different kinds of <i>BswModuleEntitys</i> the following different tag items shall be used to define them within the <i>BswInternalBehavior</i> inside the <ENTITYS> item:</p> <ul style="list-style-type: none"> <BSW-CALLED-ENTITY> ... </BSW-CALLED-ENTITY> for <i>BswCalledEntitys</i> <BSW-INTERRUPT-ENTITY> ... </BSW-INTERRUPT-ENTITY> for <i>BswInterruptEntity</i> <BSW-SCHEDULABLE-ENTITY> ... </BSW-SCHEDULABLE-ENTITY> for <i>BswSchedulableEntity</i>
[BSWMD_Entity_- 002]	<p>ShortName of a BswModuleEntity Class: NamingConvention The ShortName of a <i>BswModuleEntity</i> shall be conform to the following convention: <Prefix>_<DescriptiveText>.</p>

Id	Rule <i>Chapter 8.3.4.2</i>
[BSWMD_Entity_003]	<p>Reference to a BswModuleEntry</p> <p>Every kind of <i>BswModuleEntity</i> shall refer to at least one <i>BswModuleEntry</i>. This shall be done in the following way:</p> <ul style="list-style-type: none"> ▶ A reference shall be set using the tag item <IMPLEMENTED-ENTRY-REF>. ▶ The destination type of the ImplementedEntryRef shall be set to 'BSW-MODULE-ENTRY'. ▶ A correctly and complete path to a <i>BswModuleEntry</i> shall be set ▶ The <i>BswModuleEntry</i> shall be specified as <i>ProvidedEntry</i> within the enclosing <i>BswModuleDescription</i> (conform to rule [BSWMD_EntryExport_001] p. 346) <p>In addition the following compatibility aspects shall be considered:</p> <ul style="list-style-type: none"> ▶ <i>BswCalledEntity</i>: The referred <i>BswModuleEntry</i> shall be of Call Type "REGULAR" or "CALLBACK" and the Execution Context shall be set to "UNSPECIFIED", "HOOK" or "TASK". ▶ <i>BswInterruptEntity</i>: The referred <i>BswModuleEntry</i> shall be of Call Type "INTERRUPT" and the Execution Context shall be set to "INTERRUPT-CAT-2". ▶ <i>BswSchedulableEntity</i>: The referred <i>BswModuleEntry</i> shall be of Call Type "SCHEDULED" and the Execution Context shall be set to "TASK".
[BSWMD_Entity_004]	<p>Reference to Mode Switch Which is Received from a BswModuleEntity</p> <p>Scope: <i>BswModuleEntity</i> receiving mode switches (defined as <i>BswModeDeclarations</i>)</p> <p>If a <i>BswModuleEntity</i> receives one or more mode switches, references to <i>BswModeDeclarations</i> shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <ACCESSED-MODE-GROUPS> ▶ As next tag layer the tag item <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several references to different <i>BswModeDeclarations</i> have to be set. ▶ The reference to a <i>BswModeDeclaration</i> shall be set within the tag item <MODE-DECLARATION-GROUP-PROTOTYPE-REF> ▶ The destination type of the <i>ModeDeclarationGroupPrototypeRef</i> shall be set to 'MODE-DECLARATION-GROUP' ▶ A correctly and complete path to a <i>BswModeDeclaration</i> shall be set (which is declared as <i>RequiredModeGroup</i> within the enclosing <i>BswModuleDescription</i>)
[BSWMD_Entity_005]	<p>Reference to an Internal Trigger Which is Activated from a BswModuleEntity</p> <p>Scope: <i>BswModuleEntity</i> activating one or more internal triggers (defined as <i>ReleasedTrigger</i>)</p> <p>If a <i>BswModuleEntity</i> activates one or more internal triggers references shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <ACTIVATION-POINTS> ▶ As next tag layer the tag item <BSW-INTERNAL-TRIGGERING-POINT-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several references to different <i>BswModeDeclarations</i> have to be set. ▶ The reference to a <i>BswTrigger</i> shall be set within the tag item <BSW-INTERNAL-TRIGGERING-POINT-REF> ▶ The destination type of the <i>BswInternalTriggeringPointRef</i> shall be set to 'BSW-INTERNAL-TRIGGERING-POINT' ▶ A correctly and complete path to a <i>BswInternalTriggeringPoint</i> shall be set (which is declared as <i>BswInternalTriggeringPoint</i> within the enclosing <i>BswInternalBehavior</i>)

Id	Rule Chapter 8.3.4.2
[BSWMD_Entity_006]	<p>Reference to an Entry of Another Module (or the Same Module) Which is Called from a BswModuleEntity</p> <p>Scope: BswModuleEntity calling entries of another BSW module (or the same module)</p> <p>If a <i>BswModuleEntity</i> calls BswModuleEntrys of another (or the same) BSW modules references shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <CALLED-ENTRYS> ▶ As next tag layer the tag item <BSW-MODULE-ENTRY-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several BswModuleEntries have to be referred. ▶ The reference to a BswModuleEntry shall be set within the tag item <BSW-MODULE-ENTRY-REF> ▶ The destination type of the BswModuleEntryRef shall be set to 'BSW-MODULE-ENTRY' ▶ A correctly and complete path to a BswModuleEntry shall be set (which is provided as OutgoingCallback, ProvidedEntry or as RequiredEntry within the enclosing BswModuleDescription)
[BSWMD_Entity_007]	<p>Reference to a Trigger Which is Activated for Other BSW Modules</p> <p>Scope: BswModuleEntity activating triggers for other BSW modules</p> <p>If a <i>BswModuleEntity</i> activates triggers for other BSW modules references shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <ISSUED-TRIGGERS> ▶ As next tag layer the tag item <TRIGGER-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several references to different BswModeDeclarations have to be set. ▶ The reference to a BswModeDeclaration shall be set within the tag item <TRIGGER-REF> ▶ The destination type of the TriggerRef shall be set to 'TRIGGER' ▶ A correctly and complete path to a trigger shall be set (which is declared as ReleasedTrigger within the enclosing BswModuleDescription)
[BSWMD_Entity_008]	<p>Reference to Initiate a Mode Switch for Other BSW Modules</p> <p>Scope: BswModuleEntity initiating mode switches for other BSW modules</p> <p>If a <i>BswModuleEntity</i> initiates mode switches for other BSW modules references shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <MANAGED-MODE-GROUPS> ▶ As next tag layer the tag item <MODE-DECLARATION-GROUP-PROTOTYPE-REF-CONDITIONAL> shall be set. This tag layer can be used multiple times if several references to different BswModeDeclarations have to be set. ▶ The reference to a BswModeDeclaration shall be set within the tag item <MODE-DECLARATION-GROUP-PROTOTYPE-REF> ▶ The destination type of the ModeDeclarationGroupPrototypeRef shall be set to 'MODE-DECLARATION-GROUP' ▶ A correctly and complete path to a BswModeDeclaration shall be set (which is declared as RequiredModeGroup within the enclosing BswModuleDescription)
[BSWMD_Entity_009]	<p>Specifying the Category of the BswInterruptEntity</p> <p>Scope: Only relevant for BswInterruptEntitys</p> <p>Every BswInterruptEntity shall specify the category of the interrupt service using the tag item <INTERRUPT-CATEGORY>.</p> <p>The applied category shall be the same as used in the referred BswModuleEntry.</p>

05
Table 173 BswEvents

Id	Rule Chapter 8.3.4.3
[BSWMD_Event_001]	<p>Kind of BswEvents</p> <p>For the different kinds of <i>BswEvents</i> the following different tag items shall be used to define them within the <i>BswInternalBehavior</i> inside the <EVENTS> item:</p> <ul style="list-style-type: none"> <BSW-TIMING-EVENT> ... </BSW-TIMING-EVENT> for <i>BswTimingEvent</i> <BSW-BACKGROUND-EVENT> ... </BSW-BACKGROUND-EVENT> for <i>BswBackgroundEvent</i> <BSW-MODE-SWITCH-EVENT> ... </BSW-MODE-SWITCH-EVENT> for <i>BswModeSwitchEvent</i> <BSW-MODE-SWITCHED-ACK-EVENT> ... </BSW-MODE-SWITCHED-ACK-EVENT> for <i>BswModeSwitchedAckEvent</i> <BSW-EXTERNAL-TRIGGER-OCCURED-EVENT> ... </BSW-EXTERNAL-TRIGGER-OCCURED-EVENT> for <i>BswExternalTriggerOccuredEvent</i> <BSW-INTERNAL-TRIGGER-OCCURED-EVENT> ... </BSW-INTERNAL-TRIGGER-OCCURED-EVENT> for <i>BswInternalTriggerOccuredEvent</i>
[BSWMD_Event_002]	<p>ShortName of a BswEvent</p> <p>Class: NamingConvention</p> <p>The ShortName of a <i>BswEvent</i> shall be conform to the following convention: <Prefix>_<DescriptiveText>.</p>
[BSWMD_Event_003]	<p>Reference to a BswModuleEntity</p> <p>Every kind of a <i>BswEvent</i> shall refer to a single <i>BswModuleEntity</i> (which is started by the event) which is done in the following way:</p> <ul style="list-style-type: none"> ▶ The reference shall be set using the tag item <STARTS-ON-EVENT-REF>. ▶ The destination type of the StartsOnEventRef shall be set to the specific type of the <i>BswModuleEntity</i> ('BSW-CALLED-ENTITY', 'BSW-SCHEDULABLE-ENTITY' or 'BSW-INTERRUPT-ENTITY'). ▶ A correctly and complete path to a <i>BswModuleEntity</i> shall be set.
[BSWMD_Event_004]	<p>Timing Period for BswTimingEvents</p> <p>Scope: Only relevant for <i>BswTimingEvents</i></p> <p><i>BswTimingEvents</i> shall specify a time period (in seconds) using the tag item <PERIOD> and shall have a value greater than 0.</p>
[BSWMD_Event_005]	<p>Activation of a BswModeSwitchEvent</p> <p>Scope: Only relevant for <i>BswModeSwitchEvents</i></p> <p>The kind of activation of a <i>BswModeSwitchEvent</i> shall be specified using the tag item <ACTIVATION>.</p> <p>The following settings are possible: <i>ON-ENTRY</i>, <i>ON-EXIT</i>, <i>ON-TRANSITION</i></p> <p>If the activation is set to the value '<i>ON-TRANSITION</i>' the <i>BswModeSwitchEvent</i> shall refer to two distinct modes belonging to the same instance of a <i>ModeDeclarationGroup</i>, their order defining the direction of the transition. In all other cases the <i>BswModeSwitchEvent</i> shall refer to exactly one mode.</p>
[BSWMD_Event_006]	<p>References to Modes for BswModeSwitchEvents</p> <p>Scope: Only relevant for <i>BswModeSwitchEvents</i></p> <p>Based on the chosen kind of activation of a <i>BswModeSwitchEvent</i> one or two references to BSW modes shall be set. One reference is needed if the activation is set to '<i>ON-ENTRY</i>' or '<i>ON-EXIT</i>', two references are needed for '<i>ON-TRANSITION</i>'.</p>



