Document Title	Software Component Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	62

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History

Date	Release	Changed by	Description
2022-11-24	R22-11	AUTOSAR Release Management	• Support for service discovery • Clarification of the role of SwBaseType • minor corrections / clarifications / editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Introduction of imposition times for constraints • Clean-up of diagnostics service needs • New writing strategy for NvRAM • minor corrections / clarifications / editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• In-place Strategy for Array Data Type • New Service Use Cases • Optimization of Return Type for RTE APIs • minor corrections / clarifications / editorial changes

2019-11-28	R19-11	AUTOSAR Release Management	• Support for optimized access to coding data • Support for meta-data on application level • Support for optimized storing of bitfields in NvRAM • minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Support for optional elements in structured data types • Improved description of service use cases • minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2017-12-08	4.3.1	AUTOSAR Release Management	• minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Improved support for Unions • Improved upstream mapping • Improved description of service use cases • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2014-10-31	4.2.1	AUTOSAR Release Management	• Efficient NV data handling • Introduction of data transformation • Support for variable-size Arrays of arbitrary data types • Support for ASIL/QM development • Minor corrections / clarifications / editorial changes; For details please refer to the BWCStatement
2014-03-31	4.1.3	AUTOSAR Release Management	• Various fixes and clarifications

2013-10-31	4.1.2	AUTOSAR Release Management	• Various fixes and clarifications
2013-03-15	4.1.1	AUTOSAR Administration	• Introduction of PRPortPrototype • Definition of implicit communication behavior • Support for the formal analysis of resource locking • Introduction of refined scheduling of RunnableEntitys • Get information about activating RTEEvent • Connection of Mode Managers and Mode Users with different number of ModeDeclarations • Support activation of RunnableEntitys on remote ECUs • Support for ModeTransition • Support for the definition of the network representation of composite data types • ServiceNeeds for diagnostics over IP • Various fixes and clarifications
2011-12-22	4.0.3	AUTOSAR Administration	• Added CompuMethod categories SCALE_LINEAR_AND_TEXTTABLE and SCALE_RATIONAL_AND_TEXTTABLE (table 5.73) • Clarification concerning the usage of invalid values • Revised support for data filters • Support for partial networking • Support for the specification of local connections between software-components • Improved description of service needs • Change history of constraints and specification items • Miscellaneous improvements and clarifications • “Support for Standardization” moved to Standardization Template [1]

2011-04-15	4.0.2	AUTOSAR Administration	• Remove restriction on data type of inter-runnable variables • Rework end-to-end communication protection • Add more constraints on the usage of the meta-model • Various fixes and clarifications
2009-12-18	4.0.1	AUTOSAR Administration	• New requirements tracing table • Support for fixed data exchange • Implementation of meta-model cleanup • Fundamental revision of the data type concept • Support for variant handling • Support for end-to-end communication protection • Support for documentation • Support for stopping and restarting of software-components • Support for triggered events • Support for explicit mapping of interface elements • Revised concept of mode management • Support for integrity and scaling at ports • Support for standardization within AUTOSAR
2008-08-13	3.1.1	AUTOSAR Administration	• Improved support for on-board diagnostics • Small layout adaptations made
2007-12-21	3.0.1	AUTOSAR Administration	• Improved support for measurement and calibration • Improved semantics of delegation ports • Introduction of abstract memory classes • Document meta information extended • Small layout adaptations made

2007-01-24	2.1.15	AUTOSAR Administration	• Harmonization of the document with other specifications (e.g. RTE) • Introduction of a new concept to support calibration and measurement - harmonized with RTE • Description of needs of the Software Component Template toward AUTOSAR services and of the interaction of the Software Component Template and services (on XML level) • Legal disclaimer revised • Release notes added • “Advice for users” added • “Revision information” added
2006-05-16	2.0.0	AUTOSAR Administration	Second
2005-05-31	1.0.0	AUTOSAR Administration	Initial release