Id	Rule Chapter 8.3.4.3
	<p>The references shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ A tag structure shall be created starting with tag item <MODE-IREFS> ▶ Within that one or two sub-structures shall be created using the tag item <MODE-IREF> ▶ Within the Modelrefs, references to a BswModeDeclarationGroupPrototype and a BswModeDeclaration shall be set. Therefore the tag items <CONTEXT-MODE-DECLARATION-GROUP-REF> and <TARGET-MODE-REF> shall be used. ▶ The destination type of the ContextModeDeclarationGroupRef shall be set to 'MODE-DECLARATION-GROUP-PROTOTYPE' and the destination type of the TargetModeRef shall be set to 'MODE-DECLARATION' ▶ Complete paths to the BswModeDeclarationGroupPrototype and the BswModeDeclaration shall be set <p>Additional condition: The ModeDeclaration used by a BswModeSwitchEvent shall be specified as AccessedModeGroup as part of the BswSchedulableEntity of the enclosing BswInternalBehavior of the BSW module.</p>
[BSWMD_Event_007]	<p>Reference to a Mode Group for BswModeSwitchedAckEvents</p> <p>Scope: Only relevant for BswModeSwitchedAckEvents</p> <p>Within a BswModeSwitchedAckEvent a reference to a ModeDeclarationGroupPrototype shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ The reference shall be set using the tag item <MODE-GROUP-REF> ▶ The destination type of the ModeGroupRef shall be set to 'MODE-DECLARATION-GROUP-PROTOTYPE' ▶ A correctly and fully specified path to the ModeDeclarationGroupPrototype shall be set <p>Additional condition: The ModeDeclarationGroupPrototype used by a BswModeSwitchedAckEvent shall be specified as ProvidedModeGroup as part of the BswModuleDescription of the same BSW module.</p>
[BSWMD_Event_008]	<p>Reference to an External Trigger for a BswExternalTriggerOccuredEvent</p> <p>Scope: Only relevant for BswExternalTriggerOccuredEvents</p> <p>Within a BswExternalTriggerOccuredEvent a reference to a Trigger shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ The reference shall be set using the tag item <trigger-ref> ▶ The destination type of the TriggerRef shall be set to 'trigger' ▶ A correctly and complete path to the Trigger shall be set <p>Additional condition: A BswExternalTriggerOccurredEvent shall refer to a Trigger that is declared as RequiredTrigger within the BswModuleDescription of the same module.</p>
[BSWMD_Event_009]	<p>Reference to an Internal Trigger for a BswInternalTriggerOccuredEvent</p> <p>Scope: Only relevant for BswInternalTriggerOccuredEvents</p> <p>Within a BswInternalTriggerOccuredEvent a reference to a BswInternalTriggeringPoint shall be set in the following way:</p> <ul style="list-style-type: none"> ▶ The reference shall be set using the tag item <event-source-ref> ▶ The destination type of the EventSourceRef shall be set to 'BSW-INTERNAL-TRIGGERING-POINT' ▶ A correctly and complete path to the Trigger shall be set

05 Table 174 BswModes

Id	Rule <i>Chapter 8.3.4.4</i>
[BSWMD_Mode_-001]	<p>Definition of BSW Modes 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.</p>

Table 175 BSW Mode Management on the Mode Manager Side

Id	Rule <i>Chapter 8.3.4.4.1</i>
[BSWMD_Mode_-002]	<p>Definition of BSW Mode Management on the Mode Manager side 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.</p>

Table 176 BSW Mode Management on the Mode User Side

Id	Rule <i>Chapter 8.3.4.4.2</i>
[BSWMD_Mode_-003]	<p>Definition of BSW Mode Management on the Mode User side For BSW Modes on the Mode User side the RequiredModeGroups shall be defined in the BswModuleDescription. The ModeDeclarationGroupPrototype shall be one element of the RequiredModeGroups. A reference to the ModeDeclarationGroup shall also be defined in the TypeTref of the ModeDeclarationGroupPrototype. All these model elements shall exist.</p>
[BSWMD_Mode_-004]	<p>Definition of BswModeSwitchEvent For each BSW Mode Switch Event, a BswModeSwitchEvent element shall be defined in a BswInternalBehavior. 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.</p>

Table 177 BSW Mode Access

Id	Rule <i>Chapter 8.3.4.4.3</i>
[BSWMD_Mode_-005]	<p>Definition of BSW Mode Access If a BswSchedulableEntity needs to access to the current Mode the AccessedModeGroups shall be defined in the BswSchedulableEntity with a reference to the ModeDeclarationGroupPrototype.</p>

Table 178 BswExternalTriggers on the Sender side

Id	Rule <i>Chapter 8.3.4.5.1</i>
[BSWMD_Trigger_-001]	<p>Definition of BswExternalTriggers on the Sender side For each External Trigger, a Trigger element shall be defined in a ReleasedTrigger element of a BswModuleDescription. For the BswSchedulableEntity, a reference to the ExternalTrigger shall be defined in TriggerRef-Conditional element in the IssuedTrigger element of the BswSchedulableEntity. All these model elements shall exist.</p>

Table 179 BswExternalTriggers on the Receiver side

Id	Rule <i>Chapter 8.3.4.5.2</i>
[BSWMD_Trigger_-002]	<p>Definition of BswExternalTriggers on the Receiver side</p> <p>For each External Trigger, a Trigger element shall be defined in a RequiredTrigger element of a BswModuleDescription.</p>
[BSWMD_Trigger_-003]	<p>Definition of BSWExternalTriggerOccuredEvent</p> <p>For each BSW External Trigger Occured Event, a BswExternalTriggerOccuredEvent element shall be defined in an Events element of a BswInternalBehavior.</p> <p>The concerned BswSchedulableEntity shall be referenced in a StartsOnEventRef.</p> <p>All these model elements shall exist.</p>

Table 180 BswInternalTriggers

Id	Rule <i>Chapter 8.3.4.5.3</i>
[BSWMD_Trigger_-004]	<p>Definition of BswInternalTriggers</p> <p>For each Internal Trigger, a BswInternalTriggeringPoint element shall be defined in a InternalTriggeringPoints element of a BswInternalBehavior.</p>
[BSWMD_Trigger_-005]	<p>Definition of BswInternalTriggerOccuredEvent</p> <p>For each BSW Internal Trigger Occured Event, a BswInternalTriggerOccuredEvent element shall be defined in an Events element of a BswInternalBehavior.</p> <p>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.</p> <p>The concerned BswSchedulableEntity shall be referenced in a StartsOnEventRef.</p> <p>All these model elements shall exist.</p>

Table 181 ExecutionOrderConstraint Description

Id	Rule <i>Chapter 9.2 [ExecutionOrderConstraint_Description]</i>
[EOC_001]	<p>EOC file delivery</p> <p>Each SW module (SWCD, BSWMD) which have scheduled entities (AbstractEvents) shall provide an EOC (Execution Order Constraint) file.</p> <p>Class: NamingConvention ... for an EOC file: [ModuleName]_EOC.arxml</p> <p>If there is no AbstractEvent in a SW module which shall be scheduled by an Os task no EOC file must be provided.</p> <p>Hint Only event based EOC will be provided that means only objects which will be scheduled by an Os task.</p>
[EOC_002]	<p>EOC for generated SWCD / BSWMD</p> <p>For generated SWCD / BSWMD the corresponding EOC shall also be generated.</p> <p>If the Os task (the time slot) for an AbstractEvent will be defined by a Customer Integration Team the EOC file must be generated during the SW build by a script file. This script file must be part of the SW module.</p>

Id	Rule <i>Chapter 9.2 [ExecutionOrderConstraint_Description]</i>
[EOC_003]	<p>EOC file header</p> <p>Each EOC ARXML file shall start with a header containing the definition of the used AUTOSAR schema.</p> <pre data-bbox="335 415 1472 527"><?xml version="1.0" encoding="UTF-8"?> <AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-2-1.xsd"></pre> <p>Hint 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. It will be done automated if you use the tool iSolar.</p>
[EOC_004]	<p>Scope of an EOC file</p> <p>Goal is to describe the correct AbstractEvent sequence order on ECU level. Each EOC file shall have the scope of <ECU-TIMING>.</p>
[EOC_005]	<p>EOC relevant objects</p> <p>Each AbstractEvent shall be referenced by a <EXECUTION-ORDER-CONSTRAINT>. Each <EXECUTION-ORDER-CONSTRAINT> should refer to AbstractEvents and not to ExecutableEntities.</p> <p>The element <EOC-EVENT-REF> is used to reference RTEEvents or BswEvents.</p> <p>The <EXECUTION-ORDER-CONSTRAINT-TYPE> shall be set to <i>ORDINARY-EOC</i>.</p> <p>The Attribute <IS-EVENT> shall be set to <i>true</i>.</p>
[EOC_006]	<p>EOC sequencing strategy</p> <p>Each ordered element shall have a successor (<SUCCESSOR-REF>) or a directSuccessor (<DIRECT-SUCCESSOR-REF>).</p> <p>It is possible to describe the process/runnable sequence out of the successor view (use case 1a) or out of the predecessor view (use case 1b).</p> <p>Hint You can describe the sequence order only with successor / directSuccessor. AUTOSAR doesn't provide a predecessor tag.</p> <p>use case 1: If Proc1 is successor of Proc2 and Proc2 must not follow directly on Proc1 successor should be used.</p> <ul style="list-style-type: none"> ▶ In this case it doesn't matter where Proc2 is located in the Os task, only after Proc1. <p>use case 2: If Proc1 is the direct successor of Proc2 use directSuccessor.</p> <ul style="list-style-type: none"> ▶ In this case the sequence is hard coded, Proc2 directly after Proc1. <p>use case 3: Proc1 is autonomous, i.e. can be called from everywhere.</p> <ul style="list-style-type: none"> ▶ [ModuleName]_EOC.arxml shall be provided but with no dependencies to other processes/runnables. <p>use case 4: If an AbstractEvent is legal removable cramp this AbstractEvent with a <VARIATION-POINT>.</p> <ul style="list-style-type: none"> ▶ With a variation point you can define if a part exists or not.

Id	Rule <i>Chapter 9.2 [ExecutionOrderConstraint_Description]</i>
[EOC_007]	<p>Content of an <EXECUTION-ORDER-CONSTRAINT></p> <p>All AbstractEvents in an <EXECUTION-ORDER-CONSTRAINT> shall be compatible regarding their recurrence. So all <EOC-EVENT-REF> in an <EXECUTION-ORDER-CONSTRAINT> shall be of the same event type (TimingEvent, InitEvent, ...) and of the same event property (period: 10ms, 20ms, ...).</p>
[EOC_008]	<p>Naming Convention for an <EOC-EVENT-REF></p> <p>Class: NamingConvention ... for an <EOC-EVENT-REF>: EOC_[EventType]_[EventProperty]</p> <p>The element <EOC-EVENT-REF> shall contain information about the event type and event property. e.g.: EOC_T_10ms</p> <ul style="list-style-type: none"> ▶ event type: T <BSW-TIMING-EVENT> ▶ event property: 10ms (period of <BSW-TIMING-EVENT>) <p>Hint Input for the Customer Integration Team to know in which task the referenced entities should be scheduled.</p> <p>List of legal event types and their abbreviations see Table <i>/ARAbstractEvents "AUTOSAR conform AbstractEvents (Event Types)"</i>.</p> <p>The name of an event property correlates to the name of an Os task, like "10ms", "50ms", "ini".</p>

Table 182 Main Structure of ARPackages

Id	Rule <i>Chapter 10.2 [ARPackage-1]</i>
[ARPac_10]	<p>Main Pattern for Package Structure</p> <p>Scope: RB deliverables, except AUTOSAR specified software (e.g. BSW modules, Application Interfaces (AISpecification))</p> <p>Class: NamingConvention</p> <p>The following structure should be followed:</p> <pre data-bbox="335 1347 857 1504">/RB / {domain} [/ {subdomain}]* / {block} /{kind}[_Blueprint _Example]</pre> <p>where</p> <ul style="list-style-type: none"> ▶ 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] <p>For an AUTOSAR specified BSW module there is another package structure to be used. See rule [ARPac_41].</p> <p>For ECU Configuration values a special ARPackage structure is to be used ("/RB/UBK/Project/..."). See rule [ARPac_46].</p>