Contents

1 Introduction 30

1.1 Overview 30
1.2 Scope 30
1.3 Organization of the Meta-Model 31
1.4 Structure of the Template 33
1.4.1 Description of Software Components on VFB Level 33
1.4.2 Description of Software Components on RTE Level 33
1.4.3 Descriptions of Software Components on Implementation Level 34
1.5 Abbreviations 34
1.6 Document Conventions 35
1.7 Imposition Times of Constraints 37
1.8 Requirements Tracing 39

2 Conceptual Aspects 47

2.1 Introduction 47
2.2 Measurement and Calibration 47
2.2.1 Basic Approach of Measurement and Calibration 47
2.2.2 Calibration Parameters Overview 47
2.2.3 Using Calibration Parameters 48
2.2.3.1 Sharing Calibration Parameters within Compositions 48
2.2.3.2 Sharing Calibration Parameters between SwComponentPrototypes of the Same SwComponentType . . 50
2.2.3.3 Providing Instance Individual Characteristic Data . . 51
2.3 Runtime and Data Consistency Aspects 52
2.3.1 Background: the Issues 52
2.3.1.1 Mutual Exclusion with Semaphores 52
2.3.1.2 Interrupt Disabling 53
2.3.1.3 Priority Ceiling 53
2.3.1.4 Implicit Communication by Means of Variable Copies 53
2.3.2 Data Consistency at Runtime 54
2.3.3 Modeling Aspects of Data Consistency 55
2.4 Variant Handling in the Software Component Template 55
2.5 Communication Specification of Composition Component Types 57
2.5.1 Rationale 58
2.6 PRPortPrototype 59
2.6.1 Use Case 1 59
2.6.2 Use Case 2 60
2.6.3 Use Case 3 60
2.6.4 Solution 61
2.7 Variable-size Array Data Types 61
2.7.1 Overview and Use cases 61
2.7.1.1 “Old-world” dynamic-size Arrays 62
2.7.1.2 “New-world” variable-size Arrays 63
2.7.2 Modeling Aspects regarding Application Data Types 66

2.7.3 Modeling Aspects regarding Implementation Data Types . . . 66
2.8 Optional Elements in Structures 68
2.8.1 Background 68

3 Overview: Software Components, Ports, and Interfaces 70

3.1 Introduction 70
3.2 Software Component 71
3.2.1 Overview 71
3.2.2 PortPrototype 72
3.2.3 AtomicSwComponentType 76
3.2.4 ParameterSwComponentType 78
3.2.5 Symbolic Name of a Software-Component 79
3.3 Composition 80
3.3.1 Overview 80
3.3.2 SwComponentPrototype 81
3.3.3 Connectors 84
3.3.4 Instantiation-specific RTEEvents 90
3.4 Port Interface 91

4 Details: Software Components, Ports, and Interfaces 98

4.1 Introduction 98
4.2 Port Interface Details 98
4.2.1 Introduction 98
4.2.2 Sender Receiver Communication 99
4.2.2.1 Sender Receiver Interface 99
4.2.2.2 Communication Patterns for Sender Receiver Communication 101
4.2.2.3 Invalidation Policy 102
4.2.2.4 Meta-data on the Application Software Level 103
4.2.3 Client Server Communication 106
4.2.3.1 Client Server Interface 106
4.2.3.2 Error Handling in Client/Server Communication . . . 112
4.2.4 External Trigger Event Communication 114
4.2.5 Communication of Modes 117
4.2.5.1 Mode Switch Interface 117
4.2.5.2 Data Types used for Mode Communication 120
4.2.5.3 Relevance for Measurement and Calibration 122
4.2.6 Parameter Communication 122
4.3 PortInterface Mapping and Data Scaling 123
4.3.1 PortInterface Mapping 125
4.3.1.1 Mapping of Sender Receiver Interface, Parameter Interface and Non Volatile Data Interface Elements . . 128
4.3.1.2 Mapping of Client Server Interface Elements 131
4.3.1.3 Mapping of Mode Interface Elements 135
4.3.1.4 Mapping of Trigger Interface Elements 139
4.3.1.5 Mapping of Elements of a composite Data Type . . . 140
4.3.2 Data Conversion 146