Id	Rule Chapter 10.2 [ARPackage-1]
[ARPac_11]	<p>Top Level ARPackage</p> <p>Scope: RB deliverables, except AUTOSAR specified software (e.g. BSW modules)</p> <p>Class: NamingConvention</p> <p>There shall be exactly one top level ARPackage in an .arxml file.</p> <p>The top level package shall be named "RB".</p>
[ARPac_12]	<p>Domain oriented ARPackages</p> <p>Scope: RB deliverables, except AUTOSAR specified BSW modules</p> <p>Class: NamingConvention</p> <p>The package name on second level should be the name of the ECU-domain, as defined in Table 62.</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>
[ARPac_13]	Removed
[ARPac_14]	<p>Names for ARElement-kind based ARPackages</p> <p>Class: NamingConvention</p> <p>Whenever (sub-) ARPackages are created for the different kinds of ARElements, ARPackage'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>. A complete List is given in Table 63 [Recommended Packages]</p> <p>If the ARPackage 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_Data] or other guidelines.</p>
[ARPac_14A]	Sort kind-based ARPackages alphabetically
[ARPac_15]	Omit empty ARPackages
[ARPac_16]	ARPackage for Common ARElements
	<p>Class: NamingConvention</p> <p>There are different kinds of common ARElements. They shall be covered by an ARPackage named "-Common". The currently known common element types are: CentralElements, DesignPatterns, DocumentationPatterns, NamePatterns, PackagePatterns, SwArchitecture, NamingConventions.</p> <p>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.</p>
[ARPac_17]	<p>Category of ARPackages</p> <p>The following Categories for ARPackage are predefined by AUTOSAR:</p> <ul style="list-style-type: none"> ▶ BLUEPRINT (blueprint elements) ▶ STANDARD (standardized objects) ▶ ICS (Implementation Conformance Statement) ▶ EXAMPLE (examples how to apply Elements in STANDARD or BLUEPRINT packages) <p>You shall NOT define any other category for ARPackages.</p>
[ARPac_18]	<p>Empty Category of ARPackages</p> <p>For ARPackages which contain model elements of none of the categories BLUEPRINT, STANDARD, ICS, EXAMPLE you shall NOT set any category. The Category attribute shall be omitted.</p> <p>This is the standard case for most ARPackages.</p>

Id	Rule <i>Chapter 10.2 [ARPackage-1]</i>
[ARPac_191]	<p>Category BLUEPRINT for ARPackages</p> <p>The Category of ARPackages containing blueprint elements shall be set to BLUEPRINT.</p>
[ARPac_192]	<p>Category STANDARD for ARPackages</p> <p>The Category of ARPackages containing standardized elements shall be set to STANDARD. This Category is typically created only by architects.</p>
[ARPac_20]	<p>Category ICS, EXAMPLE for ARPackages</p> <p>The Categorys ICS, EXAMPLE shall NOT be used since the usage is not defined in our methods.</p>

Table 183 ARPackage for Basic Software

Id	Rule <i>Chapter 10.3 [ARPackage-BSW]</i>
[ARPac_41]	<p>Top level ARPackages for AUTOSAR Specified BSW Modules</p> <p>Class: NamingConvention</p> <p>Scope: BSW modules specified by AUTOSAR</p> <p>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.</p> <p>The ShortName shall be "AUTOSAR_" + {moduleName}.</p> <p>List of BSW-Modules is given in table Table 64 [BSWModuleNames], taken from AUTOSAR specification 4.1.1.</p>
[ARPac_43]	<p>Top level ARPackages for non AUTOSAR Specified BSW Modules</p> <p>Scope: BSW modules not specified by AUTOSAR</p> <p>Class: NamingConvention</p> <p>Such modules shall be put into the top level ARPackage "RB".</p> <p>They shall be put into the second level ARPackage for the respective domain. This ARPackage shall be either "RBA" if it is related to RBA, otherwise the ECU domain (e.g. PT for power train).</p> <p>The ARPackage on the next level shall get the name of the respective BSW module.</p>
[ARPac_44]	<p>Kind ARPackages for Basic Software Modules</p> <p>Class: NamingConvention</p> <p>For BSW modules the rule "Rule ARPac_14: Names for ARElement-kind based ARPackages" shall be followed.</p> <p>It shall be followed by AUTOSAR specified and non-AUTOSAR specified BSW modules.</p>
[ARPac_45]	<p>Software Module Configuration (pre-configured or recommended values) -- currently NOT used</p> <p>Scope: upcoming</p> <p>Pre-configured or recommended ECUC values for BSW modules (AUTOSAR specified or not) shall be put into the ARPackage of the module itself and within that into a sub package EcucModuleConfigurationValueess.</p> <p>Examples:</p> <p>/AUTOSAR_LinSM/EcucModuleConfigurationValueess /RB/RBA/rba_Bernd/EcucModuleConfigurationValueess</p> <p>For pre-configured values the container's ShortName shall be PreconfiguredValues.</p> <p>For recommended values the container's ShortName shall be RecommendedValues.</p>

Id	Rule Chapter 10.3 [ARPackage-BSW]
[ARPac_46]	<p>Software Module Configuration (for protected values)</p> <p>Protected configuration values for BSW modules (AUTOSAR specified or not) shall be put into the following ARPackage structure:</p> <pre>/RB /UBK /Project /EcucModuleConfigurationValues</pre> <p>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.</p> <p>Protected configuration values shall NOT be put into the BSW module's ARPackage (e.g. /AUTOSAR_LinSM or /RB/RBA/RBA_Bernd).</p>
[ARPac_47]	<p>ARPackage for BSW Component Types</p> <p>Class: NamingConvention</p> <p>Each SwComponentType (e.g. ServiceSwComponentType) which is defined as part of a BSW module shall be put into the ARPackage as defined in /ARPac_41 [ARPac_41] / /ARPac_43 [ARPac_43] and /ARPac_44 [ARPac_44].</p> <p>Note: This rule is not applicable for SwComponentTypes which are part of ASW.</p>

Table 184 Use of ReferenceBases

Id	Rule Chapter 10.4.2 [ARPackage-RefBase]
[ARPac_71]	<p>Avoid Usage of Element ReferenceBases</p> <p>The usage of the Element ReferenceBase is not yet supported by all authoring tools and should NOT be used.</p> <p>Exceptions are defined in subsequent rules.</p>
[ARPac_72]	<p>Usage of ReferenceBase</p> <p>In tool-generated arxml files ReferenceBase should be used for making the files more readable.</p>
[ARPac_73]	<p>A defined ReferenceBase shall always be used</p> <p>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.</p>
[ARPac_74]	<p>ShortName of ReferenceBases</p> <p>Class: NamingConvention</p> <p>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] (including the plural-s).</p>
[ARPac_75]	<p>ReferenceBases and their usage in references shall reside in the same delivery unit</p> <p>The definition of a ReferenceBase shall be in the same ArPackage as their usage in references. This could be the top level ArPackage eg. of a component or module.</p>
[ARPac_76]	<p>All attributes of ReferenceBases shall be specified explicitly</p> <p>All attributes of ReferenceBases shall be specified explicitly.</p> <p>Attribute <i>IsDefault</i> shall always be set to <i>false</i>.</p> <p>Attribute <i>IsGlobal</i> shall always be set to <i>false</i>.</p> <p>Attribute <i>BaselsThisPackage</i> shall always be set to <i>false</i>.</p> <p>Attribute <i>PackageRef</i> shall always be specified.</p>
[ARPac_77]	<p>Order of attributes in references</p> <p>The required attributes of a reference shall be in the following order: first <i>Base</i>, then <i>Dest</i>.</p>

Table 185 Centralization of Element Kinds

Id	Rule <i>Chapter 11.1</i>
[CEL_120]	<p>Dealing with Central Elements</p> <p>If a central element exists, it shall be used.</p> <p>In exceptional cases elements might be needed earlier than being provided in a central package. In this case it is allowed to define the element locally first. In parallel a request for a central element shall be triggered and a first consensus with the provider of the central package should be found. The switch from the local element to the central element shall be done in the next refactoring step of the component.</p> <p>These local elements shall contain a local ARPackage path, i.e. this path is local to the corresponding module and must not be referenced outside this module. Then, only the ARPackage path has to be adjusted pointing to the central element once it is supported.</p> <p>All elements that are not explicitly mentioned to be defined as central elements within these central elements guidelines shall be defined locally only.</p>

Table 186 How to use the Central Elements

Id	Rule <i>Chapter 11.2 [CenElemChapter_PackStructure]</i>
[CEL_011]	<p>ARPackage Structure for Central Elements</p> <p>Scope: Bosch defined central elements</p> <p>The path declaration for central elements shall be of the form: /RB/{domain}/Common/CentralElements/{kind}s</p> <p>A list of all possible domains can be found in Rule ARPac_12 of the ARPackage Guideline.</p> <p>Note: AUTOSAR defined standardized objects which will use other package paths.</p>
[CEL_012]	<p>Package Path for AUTOSAR StandardTypes</p> <p>The StandardTypes shall be implemented as <i>ImplementationDataTypes</i> of the top-level ARPackage AUTOSAR_Std.</p>
[CEL_013]	<p>Package Path for AUTOSAR SwBaseTypes</p> <p>The SwBaseTypes shall be implemented in the ARPackage /AUTOSAR_Platform/SwBaseTypes.</p>
[CEL_016]	<p>Package Path for RB StandardTypes</p> <p>The RB StandardTypes shall be implemented as <i>ImplementationDataTypes</i> of either ARPackage /RB/Common/CentralElements or /RB/{domain}/Common/CentralElements.</p>
[CEL_020]	<p>Referencing Central Elements</p> <p>Scope: ASW</p> <p>An ASW product line shall not use central elements of another product line. An ASW product line may use BBM common central elements, CUBAS central elements, their own central elements and the AUTOSAR central elements.</p> <p>E. g. ASW PT may use /RB/Common/CentralElements, /RB/RBA/Common/CentralElements, /AUTOSAR_* and /RB/PT/Common/CentralElements.</p> <p>E. g. ASW PT shall not use /RB/{Chassis}/Common/CentralElements.</p> <p>CUBAS may access /AUTOSAR/, /AUTOSAR_* and /RB/RBA, but not /RB/Common/CentralElements.</p> <p>Reason: Self-containment of CUBAS, because CUBAS should be delivered as separate product.</p>
[CEL_053]	<p>Package Path for AUTOSAR ComStackTypes</p> <p>The ComStackTypes shall be implemented as <i>ImplementationDataTypes</i> of the top-level ARPackage AUTOSAR_Comtype.</p>
[CEL_112]	<p>Package Path for AUTOSAR PlatformTypes</p> <p>The PlatformTypes shall be implemented as <i>ImplementationDataTypes</i> of the top-level ARPackage AUTOSAR_Platform.</p>
[CEL_113]	<p>Package Path for CUBAS CompuMethods</p> <p>Centrally defined CompuMethods shall be implemented in ARPackage /RB/RBA/Common/CentralElements/CompuMethods.</p>

Id	Rule <i>Chapter 11.2 [CenElemChapter_PackStructure]</i>
[CEL_114]	<p>Package Path for CUBAS SwAddrMethods Centrally defined <i>SwAddrMethods</i> shall be implemented in ARPackage /RB/RBA/Common/Central-Elements/<i>SwAddrMethods</i>.</p>
[CEL_115]	<p>Package Path for AUTOSAR KeywordSets The ARPackagePath for AUTOSAR <i>KeywordSets</i> shall be of the form: /AUTOSAR_AISpecification/KeywordSets/{NameOfKeywordSet} derived from /AUTOSAR/AISpecification/KeywordSets_Blueprint/KeywordList The ARPackagePath for BBM <i>KeywordSets</i> shall be of the form: /RB/Common/NamingConventions/KeywordSets/{NameOfKeywordSet}</p>
[CEL_117]	<p>Usage of ReferenceBases DerivedFrom: ARPac_73: A defined <i>ReferenceBase</i> shall always be used If a centrally defined <i>ReferenceBase</i> exists, it shall be used in all references to that particular package (cf. rule ARPac_73 of the ARPackage guideline). Reason: Avoid inconsistencies if the <i>ReferenceBase</i> path changes.</p>
[CEL_131]	<p>Selecting "_FAST", "_SLOW" SwAddrMethods <i>SwAddrMethods</i> with "_SLOW" in the name shall be used for variables / parameters accessed by runnables slower than 20 ms and other rarely called functions. <i>SwAddrMethods</i> with "_FAST" in the name shall be used for variables / parameters accessed by runnables faster than 5 ms and frequently used interrupts. In other cases <i>SwAddrMethods</i> without "_FAST" / "_SLOW" shall be used.</p>

Table 187 Delivery Aspects

Id	Rule <i>Chapter 11.3 [CenElemChapter_DeliveryAspects]</i>
[CEL_018]	<p>Delivery Units In order to refer to unique names within this guideline, a general definition of the delivery unit names was done here. Furthermore these names can be mapped to SCM-specific implementations. The delivery unit for CUBAS central elements shall be CUCEL. CUCEL shall provide 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 AISpecification and for BBM (was UBK) shall be (tbd; was UBKCEL; will be renamed somehow). The delivery unit for business unit specific central elements shall be the business unit specific CELs.</p>
[CEL_019]	<p>Additional Central Elements in Business Unit Business units may define additional delivery units for central elements.</p>
[CEL_027]	<p>Backward Compatibility of CUCEL and other central elements deliveries CUCEL and other central elements deliveries shall always be backward compatible.</p>

Table 188 Structural Conventions

Id	Rule <i>Chapter 12.2.1</i>
[CLF_Structure_-001]	<p>CLF for Packages, components and configuration 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.</p>
[CLF_Structure_-002]	<p>Project There shall be only one top-level CLF file that defines the project specific includes and variants.</p>
[CLF_Structure_-003]	<p>Package Classes 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</p>

Id	Rule <i>Chapter 12.2.1</i>
[CLF_Structure_-004]	<p>File Classes 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</p>
[CLF_Structure_-005]	<p>Exclusivity In CUBAS packages, structural components, components and configuration there shall be only one CLF file in one folder.</p>
[CLF_Structure_-006]	<p>Variants 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.</p>

Table 189 Naming Conventions

Id	Rule <i>Chapter 12.2.2</i>
[CLF_Naming_-001]	<p>Domain Class: NamingConvention The domain shall be used only once in the top level CLF file, and it shall have the value CUBAS.</p>
[CLF_Naming_-002]	<p>Name uniqueness Class: NamingConvention The CLF file name and the containing package name shall be identical. CLF files and package names shall be unique across the whole project</p>
[CLF_Naming_-003]	<p>Packages and components Class: NamingConvention 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.</p>
[CLF_Naming_-004]	<p>Structural components Class: NamingConvention CLF files for structural components shall have the name of the respective structural component extended by _Stc.</p>
[CLF_Naming_-005]	<p>Packages and components configuration Class: NamingConvention CLF files for configuration shall have the name of their respective package or component, extended by _Cfg.</p>
[CLF_Naming_-006]	<p>Structural components configuration Class: NamingConvention CLF files for configuration of structural components shall have the name of the respective structural component, extended by _Stc_Cfg.</p>
[CLF_Naming_-007]	<p>Variant Names Class: NamingConvention 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></p>

B Rules Derivation

The following [Table 190](#) 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_xyx and SWCDT are requirements from "[Software Component Template](#)"

In the last column "Reference MISRA C:2012" all references to MISRA rules are listed. Additional markers are used within this column. If more than one MISRA rule is listed BSW coding rule represents a summary of listed MISRA rules.

Table 190 Overview of Rules Derivation (AUTOSAR and MISRA)

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule CCode_001	-	Rule 1.1, Rule 1.2
Rule CCode_002	-	Rule 4.2
Rule CCode_003	-	Rule 4.1, Rule 7.1
Rule CCode_004	-	Rule 6.1, Rule 6.2
Rule CCode_005	-	Dir 1.1
Rule CCode_006	-	Rule 22.3, Rule 22.4, Rule 22.5, Rule 22.6
Rule CCode_007	-	Dir 4.11
Rule CCode_MisraHIS_001	BSW007	-
Rule CCode_MisraHIS_002	BSW007	-
Rule CCode_MisraHIS_003	-	-
Rule CCode_MisraHIS_004	-	-
Rule CCode_Inits_001	-	Rule 9.1
Rule CCode_Inits_002	-	Rule 8.12
Rule CCode_Inits_003	-	Rule 13.1
Rule CCode_Inits_004	-	Rule 9.4
Rule CCode_Inits_005	-	Rule 9.5
Rule CCode_Expr_001	-	Rule 12.1
Rule CCode_Expr_002	-	Rule 13.2
Rule CCode_Expr_003	-	Rule 13.6
Rule CCode_Expr_004	-	Rule 13.5
Rule CCode_Expr_005	-	Rule 12.1
Rule CCode_Expr_006	PLATFORM034 (partial)	Rule 10.1
Rule CCode_Expr_007	-	Rule 10.1
Rule CCode_Expr_008	-	Rule 12.2

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule CCode_Expr_009	-	Rule 10.1
Rule CCode_Expr_010	-	Rule 12.3
Rule CCode_Expr_011	-	Rule 12.4
Rule CCode_Expr_012	-	Rule 13.3
Rule CCode_Expr_014	-	-
Rule CCode_Expr_015	-	-
Rule CCode_Control_001	-	Rule 13.4
Rule CCode_Control_002	-	Rule 14.4
Rule CCode_Control_003	-	-
Rule CCode_Control_004	-	Rule 14.2
Rule CCode_Control_005	-	Rule 14.2
Rule CCode_Control_006	-	Rule 14.1
Rule CCode_Control_007	-	Rule 14.3
Rule CCode_CntrFlow_001	-	Rule 2.1
Rule CCode_CntrFlow_002	BSW00328	-
Rule CCode_CntrFlow_003	-	Rule 2.2
Rule CCode_CntrFlow_004	-	-
Rule CCode_CntrFlow_005	-	Rule 2.6, Rule 15.1
Rule CCode_CntrFlow_006	-	Rule 15.4
Rule CCode_CntrFlow_007	-	Rule 15.7
Rule CCode_Switch_007	-	Rule 16.1
Rule CCode_Switch_001	-	Rule 16.1
Rule CCode_Switch_002	-	Rule 16.2
Rule CCode_Switch_003	-	Rule 16.3
Rule CCode_Switch_004	-	Rule 16.4, Rule 16.5
Rule CCode_Switch_005	-	Rule 16.7
Rule CCode_Switch_006	-	Rule 16.6
Rule CCode_Struct_001	-	Rule 19.1, Rule 19.2
Rule CCode_Struct_002	-	-
Rule CCode_Struct_003	-	Rule 1.3
Rule CCode_Struct_004	-	-
Rule CCode_Prepo_001	-	Rule 20.1
Rule CCode_Prepo_002	-	Rule 20.2
Rule CCode_Prepo_003	-	Rule 20.3
Rule CCode_Prepo_004	-	Rule 20.5
Rule CCode_Prepo_005	-	Rule 1.3
Rule CCode_Prepo_006	-	Rule 20.13
Rule CCode_Prepo_007	-	Rule 20.14
Rule CCode_Prepo_008	-	Rule 20.9
Rule CCode_Prepo_009	-	Rule 20.8
Rule CCode_Macro_001	-	-
Rule CCode_Macro_003	-	Rule 20.7
Rule CCode_Macro_004	-	Rule 1.3

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule CCode_Macro_005	-	Rule 20.6
Rule CCode_Macro_006	-	Rule 20.4
Rule CCode_Macro_007	-	Rule 20.11
Rule CCode_Macro_008	-	Rule 20.12
Rule CCode_Essential_001	-	Rule 10.1
Rule CCode_Essential_002	-	Rule 10.2
Rule CCode_Essential_003	-	Rule 10.3
Rule CCode_Essential_004	-	Rule 10.4
Rule CCode_Essential_005	-	Rule 10.5
Rule CCode_Essential_006	-	Rule 10.6
Rule CCode_Essential_007	-	Rule 10.7
Rule CCode_Essential_008	-	Rule 10.8
Rule Abstr_Main_001	BSW006	Rule 1.3
Rule Abstr_Main_002	-	Dir 1.1, Rule 1.3, Rule 21.4, Rule 21.5, Rule 21.6, Rule 21.7, Rule 21.8, Rule 21.9, Rule 21.10, Rule 21.11, Rule 21.12
Rule Abstr_Main_003	-	-
Rule Abstr_Main_004	-	Rule 21.1, Rule 21.2
Rule Abstr_Main_005	-	Rule 8.14
Rule Abstr_NearFar_001	BSW00306, BSW00361, COMPILER035, COMPILER036, COMPILER040, COMPILER041, COMPILER046, COMPILER052, COMPILER055, COMPILER059	-
Rule Abstr_NearFar_002	-	-
Rule Abstr_NearFar_003	-	-
Rule Abstr_MemMap_001	MEMMAP021	-
Rule Abstr_MemMap_002	MEMMAP023	-
Rule Abstr_MemMap_003	-	-
Rule Abstr_MemMap_004	-	-
Rule Abstr_MemMap_005	-	-
Rule Abstr_MemMap_006	-	-
Rule Abstr_Inline_001	COMPILER060	Rule 8.10
Rule Abstr_Inline_002	-	-
Rule Abstr_Asm_001	-	Dir 4.2
Rule Abstr_DynMem_001	-	Dir 4.12, Rule 21.3, Rule 22.2
Rule Abstr_DynMem_002	-	-
Rule Abstr_PtrConv_001	-	Rule 11.1
Rule Abstr_PtrConv_002	-	Rule 11.2
Rule Abstr_PtrConv_003	-	Rule 11.6
Rule Abstr_PtrConv_004	-	Rule 11.3
Rule Abstr_PtrConv_005	-	Rule 11.8
Rule Abstr_PtrConv_006	-	Rule 11.4

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule Abstr_PtrConv_007	-	Rule 11.5
Rule Abstr_PtrConv_008	-	Rule 11.7
Rule Abstr_PtrArith_001	-	Rule 18.1
Rule Abstr_PtrArith_002	-	Rule 18.2
Rule Abstr_PtrArith_003	-	Rule 18.3
Rule Abstr_PtrArith_004	-	Rule 18.6
Rule Abstr_PtrArith_005	-	Rule 18.5
Rule Abstr_PtrArith_006	-	Rule 18.7
Rule Abstr_PtrArith_007	-	Rule 18.8
Rule Abstr_OSS_001	-	-
Rule CCode_Types_001	BSW00304, PLATFORM013, PLATFORM014, PLATFORM015, PLATFORM016, PLATFORM017, PLATFORM018, PLATFORM041, PLATFORM042, PLATFORM061	-
Rule CCode_Types_002	BSW00304	-
Rule CCode_Types_003	-	-
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 11.9
Rule CCode_ComStackTypes_001	COMTYPE005	-
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 5.8
Rule CDGNaming_002	BSW00441 (partial)	-
Rule CDGNaming_012	ECUConfiguration: ecuc_sws_2108	-

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule CDGNaming_003	BSW00307, BSW00305 (partial), BSW00310	-
Rule CDGNaming_004	-	Rule 5.6
Rule CDGNaming_005	-	Rule 5.7
Rule CDGNaming_006	-	-
Rule CDGNaming_007	-	Rule 5.9
Rule CDGNaming_008	-	-
Rule CDGNaming_009	-	Rule 5.3
Rule CDGNaming_010	-	Rule 5.1, Rule 5.2, Rule 5.4, Rule 5.5
Rule CDGNaming_011	BSW00300 (together with Rule CDGNaming_001)	-
Rule CDGNaming_013	BSW00392	Rule 7.2
Rule CDGNaming_014	BSW00392	Rule 7.3
Rule CDGNaming_015	BSW00392	-
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	Rule 20.9
Rule BSW_HeaderInc_010	BSW004	-
Rule BSW_HeaderInc_011	BSW00464	-
Rule BSW_ServiceRTE_001	-	-
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	-	Dir 4.7
Rule BSW_APIDesign_010	-	Rule 17.1
Rule BSW_APIDesign_011	-	Rule 17.2
Rule BSW_APIDesign_012	-	Rule 17.3