4.3.2.1 Linear Data Scaling 146
4.3.2.2 Table Conversion 147
4.3.3 Relevance for Data Transformation 152
4.4 Port Annotation 156
4.4.1 Introduction 156
4.4.2 SenderReceiverAnnotation 156
4.4.3 ClientServerAnnotation 160
4.4.4 Annotation for the I/O Hardware Abstraction Layer 161
4.4.5 Parameter Port Annotation 164
4.4.6 Mode Port Annotation 164
4.4.7 Trigger Port Annotation 165
4.4.8 Non Volatile Data Port Annotation 166
4.4.9 Delegated Port Annotations 166
4.4.10 General Annotation 167
4.5 Communication Specification 168
4.5.1 Communication Specification for Sender-Receiver Communication 173
4.5.1.1 Receiver ComSpec 173
4.5.1.2 Sender ComSpec 181
4.5.1.3 Data Filter 186
4.5.1.4 Communication between Application and NV Block Software Components 188
4.5.1.5 Communication behavior to be implemented by the Software Component 190
4.5.2 Communication Specification for Client-Server Communication 191
4.5.2.1 Client ComSpec 191
4.5.2.2 Server ComSpec 192
4.5.3 Communication Specification for Mode Switch Communication 194
4.5.3.1 Mode Switch Sender ComSpec 194
4.5.3.2 Mode Switch Receiver ComSpec 195
4.5.4 Communication Specification for Parameters 196
4.5.5 Communication Specification for NV Data 198
4.5.5.1 NV Require ComSpec 198
4.5.5.2 NV Provide ComSpec 199
4.5.6 Configuration of Data Transformation 200
4.6 Port Groups within Component Types 206
4.7 End to End Protection 208
4.8 Partial Networking 220
4.8.1 VFC Control Ports 220
4.8.2 VFC Status Ports 221
4.9 Formal Definition of implicit Communication Behavior 222
4.9.1 Consistency Needs on Receiver Side 227
4.9.2 Consistency Needs on Sender Side 228
4.9.3 Consistency Needs for Senders and receivers of the same Data inside on RunnableEntityGroup 229

5 Data Description 230

5.1 Introduction 230
5.2 Data Types 234
5.2.1 Overview 234
5.2.2 Data Type Mapping 236
5.2.3 Data Categories 240
5.2.4 Application Data Type 244
5.2.4.1 Application Primitive Data Types 247
5.2.4.2 Application Composite Data Types 259
5.2.5 Implementation Data Type 273
5.2.5.1 Overview 273
5.2.5.2 Modeling of Type Reference using Implementation Data Type 279
5.2.5.3 Modeling of Structure using Implementation Data Type 280
5.2.5.4 Modeling of Union using Implementation Data Type . 283
5.2.5.5 Modeling of Array using Implementation Data Type . 283
5.2.5.6 Modeling of Pointer using Implementation Data Type 293
5.2.6 Base Type 298
5.2.6.1 Big Picture 298
5.2.6.2 Category 302
5.2.6.3 Native Declaration 302
5.2.6.4 Base Type Size 303
5.2.6.5 Base Type Encoding 304
5.2.6.6 Byte Order and Memory Alignment 306
5.2.7 Data Type Terminology 308
5.2.7.1 Primitive Type 308
5.2.7.2 Compound Primitive Data Type 309
5.2.7.3 Integral Primitive Type 309
5.2.7.4 Variable-Size Array Data Type 311
5.2.7.5 Wrapped Union Data Type 311
5.2.7.6 Optional Element Structure 314
5.3 Data Prototypes 314
5.3.1 Overview 314
5.3.2 Data Constraints for DataPrototypes typed by Array DataTypes 321
5.3.3 Reference to Data Prototypes 324
5.3.3.1 AUTOSAR Variable Ref 325
5.3.3.2 AUTOSAR Parameter Ref 326
5.3.3.3 Modeling Approach 328
5.3.3.4 Access into VariableDataPrototype typed by an ImplementationDataType 330
5.3.3.5 Access into ParameterDataPrototype typed by an ImplementationDataType 333
5.4 Properties of Data Definitions 335
5.4.1 Overview 335
5.4.2 Invalid Value 350
5.4.3 Properties for Measurement 358
5.4.4 Properties of Curves and Maps 360