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule BSW_APIDesign_013	-	Rule 17.4
Rule BSW_APIDesign_019	-	Rule 17.5
Rule BSW_APIDesign_020	-	Rule 17.6
Rule BSW_APIDesign_021	-	Rule 17.7
Rule BSW_APIDesign_022	-	Rule 17.8
Rule BSW_APIDesign_014	-	-
Rule BSW_APIDesign_015	-	-
Rule BSW_APIDesign_016	-	Rule 2.7
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_ProcISR_007	-	-
Rule BSW_FuncObj_001	BSW00308	-
Rule BSW_FuncObj_002	-	Rule 8.1
Rule BSW_FuncObj_003	-	Rule 8.2
Rule BSW_FuncObj_004	-	Rule 8.6
Rule BSW_FuncObj_005	-	-
Rule BSW_FuncObj_006	-	Rule 8.2, Rule 8.4, Rule 8.5, Rule 17.3
Rule BSW_FuncObj_007	-	Rule 8.2, Rule 8.3
Rule BSW_FuncObj_008	-	Rule 8.8
Rule BSW_FuncObj_012	-	Rule 8.7
Rule BSW_FuncObj_009	-	Rule 8.13
Rule BSW_FuncObj_010	-	-
Rule BSW_FuncObj_011	BSW00309	-
Rule BSW_FuncObj_013	-	Rule 7.4
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 CCode_Style_001	-	-
Rule CCode_Style_002	-	-
Rule CCode_Style_003	-	-
Rule CCode_Style_004	-	-

Rule	Reference AUTOSAR	Reference MISRA C:2012
Rule CCode_Style_005	-	-
Rule CCode_Style_006	-	-
Rule CCode_Style_007	-	Rule 9.2, Rule 9.3
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	-	Rule 15.6
Rule CCode_BlockStyle_002	-	-
Rule CCode_BlockStyle_003	-	-
Rule CCode_BlockStyle_004	-	Rule 15.6
Rule CCode_Comments_001	-	-
Rule CCode_Comments_002	-	-
Rule CCode_Comments_003	-	Rule 1.3, Rule 3.1
Rule CCode_Comments_004	-	Dir 4.4
Rule CCode_Comments_005	-	-
Rule CCode_Comments_009	-	Rule 3.2
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, RS_BSWMD_00014	-
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 BSWMD_Impl_007	TPS_BSWMDT_4030, BSW00318	-
Rule BSWMD_Impl_008	-	-
Rule BSWMD_Split_001	-	-

Rule	Reference AUTOSAR	Reference MISRA C:2012
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_Entity_001	BSWMDT: Figure 6.2	-
Rule BSWMD_Entity_002	-	-
Rule BSWMD_Entity_003	BSWMDT: constr_4015, constr_4016, constr_4017, constr_4018, constr_4022	-
Rule BSWMD_Entity_004	TPS_BSWMDT_4019, BSWMDT: constr_4022, Table 6.3	-
Rule BSWMD_Entity_005	BSWMDT: Table 6.3	-
Rule BSWMD_Entity_006	TPS_BSWMDT_4018, BSWMDT: constr_4022, Table 6.3	-
Rule BSWMD_Entity_007	TPS_BSWMDT_4019, BSWMDT: constr_4022, Table 6.3	-
Rule BSWMD_Entity_008	TPS_BSWMDT_4019, BSWMDT: constr_4022, Table 6.3	-
Rule BSWMD_Entity_009	BSWMDT: Table 6.7, constr_4018	-
Rule BSWMD_Entry_001	-	-
Rule BSWMD_Entry_002	BSWMDT: Table 5.1, TPS_BSWMDT_4008	-
Rule BSWMD_Entry_009	-	-
Rule BSWMD_Entry_010	-	-
Rule BSWMD_Entry_011	-	-
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, Table 6.7, 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	Reference AUTOSAR	Reference MISRA C:2012
Rule BSWMD_EntryRetArg_008	TPS_BSWMDT_4009, BSWMDT: Table 5.5, Table 5.6, T-PS_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	BSWMDT: constr_4087	-
Rule BSWMD_EntryRetArg_013	BSWMDT: constr_4056	-
Rule BSWMD_EntryRetArg_014	BSWMDT: constr_4057	-
Rule BSWMD_EntryExport_001	TPS_BSWMDT_4002, BSWMDT: constr_4036	-
Rule BSWMD_EntryExport_002	TPS_BSWMDT_4002, BSWMDT: constr_4036	-
Rule BSWMD_Event_001	BSWMDT: Figure 6.4	-
Rule BSWMD_Event_002	-	-
Rule BSWMD_Event_003	BSWMDT: Table 6.12	-
Rule BSWMD_Event_004	BSWMDT: constr_4043, Table 6.13	-
Rule BSWMD_Event_005	BSWMDT: Table 6.18, Table 6.21, constr_4024	-
Rule BSWMD_Event_006	BSWMDT: constr_4024, constr_4025	-
Rule BSWMD_Event_007	BSWMDT: Table 6.19, constr_4026	-
Rule BSWMD_Event_008	BSWMDT: Table 6.17, constr_4023	-
Rule BSWMD_Event_009	BSWMDT: Table 6.16	-

In the introduction chapters following AUTOSAR requirements are used: BSW161, BSW162, BSW00342.

C List of Basic Software Modules

In [Table 191](#), [Table 192](#) and [Table 193](#) overview lists of all BSW and Library modules are shown based on AUTOSAR 4.1 Rev 1. All lists are taken from: [\[Document / URL: |SI8256|autosar\\$|SVN3-COPY|22_Releases|41_Release4.1R0001|02-Auxiliary|AUTOSAR_TR_BSWModuleList.pdf\]](Document / URL: |SI8256|autosar$|SVN3-COPY|22_Releases|41_Release4.1R0001|02-Auxiliary|AUTOSAR_TR_BSWModuleList.pdf).

Table 191 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 Tranceiver 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	DolP	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	Frlf	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	TcpIp	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 192 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
IFI Library	Ifl	209
MFI Library	Mfl	210
MFx Library	Mfx	211
IFx Library	Ifx	213

Table 193 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 194](#) and [Table 195](#) 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 BBM (was UBK) naming convention. The lists are therefore specific for BSW.

Table 194 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	MtrPerSecSqd	acceleration	Beschleunigung
ag	°	Rad	angle	Winkel
am	$Nm*s$	NwtMtrSec	angular momentum	Drehimpuls
amt	mol	Mol	amount of substance	Stoffmenge
aosc	mol/m^3	MolPerMtrCubd	amount of substance concentration	Stoffmengenkonzentration
ar	m^2	MtrSqd	area	Fläche
cnd	S/m	SPerMtr	conductivity	Leitfähigkeit
egy	J	Jou	heat (combustion heat)	Wärme
fac	1	No unit	factor	Faktor
frq	Hz	Hz	frequency	Frequenz
htc	$W/(K^*m^2)$	WattPerKelvinMtrSqd	heat transfer coefficient	Wärmeübergangskoeffizient
i	A	Ampr	electric current	Elektrischer Strom
jerk	m/s^3	MtrPerSecCubd	jerk	Ruck
len	m (mile)*	Mtr	length, distance	Länge, Strecke
m	g	Gr	mass	Masse
moi	$kg*m^2$	KiloGrMtrSqd	moment of inertia	Trägheitsmoment
n	rpm	Rpm	rotational speed	Drehzahl
p	Pa (bar, mm-Hg)*	Pa	pressure	Druck
psi	-	-	flux	Fluss
pwr	W	Watt	power	Leistung
q	$A*s$ (C)*	AmprSec, Coulmb	electric charge	elektrische Ladung
r	Ohm	Ohm	resistance	Widerstand
rad	rad	NoUnit	measurement of plain angle	Radiant
rat	%, 1	Perc	ratio, duty cycle	Verhältnis, Tastverhältnis
rd	m	Mtr	radius	Radius
re	-	NoUnit	reynolds number	Reynoldszahl
rho	kg/m^3	KiloGrPerMtrCubd	density	Dichte
spl	dB	DeciBel	sound pressure level	Schalldruckpegel
t	degC	DegCgrd	temperature	Temperatur
ti	s (min, h)*	Sec (Mins, Hr)	time, duration	Zeitpunkt, zeitliche Dauer
tq	Nm	NwtMtr	torque	Drehmoment
u	V	Volt	voltage	Spannung
v	m/s	MtrPerSec	velocity	Geschwindigkeit
vcos	m^2/s , $N*s/m^2$	MtrSqdPerSec, NwtSecPerMtr-Sqd	viscosity (kinematical), viscosity (dynamical)	Viskosität (kinematisch), Viskosität (dynamisch)
vol	m^3 (L)*	MtrCubd, Liter	volume	Volumen

pp-Name	Display Unit	Short-Name of Display Unit	Long Name (English)	Long Name (German)
w	Ws	WattSec	<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 BBM (was UBK) list of physical types.

Table 195 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 196](#) all HIS metrics are listed.

Table 196 Overview HIS Metrics

Name of Metric	Description	Value	Compliance
COMF Comment Density	Relationship of the number of comments (outside of and within functions) to the number of statements.	> 0.2	yes
PATH Number of paths	Number of non cyclic remark paths (i.e. minimum number of necessary cases of test)	1 – 80	yes
GOTO Number of goto statements	Number of goto statements	0	yes
v(G) Cyclomatic Complexity	In accordance with the Cyclomatic Number	1 – 10	yes
CALLING Number of calling functions	By how many subfunctions is this function called?	0 – 5	no * ¹
CALLS Number of called functions	How many different functions does this function call? Calling the same subfunction counts only once.	0 – 7	yes
PARAM Number of function parameters	How complex is the function interface?	0 – 5	yes
STMT Number of instructions per function	How complex is the function?	1 – 50	yes
LEVEL Number of call levels	Depth of nesting of a function.	0 – 4	yes
RETURN Number of return points	Number of return points within a function	0 – 1	yes
Si 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 * ²
VOCF Language scope	The language scope is an indicator of the cost of maintaining/changing functions. $\text{VOCF} = (N_1 + N_2) / (n_1 + n_2),$ where n1 = Number of different operators N1 = Sum of all operators n2 = Number of different operands N2 = Sum of all operands	1 – 4	no * ³
NOMV Number of MISRA violations	Total number of the violations of the Rules from MISRA	0	yes
NOMVPR 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
ap_cg_cycle Number of recursions	Call graph recursions	0	yes

05
10
15
10: *¹: 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.

20
20: *²: 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.

25
25: *³: 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 until now.

30
30: Reference: *Document "HIS Metrics Definition" [HIS]*.

35

40

45

50

55

60

65

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 Architecture / Name: G_Arch / 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 AUTOSAR Static View / Name: A_StaticView / 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_AppI]

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_IfI / 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 AUTOSAR Application Interfaces Documents

"Documents belonging to this Release cluster provide specifications of interfaces between applications and related explanatory material [\[R4Overview\]](#) ."

- ▶ [Document Application Interfaces User Guide / Name: EXP_AIUG / Publisher: AUTOSAR]
- ▶ [Document Application Interface Examples / Name: MOD_AISpecificationExamples / Publisher: AUTOSAR]

F.7 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]
- ▶ [Document HIS Metrics Definition / Name: HIS / URL: 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>]
- ▶ [Document Documentation Guideline / Name: DocuGuide / URL: http://abt-ismtwiki.abt.de.bosch.com/twiki/pub/Tools/-GuideLines/ECUDocu_cur.pdf]

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-120055 / 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.

The CEL guideline artefact was integrated before its own approval cycle was finished.