5.4.4.1 Specification of grouped Axes 368
5.4.4.2 Specification of fix Axes 371
5.4.5 Setting an Axis Input Value 374
5.4.6 Setting a Group Axis 380
5.4.7 Specifying Data Dependencies 384
5.4.8 Precedence of data properties with respect to data elements, axis elements, computation methods, units 387
5.5 Elements used in Properties of Data Definitions 392
5.5.1 Computation Methods 392
5.5.1.1 Category Values in the context of a CompuMethod . 403
5.5.1.2 Applicability of Attributes in the context of a CompuMethod 404
5.5.1.3 CompuMethod and AutosarDataType 406
5.5.1.4 Example for Enumeration 408
5.5.1.5 Example for Linear Conversion 409
5.5.1.6 Example for Linear Conversion with texttable 409
5.5.1.7 Example for conversion specified by a rational function 410
5.5.1.8 Example for BITFIELD_TEXTTABLE 411
5.5.2 Physical Units, Physical Dimensions and Unit Groups 414
5.5.3 Data Constraints 421
5.5.3.1 Physical Limits 428
5.5.4 Addressing Methods 429
5.5.5 Record Layouts 437
5.5.5.1 Specifying Record Layouts 438
5.5.5.2 RecordLayouts and DataTypes 446
5.5.5.3 Record Layouts and Interpolation Routines 452
5.5.6 Display Presentation 453
5.6 Specification of Constant Values 455
5.6.1 Overview 455
5.6.2 Reference to Constant 463
5.6.3 Constant Specification Mapping 464
5.6.4 Values for Variable-Size Array 466
5.6.5 Values for Compound Primitive Data Types 468
5.6.6 Values for TEXTTABLE 477
5.6.7 Values for BITFIELD_TEXTTABLE 478
5.6.8 Values for Strings 479
5.6.8.1 String is modeled by ApplicationDataType 479
5.6.8.2 String is modeled by ImplementationDataType . . . 479
5.6.9 Specification of Values based on Rules 479
5.6.9.1 Support for primitive Data Types 479
5.6.9.2 Support for composite Data Types 489
5.6.9.3 Support for compound primitive Data Types 499
5.6.10 Examples 500
5.6.10.1 Example for Constant Specification for CURVE . . . 500
5.6.10.2 Example for Constant Specification for MAP 501

5.6.10.3 Example for Constant Specification for MAP with two STD_AXIS . . . 502
5.6.10.4 Example for Constant Specification for COM_AXIS . 503
5.6.10.5 Example for Constant Specification for an Array of compound primitive Objects . . . 504
5.7 Initial Values . . . 507
5.7.1 Overview . . . 507
5.7.2 Initial Value Representation . . . 508
5.7.3 Initial Values For CalibrationParameters . . . 510
5.7.4 Initial Value for optional Element . . . 511
5.7.4.1 Initial Value for optional ApplicationRecordElement . 511
5.7.4.2 Initial Value for optional ImplementationDataType-Element . . . 512