H Documentation

The BSW Coding Guideline is 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: http://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/]

05	262, 262, 262, 262, 262, 262, 262, 262, 262, 262, 262, 263, 263, 263, 263, 263, 263, 263, 263, 264, 264, 264, 264, 467, 471, 471, 471, 471, 471, 471, 471	272, 272, 272, 472, 472, 473, 473, 473, 473, 473	LinearCompuMethod 264, 274
10	IDENTICAL, LINEAR 270, 472	IDENTICAL, LINEAR 270, 472	LongName 234, 234, 234, 234, 234, 263, 270, 270, 273, 274, 275, 468, 468, 471, 472, 473
15	IdenticalCompuMethod 264	ImplementationDataType 225, 227, 227, 239, 239, 239, 239, 239, 239, 241, 243, 243, 243, 243, 243, 243, 243, 245, 245, 245, 245, 245, 245, 245, 251, 251, 251, 251, 251, 251, 251, 256, 262, 263, 277, 277, 308, 309, 309, 338, 339, 339, 339, 339, 339, 339, 340, 340, 340, 340, 340, 340, 340, 341, 343, 343, 343, 411, 412, 412, 412, 412, 413, 467, 467, 468, 468, 468, 468, 468, 469, 469, 469, 469, 469, 469, 470, 470, 470, 483, 483, 496, 496, 496, 496	LowerLimit 263, 278, 471, 474
20	ImplementationDataType 225, 227, 227, 239, 239, 239, 239, 239, 239, 241, 243, 243, 243, 243, 243, 243, 243, 245, 245, 245, 245, 245, 245, 245, 251, 251, 251, 251, 251, 251, 251, 256, 262, 263, 277, 277, 308, 309, 309, 338, 339, 339, 339, 339, 339, 339, 340, 340, 340, 340, 340, 340, 340, 341, 343, 343, 343, 411, 412, 412, 412, 412, 413, 467, 467, 468, 468, 468, 468, 468, 469, 469, 469, 469, 469, 469, 470, 470, 470, 483, 483, 496, 496, 496, 496	luminousIntensityExp 281	
25	ImplementationDataTypeElement 241, 243, 469	ImplementationDataTypeElement 241, 243, 469	M
30	IMPLEMENT-INTER-RUNNABLE-VA- RIABLE 254, 254, 470	IMPLEMENT-INTER-RUNNABLE-VA- RIABLE 254, 254, 470	massExp 281
35	Dest 406, 495	Dest 406, 495	maxDiff 263
40	DisplayFormat 223, 223, 223, 223, 223, 223, 223, 223, 227, 230, 230, 258, 258, 270, 270, 467, 467, 471, 472	ImplementationDataTypeElement 241, 243, 469	maxGradient 263
45	displayName 279, 279, 282, 476	ImplementationDataTypeElement 241, 243, 469	MaxNumberOfElements 232, 232, 232, 234, 234, 234, 468, 468, 468
50	E	IMPLEMENT-INTER-RUNNABLE-VA- RIABLE 254, 254, 470	MCAL 16, 17, 68, 68, 437
55	Element 231, 234, 234, 234, 237, 237, 237, 468, 468, 468, 468	InstantiationDataDefProps 258, 258, 258, 470	McSupport 313, 317, 403, 403
60	EXPLICIT-INTER-RUNNABLE-VA- RIABLE 254, 254, 470	internalConstr 262, 262, 262, 272, 274, 473, 473	molarAmountExp 281
65	F	INTERVAL-TYPE="CLOSED" 278, 474	Monotony 263
70	factorSiToUnit 282, 283, 283, 283, 475, 475	IntervalType 262, 262, 262	O
	factorSiToUnit 279, 280, 280, 280, 283	Introduction 270, 270, 472	OffsetSiToUnit 282, 475
	FibexElement 398, 493	IsDefault 407, 495	offsetSiToUnit 279, 280, 280, 280, 283, 283, 283, 475, 475
	FIXED-SIZE 232	IsGlobal 407, 495	P
	I	K	PackageRef 406, 407, 495
	ImplementationDataType 234, 237, 237, 237, 237, 237, 237, 237, 237, 238, 468	KeywordSets 415, 415, 497	ParameterAccess 256
	IDENTICAL 228, 264, 270, 271, 272, 272, 272, 272, 272, 272,	KeywordSets shall be of the form 415, 497	ParameterDataPrototype 247, 248, 248, 256, 317, 469
		L	perInstanceParameter 252
		lengthExp 281	PerInstanceParameters 316, 480
		LINEAR 264, 270, 271, 273, 273, 273, 273, 274, 274, 472, 472, 473, 473, 473, 474	physConstr 262, 262, 262, 262, 263, 263, 263, 263, 263, 263, 272, 274, 283, 471, 471, 471, 471, 473, 473, 475
			PhysicalDimension 282, 282, 282, 283, 283, 283, 283, 283, 283, 284, 284, 284, 475, 475, 476, 476, 476
			R
			RAT_FUNC 265, 270, 270, 271, 275, 275, 276, 276, 276, 276,

05	276, 472, 472, 472, 474, 474, 474	SwAddrMethods 413, 413, 414, 497	TAB_NOINTP 265
10	RatFuncCompuMethod 264	SwBaseType 243, 410, 411, 411, 411, 411, 412, 412, 496	temperatureExp 281
15	recommendedPackage 398	SwCalibrationAccess 222, 222, 222, 222, 227, 227, 230, 230, 237, 467, 467, 467, 468	TEXTTABLE 228, 251, 265, 265, 270, 270, 270, 271, 277, 277, 277, 278, 278, 278, 472, 472, 472, 475
20	ReferenceBase 394, 394, 395, 395, 395, 395, 406, 406, 406, 406, 406, 406, 406, 406, 406, 406, 406, 406, 406, 406, 407, 407, 407, 407, 407, 407, 407, 407, 409, 410, 410, 410, 410, 410, 410, 410, 410, 410, 410, 495, 495, 495, 495, 495, 495, 495, 497, 497, 497	SwComponentType 227, 404, 404, 467, 495, 495	TextTableCompuMethod 264, 278
25	S	SwDataDefProps 222, 222, 222, 222, 223, 223, 223, 223, 223, 223, 223, 227, 227, 227, 227, 230, 230, 230, 230, 230, 237, 237, 237, 237, 243, 256, 256, 256, 256, 256, 256, 256, 257, 257, 257, 263, 283, 333, 338, 339, 339, 340, 340, 340, 340, 340, 340, 341, 342, 342, 342, 343, 343, 343, 343, 467, 467, 467, 467, 467, 467, 467, 468, 468, 468, 468, 470, 470, 471, 475, 482, 483, 483, 483, 483, 483, 483, 483	timeExp 281
30	SCALE_LINEAR 265	SwImplPolicy 223, 223, 223, 223, 227, 227, 230, 230, 247, 247, 248, 248, 248, 248, 248, 248, 467, 467, 467, 467, 467, 469, 469	TPS_GST_00083 403
35	SCALE_LINEAR_AND_TEXTTABLE 265	SwPointerTargetProps 243, 243	U
40	SCALE_RAT_FUNC 265	SwRecordLayout 223, 223, 227, 228, 228, 230, 230, 467, 467	Unit 263, 265, 270, 270, 270, 270, 270, 270, 272, 272, 273, 273, 274, 274, 275, 276, 276, 279, 279, 279, 279, 279, 280, 281, 281, 281, 281, 282, 282, 282, 282, 282, 282, 283, 283, 283, 283, 285, 285, 393, 471, 472, 472, 472, 472, 472, 472, 472, 472, 472, 473, 473, 474, 474, 474, 475, 475, 475, 475, 475, 475, 476
45	SCALE_RATIONAL_AND_TEXTTABLE 265	SwRecordLayoutRef 223, 228	UnitGroup 285, 285, 285, 476
50	scaleConstr 263	SwRecordLayouts 412	UpperLimit 263, 278, 471, 474
55	ServiceSwComponentType 404, 495	SwValueBlockSize 230, 230	V
60	sharedParameter 252	Symbol 279, 279, 279, 279, 279, 475, 475	V 269, 269, 269, 269, 274, 276, 276, 473, 474, 474
65	Short-Name 230, 467	SwRecordLayoutRef 223, 228	VALUE 231
70	ShortName 222, 227, 227, 262, 264, 267, 270, 270, 270, 272, 272, 273, 274, 275, 276, 279, 281, 282, 284, 393, 398, 402, 405, 405, 407, 407, 471, 472, 472, 473, 473, 474, 475, 493, 494, 494, 494, 495, 271, 271, 473	SwValueBlockSize 230, 230	VARIABLE-SIZE 232
	Static Memory 254, 254, 254, 470	Symbol 279, 279, 279, 279, 279, 475, 475	VariableDataPrototype 248, 248, 248, 254, 254, 254, 256, 314, 314, 469, 470, 470
	StaticMemorys 313, 314, 314, 479	T	VT 269, 277, 278, 278, 279, 279, 279, 279, 474, 474, 474, 475, 475
	STRUCTURE 231, 234, 234, 234, 234, 468	constr_1015 256	constr_1175 270, 472
	SwAddrMethod 414, 414, 414, 414, 414, 497, 497, 497	constr_1024 267	constr_1191 262

AUTOSAR RULES

C

constr_1021 264, 270, 472

05	constr_2561 263, 471	TPS_GST_00049 398	TPS_GST_00085 398
	T	TPS_GST_00082 404	TPS_SWCT_02000 248, 248, 469,
10	TPS_GST_00017 396, 403	TPS_GST_00083 396	469 TR_PDN_00001 402, 403, 494

BOSCH RULES

15	A	CEL_064 245, 469	CEL_052 269, 472
20	ARPac_14 227, 467	CEL_001 282, 475	CEL_061 227, 467
25	ARPac_73 409, 497	CEL_011 397	CEL_063 263, 471
30	B	CEL_020 397	CEL_116 407
35	Blueprint_011 227, 467	CEL_031 282, 475	R
40	C	CEL_051 269, 472	Rule Static_0108 227, 467
45			
50	1	ApplicationDataTypes_Blueprint 398, 493	L
55	-	AUTOSAR_Rte 403	Lnr 274, 474
60	.arxml 393, 393	B	M
65	.../EcucModuleDefs 404	BLUEPRINT 402, 402, 402, 402, 493, 493, 494	m?{number}(p{number})?(Em?{number})? 271, 473
70	/AUTOSAR/EcucDefs 404	C	N
	/AUTOSAR_LinSM 405, 495	CentralElements 401, 493	NamePatterns 401, 493
	/RB/RBA/RBA_Bernd 405, 495	CUCEL_RecordLayouts 407	NamingConventions 401, 493
	<shortName of the Unit> 273, 473	D	P
	<shortName of the Unit>+Identcl 272, 473	DesignPatterns 401, 493	PackagePatterns 401, 493
	<shortName of Unit> 274, 474	DocumentationPatterns 401, 493	Platform 403
	<shortName of Unit>+Lnr+se- quenceNumber> 274, 474	E	PlatformTypes 403
	[1-9]([0-9])* 271, 473	EcucModuleConfigurationValuess 405, 494	PreconfiguredValues 405, 494
	_Blueprint 398, 493	ElementKind 407, 407, 495, 495	PTCEL_Units 407
	_Example 398, 493	EXAMPLE 402, 402, 402, 402, 493, 493, 494	R
	{Counter} 276, 279, 474, 475	F	Rat(_n{Nominator})*(_d{Denomi- nator})*_{Unit}(_Coun- ter)? 276, 474
	{Denominator} 276, 474	false 407, 407, 407, 495, 495, 495	RecommendedValues 405, 494
	{Nominator} 276, 474	I	S
	{sequenceNumber} 274, 474	ICS 402, 402, 402, 402, 493, 493, 494	Scope 407, 407, 495, 495
	{TextValue} 279, 475	Identcl 273, 473	STANDARD 402, 402, 402, 402, 403, 493, 493, 494
	{Unit} 276, 474		Std 403
	A		
	ApplicationDataTypes 227, 467		

05 SwArchitecture 401, 493

TTxt(_{TextValue})*(_Counter)? 278,
475

Control elements

10 |

InternalBehavior 246

P

ParameterDataPrototype 248, 252

MISRA_RULES

15 **M**

20 MISRA C:2012 Dir 1.1 22, 69, 430,

437

25 MISRA C:2012 Dir 4.11 23, 430

MISRA C:2012 Dir 4.12 83, 438

30 MISRA C:2012 Dir 4.13 155, 451

MISRA C:2012 Dir 4.2 82, 438

35 MISRA C:2012 Dir 4.3 82, 438

MISRA C:2012 Dir 4.4 162, 453

40 MISRA C:2012 Dir 4.5 112, 113,

444

45 MISRA C:2012 Dir 4.6 96, 98, 441,

441

MISRA C:2012 Dir 4.7 138, 448

MISRA C:2012 Rule 1.1 19, 430

MISRA C:2012 Rule 1.2 19, 430

45 MISRA C:2012 Rule 1.3 49, 52, 56,

68,

162,

434,

435,

437,

437,

453

50 MISRA C:2012 Rule 10.1 33, 33, 34,

61,

432,

432,

436

55 MISRA C:2012 Rule 10.2 63, 436

MISRA C:2012 Rule 10.3 64, 436

MISRA C:2012 Rule 10.4 65, 436

MISRA C:2012 Rule 10.5 65, 436

MISRA C:2012 Rule 10.6 66, 436

MISRA C:2012 Rule 10.7 67, 436

MISRA C:2012 Rule 10.8 68, 436

MISRA C:2012 Rule 11.1 84, 439

MISRA C:2012 Rule 11.2 85, 439

MISRA C:2012 Rule 11.3 86, 439

MISRA C:2012 Rule 11.4 87, 439

MISRA C:2012 Rule 11.5 87, 439

MISRA C:2012 Rule 11.6 85, 439

MISRA C:2012 Rule 11.7 87, 439

MISRA C:2012 Rule 11.8 86, 439

MISRA C:2012 Rule 11.9 101, 442

MISRA C:2012 Rule 12.1 29, 32,

431, 432

MISRA C:2012 Rule 12.2 34, 432

MISRA C:2012 Rule 12.3 34, 432

MISRA C:2012 Rule 12.4 35, 432

MISRA C:2012 Rule 13.1 28, 431

MISRA C:2012 Rule 13.2 30, 431

MISRA C:2012 Rule 13.3 35, 432

MISRA C:2012 Rule 13.4 37, 432

MISRA C:2012 Rule 13.5 32, 431

MISRA C:2012 Rule 13.6 31, 431

MISRA C:2012 Rule 14.1 40, 433

MISRA C:2012 Rule 14.2 39, 40,

433, 433

MISRA C:2012 Rule 14.3 41, 433

MISRA C:2012 Rule 14.4 38, 433

MISRA C:2012 Rule 15.1 43, 433

MISRA C:2012 Rule 15.4 43, 433

MISRA C:2012 Rule 15.6 160, 161,

452, 452

MISRA C:2012 Rule 15.7 43, 433

MISRA C:2012 Rule 16.1 44, 45,

434, 434

MISRA C:2012 Rule 16.2 45, 434

MISRA C:2012 Rule 16.3 45, 434

MISRA C:2012 Rule 16.4 46, 434

MISRA C:2012 Rule 16.5 46, 434

MISRA C:2012 Rule 16.6 47, 434

MISRA C:2012 Rule 16.7 47, 434

MISRA C:2012 Rule 17.1 138, 448

MISRA C:2012 Rule 17.2 138, 448

MISRA C:2012 Rule 17.3 139, 146,

448, 449

MISRA C:2012 Rule 17.4 139, 448

MISRA C:2012 Rule 17.5 139, 448

MISRA C:2012 Rule 17.6 140, 448

MISRA C:2012 Rule 17.7 140, 448

MISRA C:2012 Rule 17.8 141, 448

MISRA C:2012 Rule 18.1 88, 439

MISRA C:2012 Rule 18.2 89, 439

MISRA C:2012 Rule 18.3 89, 439

MISRA C:2012 Rule 18.5 90, 439

MISRA C:2012 Rule 18.6 90, 439

MISRA C:2012 Rule 18.7 91, 440

MISRA C:2012 Rule 18.8 91, 440

MISRA C:2012 Rule 19.1 48, 434

MISRA C:2012 Rule 19.2 48, 434

MISRA C:2012 Rule 2.1 41, 433

MISRA C:2012 Rule 2.2 42, 433

MISRA C:2012 Rule 2.6 43, 433

MISRA C:2012 Rule 2.7 143, 448

MISRA C:2012 Rule 20.1 50, 435

MISRA C:2012 Rule 20.11 57, 436

MISRA C:2012 Rule 20.12 58, 436

MISRA C:2012 Rule 20.13 52, 435

MISRA C:2012 Rule 20.14 53, 435

05	MISRA C:2012 Rule 20.2 51, 435	MISRA C:2012 Rule 22.2 83, 438	MISRA C:2012 Rule 7.2 120, 444
	MISRA C:2012 Rule 20.3 51, 435	MISRA C:2012 Rule 22.3 23, 430	MISRA C:2012 Rule 7.3 121, 444
10	MISRA C:2012 Rule 20.4 57, 436	MISRA C:2012 Rule 22.4 23, 430	MISRA C:2012 Rule 7.4 150, 450
	MISRA C:2012 Rule 20.5 52, 435	MISRA C:2012 Rule 22.5 23, 430	MISRA C:2012 Rule 8.1 145, 449
15	MISRA C:2012 Rule 20.6 57, 436	MISRA C:2012 Rule 22.6 23, 430	MISRA C:2012 Rule 8.10 82, 438
	MISRA C:2012 Rule 20.7 55, 435	MISRA C:2012 Rule 3.1 162, 453	MISRA C:2012 Rule 8.12 26, 431
20	MISRA C:2012 Rule 20.8 54, 435	MISRA C:2012 Rule 3.2 163, 453	MISRA C:2012 Rule 8.13 148, 450
	MISRA C:2012 Rule 20.9 54, 132, 435, 446	MISRA C:2012 Rule 4.1 21, 430	MISRA C:2012 Rule 8.14 71, 437
25	MISRA C:2012 Rule 21.1 70, 437	MISRA C:2012 Rule 4.2 20, 430	MISRA C:2012 Rule 8.2 146, 146, 147, 449, 449, 450
	MISRA C:2012 Rule 21.10 69, 437	MISRA C:2012 Rule 5.1 118, 444	MISRA C:2012 Rule 8.3 147, 450
30	MISRA C:2012 Rule 21.11 69, 437	MISRA C:2012 Rule 5.2 118, 444	MISRA C:2012 Rule 8.4 146, 449
	MISRA C:2012 Rule 21.12 69, 437	MISRA C:2012 Rule 5.3 118, 444	MISRA C:2012 Rule 8.5 146, 449
35	MISRA C:2012 Rule 21.2 70, 437	MISRA C:2012 Rule 5.4 119, 444	MISRA C:2012 Rule 8.6 146, 449
	MISRA C:2012 Rule 21.3 83, 438	MISRA C:2012 Rule 5.5 119, 444	MISRA C:2012 Rule 8.7 150, 450
40	MISRA C:2012 Rule 21.4 69, 437	MISRA C:2012 Rule 5.6 116, 443	MISRA C:2012 Rule 8.8 148, 450
	MISRA C:2012 Rule 21.5 69, 437	MISRA C:2012 Rule 5.7 116, 443	MISRA C:2012 Rule 9.1 25, 431
45	MISRA C:2012 Rule 21.6 69, 437	MISRA C:2012 Rule 5.8 111, 443	MISRA C:2012 Rule 9.2 158, 452
	MISRA C:2012 Rule 21.7 69, 437	MISRA C:2012 Rule 5.9 117, 444	MISRA C:2012 Rule 9.3 158, 452
50	MISRA C:2012 Rule 21.8 69, 437	MISRA C:2012 Rule 6.1 22, 430	MISRA C:2012 Rule 9.4 28, 431
	MISRA C:2012 Rule 21.9 69, 437	MISRA C:2012 Rule 6.2 22, 430	MISRA C:2012 Rule 9.5 29, 431
55		MISRA C:2012 Rule 7.1 21, 430	

Others

—
«atpSplittable» 305, 306, 306

Products

50	A	E	H
	A2L 297, 313, 313, 314, 316, 479, 480	ECU 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 17, 68, 102, 102, 109, 154, 220, 246, 451	hex file 220, 313, 313

Standard

60	A	E	H
	AUTOSAR 3.x 288	AUTOSAR 4.x 288	
	AUTOSAR 4.1 286, 288		

Tool

65	C	calibration tool 264, 264
----	----------	---------------------------

R

05	RTE 16, 17, 82, 129, 129, 130, 133, 133, 134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 135, 135, 135, 135, 135, 135, 135, 135, 135,	135, 135, 135, 136, 137, 137, 137, 137, 137, 147, 155, 155, 303, 311, 311, 312, 313, 313, 313, 313, 313, 316, 317, 318, 382,	438, 446, 446, 447, 447, 447, 447, 447, 447, 447, 447, 447, 480 RTE generator 303, 303, 303, 303
10			RTE generator 304, 304, 304, 304

Variables

15	C	Calibration parameter 250, 470
	Calibration and Measurement data 220	curves 220

M
maps 220

XML/SGML-Attributes

DEST 394

XML/SGML-Tags

	ARGUMENT 344, 484	BSW-SCHEDULABLE-ENTITY 349, 484
30	/AUTOSAR 290	AUTOSAR 290
	/BSW-BACKGROUND-EVENT 358, 487	
35	/BSW-CALLED-ENTITY 349, 484	B
	/BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT 358, 487	BASE-TYPE-REF 340, 483
40	/BSW-INTERNAL-TRIGGER-OCCURRED-EVENT 358, 487	BEHAVIOR-REF 304, 478
	/BSW-INTERRUPT-ENTITY 349, 484	BSW-BACKGROUND-EVENT 358, 487
45	/BSW-MODE-SWITCH-EVENT 358, 487	BSW-CALLED-ENTITY 349, 484
	/BSW-MODE-SWITCHED-ACK--EVENT 358, 487	BSW-EXTERNAL-TRIGGER-OCCURRED-EVENT 358, 487
50	/BSW-SCHEDULABLE-ENTITY 349, 484	BSW-INTERNAL-TRIGGER-OCCURRED-EVENT 358, 487
	/BSW-TIMING-EVENT 358, 487	BSW-INTERNAL-TRIGGERING--POINT-REF 351, 485
55	A	BSW-INTERNAL-TRIGGERING--POINT-REF-CONDITIONAL 351, 485
	ACCESSED-MODE-GROUPS 350, 485	BSW-INTERRUPT-ENTITY 349, 484
60	ACTIVATION 360, 487	BSW-MODE-SWITCH-EVENT 358, 487
	ACTIVATION-POINTS 351, 485	BSW-MODE-SWITCHED-ACK--EVENT 358, 487
65	ADDITIONAL-NATIVE-TYPE-QUALIFIER 343, 483	BSW-MODULE-ENTRY-REF 346, 346, 352, 484, 484, 486
	AR-RELEASE-VERSION 304, 478	BSW-MODULE-ENTRY-REF-CONDITIONAL 346, 346, 352, 484, 484, 486
		C
		CALL-TYPE 330, 481
		CALLED-ENTRYS 352, 486
		CATEGORY 297, 337, 477, 482
		CONTEXT-MODE-DECLARATION--GROUP-REF 361, 487
		D
		DATA-TYPE-MAPPING-REF 300, 477
		DATA-TYPE-MAPPING-REFS 300, 477
		DESC 173, 328, 336, 456, 482, 484
		DIRECTION 338, 482
		E
		ECUC-DEFINITION-COLLECTION 172, 455
		ECUC-MODULE-DEF 172, 455
		ECUC-VALUE-COLLECTION 170, 455
		ENTITYS 348, 349, 484
		EVENT-SOURCE-REF 363, 488
		EVENTS 358, 487