6 Compatibility 514

6.1 Introduction . . . 514
6.2 Compatibility of Data Types . . . 514
6.2.1 ApplicationDataType . . . 514
6.2.1.1 ApplicationPrimitiveDataType . . . 514
6.2.1.2 ApplicationCompositeDataType . . . 515
6.2.2 ImplementationDataType . . . 516
6.2.3 Compatibility of SwBaseType . . . 518
6.2.4 Compatibility of SwDataDefProps . . . 518
6.2.4.1 Compatibility of Units . . . 519
6.2.4.2 Compatibility of PhysicalDimensions . . . 520
6.2.4.3 Compatibility of Data Constraints . . . 521
6.2.4.4 Compatibility in case of ImplementationDataType . . 522
6.2.4.5 Compatibility of CompuMethods . . . 523
6.2.5 Compatibility of ApplicationDataType and Implementation-DataType . . . 525
6.2.5.1 General Rule . . . 526
6.2.5.2 Category VALUE . . . 526
6.2.5.3 Category ARRAY, VAL_BLK . . . 527
6.2.5.4 Category STRUCTURE . . . 527
6.2.5.5 Category BOOLEAN . . . 528
6.2.5.6 Category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 . . . 528
6.2.5.7 Forbidden Mappings . . . 529
6.3 Compatibility of Variable Data Prototypes and Parameter Data Prototypes 530
6.4 Compatibility of Sender Receiver Interfaces, Parameter Interfaces and Non Volatile Data Interfaces . . . 532
6.4.1 Connection of Required and Provided Port via AssemblySwConnector . . . 532
6.4.2 Connection of Inner and Outer Port via DelegationSwConnector . . . 533

6.4.3 Connection of Required and Provided Port via PassThroughSwConnector . . . 534
6.4.4 Compatibility of ParameterDataPrototype and VariableDataPrototype depending on PortInterface Type . . . 535
6.5 Compatibility of Mode Switch Interfaces . . . 536
6.5.1 Connection of Required and Provided Port via AssemblySwConnector . . . 536
6.5.2 Connection of Inner and Outer Port via DelegationSwConnector . . . 537
6.5.3 Connection of Outer and Outer Port via PassThroughSwConnector . . . 537
6.6 Compatibility of Mode Declaration Group Prototypes . . . 538
6.7 Compatibility of Mode Declaration Groups . . . 538
6.8 Compatibility of Argument Prototypes . . . 540
6.9 Compatibility of Application Errors . . . 540
6.10 Compatibility of Client/Server Operations . . . 540
6.11 Compatibility of Client Server Interfaces . . . 541
6.11.1 Connection of Required and Provided Port via AssemblySwConnector . . . 541
6.11.2 Connection of Inner and Outer Port via DelegationSwConnector . . . 542
6.11.3 Connection of Outer and Outer Port via PassThroughSwConnector . . . 542
6.12 Compatibility of Trigger Interfaces . . . 543
6.12.1 Connection of Required and Provided Port via AssemblySwConnector . . . 543
6.12.2 Connection of Inner and Outer Port via DelegationSwConnector . . . 544
6.12.3 Connection of Outer and Outer Port via PassThroughSwConnector . . . 544
6.13 Compatibility of Trigger . . . 545
6.14 Entire Delegation of a Provided Port Prototype . . . 545
6.14.1 Split and Merge of PortInterface Elements . . . 546
6.15 Compatibility in Case of a Flat ECU Extract . . . 547
6.16 Compatibility Examples . . . 548
6.16.1 Compatibility on Assembly Level . . . 548
6.16.1.1 Legal Use . . . 548
6.16.1.2 Illegal Use . . . 549
6.16.2 Compatibility on Delegation Level . . . 549
6.16.2.1 Legal Use . . . 549
6.16.2.2 Illegal Use . . . 552

7 Internal Behavior 554

7.1 Introduction . . . 554
7.2 Runnable Entity . . . 561

7.2.1 Concurrency and Reentrancy of a RunnableEntity that cannot be Invoked Concurrently . . . 567
7.2.2 Concurrency and Reentrancy of a RunnableEntity that can be Invoked Concurrently . . . 568
7.2.3 Timed Activation of Runnable Entities . . . 569
7.2.4 Additional Remarks and Clarifications . . . 570
7.2.4.1 Reentrancy and Multiple Instantiation . . . 570
7.2.4.2 Reentrancy and “Library Functions” . . . 571
7.2.4.3 Compatibility of ClientServerOperations triggering the same RunnableEntity . . . 571
7.2.4.4 Categories of Runnable Entities . . . 572
7.2.4.5 Arguments of a Runnable Entity . . . 574
7.2.5 Activation Reason of a Runnable Entity . . . 575
7.2.6 Runnable Entity for Initialization Purpose . . . 577
7.3 RTEEvent . . . 579
7.3.1 Defining an Event . . . 586
7.3.2 Defining how to Respond to an Event . . . 588
7.4 Communication among Runnable Entities . . . 590
7.4.1 Description Possibility 1: Exclusive Area . . . 591
7.4.1.1 Entire Runnable Runs in the Exclusive Area . . . 593
7.4.1.2 Runnable would Dynamically Enter and Leave the Exclusive Area . . . 594
7.4.1.3 Configuration of API Generation . . . 594
7.4.2 Description Possibility 2: Inter-Runnable Variable . . . 596
7.4.3 Inter Runnable Triggering . . . 598
7.5 Data Access of RunnableEntities . . . 600
7.5.1 RunnableEntities and Sender Receiver Communication . . . 603
7.5.1.1 Terminology . . . 604
7.5.1.2 Data Access . . . 604
7.5.1.3 Explicit Sending and Receiving . . . 607
7.5.1.4 Implicit Sending and Receiving . . . 611
7.5.1.5 DataSendCompletedEvent . . . 612
7.5.1.6 DataWriteCompletedEvent . . . 613
7.5.1.7 DataReceivedEvent . . . 614
7.5.1.8 DataReceiveErrorEvent . . . 614
7.5.2 RunnableEntities and Client Server Communication . . . 616
7.5.2.1 Invoking an Operation . . . 616
7.5.2.2 Providing an Implementation of an Operation . . . 621
7.5.2.3 Reacting on Data Transformation Errors . . . 622
7.5.3 RunnableEntities and External Trigger Event Communication 622
7.5.3.1 Trigger Source . . . 622
7.5.3.2 Trigger Sink . . . 624
7.5.4 RunnableEntities and Parameter Access . . . 624
7.5.4.1 InstantiationDataDefProps . . . 626
7.5.5 RunnableEntities and Mode Communication . . . 628
7.6 Port API Options . . . 628

7.6.1 Enable to Take Address . . . 631
7.6.2 Indirect API Generation . . . 631
7.6.3 Port Defined Argument Value . . . 631
7.6.4 Supported Features . . . 633
7.6.4.1 Buffer Locking . . . 633
7.7 PerInstanceMemory . . . 634
7.7.1 PerInstanceMemory typed by “C” Data Types . . . 635
7.7.2 PerInstanceMemory typed by AUTOSAR Data Types . . . 637
7.8 Static Memory and Constant Memory . . . 637
7.9 Included AUTOSAR Data Types . . . 638
7.10 Included Mode Declaration Groups . . . 640
7.11 Service Needs . . . 641
7.11.1 Overview . . . 641
7.11.2 Assignment of Service Needs to Ports and Data . . . 643
7.12 Variation Point Proxy . . . 651

8 Implementation 658

9 Mode Management 663

9.1 Declaration of Modes . . . 663
9.2 Modes and Events . . . 668
9.3 Initialization / Finalization . . . 672
9.4 Mode Error Behavior . . . 673
9.5 Summary Meta-Model Excerpt Related to Modes . . . 677

10 ECU Abstraction and Complex Drivers 678

10.1 Introduction . . . 678
10.2 High Level Hardware and Software Architecture . . . 678
10.3 Interfaces and APIs . . . 681
10.3.1 ECU Abstraction and its AUTOSAR Interfaces . . . 681
10.4 Sensors/Actuators . . . 682
10.5 I/O Hardware Abstraction . . . 684
10.6 Complex Driver . . . 685