05	EXECUTION-CONTEXT 330, 481	MODE-DECLARATION-GROUP-- PROTOTYPE-REF 350, 354, 485, 486	RETURN-TYPE 344, 484
	F		S
10	FUNCTION-POINTER-SIGNATURE-- REF 341, 483	MODE-DECLARATION-GROUP-- PROTOTYPE-REF-CON- DITIONAL 350, 354, 485, 486	SERVICE-ID 329, 481
	I	MODE-GROUP-REF 362, 488	SHORT-NAME 172, 172, 456, 456
15	IMPLEMENTATION-DATA-TYPE-REF 338, 339, 482, 483	MODE-IREF 361, 487	STARTS-ON-EVENT-REF 359, 487
	IMPLEMENTED-ENTRY-REF 350, 485	MODE-IREFS 361, 487	SW-IMPL-POLICY 342, 483
20	INTERRUPT-CATEGORY 354, 486	MODULE-ID 298, 477	SW-SERVICE-IMPL-POLICY 331, 482
	INTRODUCTION 173, 329, 337, 456, 482, 484	MULTIPLE-CONFIGURATION-CON- TAINER 179, 460	SW-VERSION 303, 478
25	IS-REENTRANT 329, 481	O	T
	IS-SYNCHRONOUS 330, 481	ORIGIN 173, 174, 456	TARGET-CATEGORY 343, 483
	ISSUED-TRIGGERS 353, 486	OUTGOING-CALLBACKS 346, 484	TARGET-MODE-REF 361, 487
30	L	P	TRIGGER-REF 353, 362, 486, 488
	LONG-NAME 173, 336, 456, 484	PERIOD 359, 487	TRIGGER-REF-CONDITIONAL 353, 486
	M	PROGRAMMING-LANGUAGE 303, 478	U
35	MANAGED-MODE-GROUPS 354, 486	PROVIDED-ENTRYS 346, 484	UNIT-REF 394
	R	V	VENDOR-API-INFIX 302, 478
40			VENDOR-ID 302, 478
45			
50			
55			
60			
65			
70			

J Version Information

1 version overview

Version	Date	Publisher	State
1.10	2016-01-31	Volker Kairies (CDG-SMT/ESM1)	released for CDG-SMT
1.9	2015-07-31	Volker Kairies (CDG-SMT/ESM1)	released for CDG-SMT
1.8	2015-01-30	Volker Kairies (CDG-SMT/ESM1)	Released for CDG-SMT
1.7	2014-07-31	Volker Kairies (CDG-SMT/ESM1)	Released for CDG-SMT
1.6	2014-02-21	Volker Kairies (CDG-SMT/ESM1)	released
1.5	2013-10-17	Volker Kairies (CDG-SMT/ESM1)	released
1.4	2012-05-21	Volker Kairies (CDG-SMT/ESM1)	released
1.3	2011-12-09	Volker Kairies (CDG-SMT/ESM1)	released
1.2	2011-03-25	Volker Kairies (CDG-SMT/ESM1)	released
1.1	2010-12-17	Volker Kairies (CDG-SMT/ESM1)	released
1.0	2010-12-01	Volker Kairies (CDG-SMT/ESM1)	released
0.5		Volker Kairies (CDG-SMT/ESM1)	under development
0.4		Volker Kairies (CDG-SMT/ESM1)	under development
0.3		Volker Kairies (CDG-SMT/ESM1)	under development
0.2		Volker Kairies (CDG-SMT/ESM1)	under development
0.1		Volker Kairies (CDG-SMT/ESM1)	under development

2 modifications

Version	Change	Related to
1.10	ARMETH-119781: Integration of EOC artefact V1.0	Content
1.9	ARMETH-84847: Rule CDGNaming_014: Change the example to reduce confusion ARMETH-84851: Rule CCode_MisraHIS_002: Modify the comment for MISRA rules violation ARMETH-84853: Rule CDGNaming_014: Expand the description and the example for 64 bit and LL ARMETH-84876: Consider MISRA C 2012 rule 2.7 and 11.9 ARMETH-84891: Rule CDGNaming_003: Existence of an underscore is not clearly specified ARMETH-84920: Rule BSW_APIDesign_016: Specify the rule more precisely ARMETH-84922: Rule BSW_Sched_001: Expand description for locks using Macro interfaces ARMETH-84928: Integration of Pragma Concept Artifact V1.3.0 AML 84934: Rule Abstr_PtrArith_006: The long name of the rule is not correct ARMETH-84936: Rule BSW_Sched_001: Longname of the rule is missing ARMETH-84962: Handling of #warning preprocessor directive ARMETH-85154: Mark derived rules as explained in authoring guideline ARMETH-86775: Rule CCode_Style_009: Limit the rule to a block of preprocessor lines	Content

Version	Change	Related to
△	ARMETH-87009: Replace the abbreviation UBK by BBM	Content
	Integration of CEL Guideline Artefact V1.5	Content
	ARMETH-96526: Integration of Data Specification Guideline Artefact V1.8	Content
	Derive chapter Scope from General Aspects Guideline V1.3	Content
	More details to the listed changes --> see change description in SVN and CDG migration repository	Content
1.8	ARMETH-996: Add rule to suffix float constants with "f"; New rule: CDGNaming_015	Content
	ARMETH-1318: New rule to specify that the package item inside a perl module is set identical to the name of the perl module; New rule: Perl_021	Content
	ARMETH-1325: BSWMD_MCSupport_004: Review finding: Adapt the naming convention for measured values; New rule: BSWMD_MCSupport_008; Changed rule: BSWMD_MCSupport_004	Content
	ARMETH-1333: Rule CCode_MisraHIS_004: Adapt the comment for HIS Metric Violations; Changed rule: CCode_MisraHIS_004	Content
	ARMETH-1348: Rule BSWMD_MCSupport_002: C-Scope of Measurement Variables shall be defined as global; Changed rule: BSWMD_MCSupport_002	Content
	ARMETH-1362: Rule BSWMD_EntryRetArg_012: Modify rule to new definitions in AUTOSAR; Changed rule: BSWMD_EntryRetArg_012	Content
	ARMETH-1382: Rule BSW_VersionInfo_005: Expand the explanation of the rule that it is only valid for AR based modules; Changed rule: BSW_VersionInfo_005	Content
	ARMETH-1415: Introduction of MISRA C 2012 standard to the BSW Coding Guideline; Many rules are changed and new	Content
	ARMETH-1464: Update chapter "Used Keywords" from new version of generic aspects guideline; Chapter a.1.1 and a.1.2 changed	Content
	ARMETH-1467: Rule BSWMD_ModuleDesc_004: Define the term BSW_CLUSTER more precised; Changed rule: BSWMD_ModuleDesc_004	Content
	ARMETH-1526: Integration of Central Elements guideline artefact V1.4	Content
	ARMETH-1527: Integration of Data Description guideline artefact V1.7	Content
	More detailed list of changes --> see change description in SVN and CDG migration repository	Content
	ARMETH-703: isModuleExistent in oAW; Changed rule OAW_Auxilliary_001	Content
	ARMETH-707: Keine Aufrufe von Dem/Fim (bzw. DSM) in Interrupt Service Routinen; New rule BSW_ProcISR_007	Content
▽	ARMETH-1161: Consider review findings in chapter "Scheduling of BSW via RTE"; Changed chapter 8.3.2	Content
	ARMETH-1174: Describe the use of documentation-related ARXML elements in context of BswModuleEntrys; Changed Rule BSWMD_Entry_009 to Rule BSWMD_Entry_011 and Rule BSWMD_EntryRetArg_015 to Rule BSWMD_EntryRetArg_017	Content
	ARMETH-1195: Chapter 8.2.1.3 BswImplementation: Reference to BswInternalBehavior in ARXML example is incomplete; Changed chapter 8.2.1.3	Content
	ARMETH-1196: What is venorId and vendorApiInfix of RB, RBA, RBM etc.?; Changed rule BSWMD_Impl_003	Content
	ARMETH-1214: oAW coding guideline: Prohibit 'set' method for oAW scripts; New rule OAW_Xtend_010	Content
	ARMETH-1215: oAW coding guideline: Make Prepare / Generate / Forwarder interfaces for oAW consistent to interfaces in Perl; Changed rule E-CUC_OP_001, rule OAW_XpandIdGen_001 and rule OAW_XpandIdGen_002	Content

Version	Change	Related to
△	ARMETH-1218: Correction: In the summary report rule Abstr_CPP_005 is listed as Abstr_CPP_004. Rule Abstr_CPP_001, BSW_HeaderInc_003, BSW_ServiceRTE_002, OAW_Xtend_008 and OAW_Xtend_007 are on the wrong position; Changed rules as listed ARMETH-1220: Removing leading white spaces in paragraphs; Changed some chapters and rules ARMETH-1226: Integrate AUTOSAR guideline artifact CentralElements V1.3 to BSW Coding Guideline V1.7; Changed chapter 10 ARMETH-1247: Rule BSWMD_Entry_007: Update / expand description of interrupt categories; Changed rule BSWMD_Entry_007 ARMETH-1250: Mark Naming Convention related rules as such; Various rules changed ARMETH-1252: Describe the export of BswModuleEntrys via ProvidedEntries and OutgoingCallbacks; New chapter 8.3.4.1.3 ARMETH-1253: Add a new chapter which describes the model point of view of definition of APIs; New chapter 8.2.3 ARMETH-1256: Integrate Pragma Concept Artefact V1.2.6 to BSW Coding Guideline; Updated chapters 3.2.3 and 3.2.4 ARMETH-1259: Integration of ARPackage guideline artefact V1.5.2; Updated Chapter 9 ARMETH-1260: Specify a new rule set for the description of the properties of a code fragment (BswModuleEntitys); New chapter 8.3.4.2, New rules BSWMD_Entity_001 to BSWMD_Entity_009; removed rules BSWMD_Sched_001 to BSWMD_Sched_003 ARMETH-1261: Reorganization of the chapter "Scheduling of BSW via RTE"; Redesigned chapter 8.3.2 and 8.3.4 ARMETH-1284: Integrate AUTOSAR guideline artifact DataDescription V1.6 to BSW Coding Guideline V1.7; Updated chapter 7 ARMETH-1285: Rule OAW_General_002: Update rule to make it more precise; Updated rule OAW_General_002 ARMETH-1286: Rule OAW_Xtend_003: Add hint that Aspect is not recommended; Update rule OAW_Xtend_003 More detailed list of changes --> see change description in SVN and CDG migration repository	Content
1.6	ARMETH-782: BSWMD shall always contain module's longname; New rule BSWMD_ModuleDesc_003 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 ARMETH-1030: BSWMD: Add CodeDescriptors as sub-element from BSW-IMPLEMENTATION; New Rule BSWMD_Impl_004 ARMETH-1031: BSWMD: Specify rules for handling of splitable; New chapter 8.2.2 and new rules Rule BSWMD_Split_001 to Rule BSWMD_Split_006chapter 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 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 ARMETH-1038: Rule CCode_Prepro_005: Show #ifdef and #ifndef in the example because both are not forbitten; Modified Rule CCode_Prepro_005	Content
▽		

Version	Change	Related to
△	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, Update 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
1.5	Update of Abstract chapter with "Consideration of BSW Coding Guideline"	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 "Auxillary 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 Adressing 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

Version	Change	Related to
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 Adressing 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: (http://abt-ismtwiki.abt.de.bosch.com/twiki/bin/view-/Tools/GuideLines)	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
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