11 Services 690

11.1 Overview: Generation of Service-related Model Elements . . . 690
11.2 Service Software Component Type . . . 693
11.3 Service Proxy Component Type . . . 696
11.4 Non Volatile Memory . . . 699
11.4.1 Introduction . . . 699
11.4.2 NvBlockComponent . . . 699
11.4.3 Software-Components using *NVRAM data* of NvBlockComponents . . . 701
11.4.4 Software-Components connected to NvBlockComponents . . . 704
11.4.5 NvBlockDescriptor . . . 706
11.4.5.1 Writing Strategies . . . 709
11.4.5.2 NvBlockNeeds . . . 715

11.4.5.3 RAM Block and ROM Block 718
11.4.5.4 NvBlockDataMapping 719
11.4.5.5 Client Server Ports 726
11.4.6 BulkNvDataDescriptor 727
11.4.7 SwcInternalBehavior of an NvBlockSwComponentType . . . 729

12 Software Component Documentation 733

13 Service Dependencies and Service Use Cases 737

13.1 Overview 737
13.2 NvM Service Dependencies 737
13.2.1 `Nvm` Use Case: Permanent RAM Block 738
13.2.2 Nvm Use Case: `Temporary RAM Block` 739
13.2.3 Nvm Use Case: RAM Block with explicit synchronization using Mirror Interfaces 740
13.2.4 NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType (not ServiceSwComponent of NvM) 741
13.3 Watchdog Service Dependencies 742
13.3.1 Watchdog Service use Case: Local Supervision 742
13.3.2 Watchdog Service use Case: *Global Supervision Status* notification 744
13.3.3 Watchdog Service use Case: Control global supervision or get global supervision status 745
13.3.4 Watchdog Service use Case: Software-component wants to obtain the status of a local supervision 746
13.3.5 Watchdog Service use Case: Software-component wants to report a checkpoint 746
13.4 COM Manager Service Needs 747
13.4.1 ComM Use Case: read current ComM Mode 747
13.4.2 ComM Use Case: request ComM Mode 748
13.4.3 ComM Use Case: Software-Component acts as a Mode Manager that influences the ECU State 748
13.4.4 Read PNC ComM Mode 749
13.5 ECU State Manager Service Needs 749
13.5.1 EcuM Use Case: read current ECU Mode 750
13.5.2 EcuM Use Case: request a certain ECU state 750
13.5.3 EcuM Use Case: select Shutdown Target 751
13.5.4 EcuM Use Case: select Boot Target 751
13.5.5 EcuM Use Case: use Alarm Clock 751
13.6 BswM 752
13.6.1 Partial Networking 752
13.6.2 Mode Manager 753
13.6.3 Mode User 755
13.6.4 Mode Requester 756
13.6.5 Service Oriented Use Cases 756

13.6.5.1 PortGroup supports Relation between Software-Component and service Instance 757
13.6.5.2 General Server interaction 758
13.6.5.3 General Client interaction 759
13.6.5.4 Event Reception Without Subscription 761
13.6.5.5 Use Case: Server Service Offer 763
13.6.5.6 Use Case: Server Service Subscription Status . . . 763
13.6.5.7 Use Case: Client Event / Method Subscription 765
13.6.5.8 Use Case: Client Event and Method Subscription Status 766
13.7 Crypto Service Dependencies 767
13.7.1 Overview 767
13.7.2 Crypto Service Use Cases 769
13.7.2.1 Crypto Service Use Case: Hash calculation 769
13.7.2.2 Crypto Service Use Case: MAC calculation 770
13.7.2.3 Crypto Service Use Case: MAC verification 771
13.7.2.4 Crypto Service Use Case: generation of random numbers 771
13.7.2.5 Crypto Service Use Case: Encryption with Authenticated Encryption with Associated Data (AEAD) . . . 772
13.7.2.6 Crypto Service Use Case: Decryption with Authenticated Encryption with Associated Data (AEAD) . . . 772
13.7.2.7 Crypto Service Use Case: encryption 773
13.7.2.8 Crypto Service Use Case: decryption 773
13.7.2.9 Crypto Service Use Case: signature generation . . . 774
13.7.2.10 Crypto Service Use Case: signature verification . . . 775
13.7.2.11 Crypto Service Use Case: usage of key management 775
13.7.3 Crypto Service Job Use Cases 776
13.7.3.1 Crypto Service Use Case: usage of job API to set key valid 776
13.7.3.2 Crypto Service Use Case: usage of job API to create a random seed 776
13.7.3.3 Crypto Service Use Case: usage of job API to generate a key 777
13.7.3.4 Crypto Service Use Case: usage of job API to derive a key 777
13.7.3.5 Crypto Service Use Case: usage of job API to execute calculation of the public value for key exchange 778
13.7.3.6 Crypto Service Use Case: usage of job API to execute calculation of shared secret for key exchange . 779
13.7.3.7 Crypto Service Use Case: usage of job API to execute certificate parsing 779
13.7.3.8 Crypto Service Use Case: usage of job API to execute certificate verification 780
13.7.4 Crypto Key Management Use Cases 780

13.7.4.1 KeyM Service Use Case: Software-Component wants check a certificate on KeyM 781
13.7.4.2 KeyM Service Use Case: Software-Component wants to retrieve a certificate from KeyM 781
13.7.4.3 KeyM Service Use Case: Software-Component wants to retrieve elements of a certificate from KeyM 782
13.7.4.4 KeyM Service Use Case: Software-Component wants to check the existence of a certificate from KeyM 782
13.7.4.5 KeyM Service Use Case: Software-Component wants to store a (derived) key in KeyM 783
13.7.4.6 KeyM Service Use Case: Software-Component wants to store a container with (encrypted) keys in KeyM 783
13.7.4.7 KeyM Service Use Case: Software-Component wants to verify if cryptographic operation was executed using a specific key 784
13.8 Diagnostic Service Dependency 784
13.8.1 Development Approach 785
13.8.2 Function Inhibition Needs 785
13.8.2.1 Function Inhibition Manager Service use Case: read function permission 787
13.8.2.2 Function Inhibition Manager Use Case: react on suppressed or unavailable events 787
13.8.3 Diagnostic Event Needs 788
13.8.3.1 Dem Service Use Case: diagnostic monitor, debouncing by Dem 798
13.8.3.2 Dem Service Use Case: diagnostic monitor, debouncing by SWC 798
13.8.3.3 Dem Service Use Case: software-component provides information about operation cycles 799
13.8.3.4 Dem Service Use Case: software-component enables reporting of DTCs in general 799
13.8.3.5 Dem Service Use Case: software-component enables storage of subsequent DTCs 800
13.8.3.6 Dem Service Use Case: retrieve information of the lamp status 800
13.8.3.7 Dem Service Use Case: DEM provides information that the fault storage overflows 802
13.8.3.8 Dem Service Use Case: software-component suppresses the storage of DTCs 802
13.8.3.9 Dem Service Use Case: software-component informs that the PTO is active 803
13.8.3.10 Dem Service Use Case: software-component needs information about any DTC status change 803
13.8.3.11 Dem Service Use Case: call operation if the data of a given diagnostic event changes (I) 804

13.8.3.12 Dem Service Use Case: call operation if the data or status of any diagnostic event changes (II) 804
13.8.3.13 Dem Service Use Case: software-component provides data for diagnostic purposes 805
13.8.3.14 Dem Service Use Case: software-component gets information about a specific DTC 806
13.8.3.15 Dem Service Use Case: Software-Component wants to be triggered on Monitor Status Changes 807
13.8.3.16 Dem Service Use Case: write parameter identifier (PID) by software-component 807
13.8.3.17 Dem Service Use Case: read parameter identifier by software-component 808
13.8.3.18 Dem Service Use Case: diagnostic monitor provides monitor data, debouncing by Dem 808
13.8.3.19 Dem Service Use Case: diagnostic monitor provides monitor data, debouncing by software-component . . 809
13.8.3.20 Dem Service Use Case: software-component checks whether an event is suppressed 810
13.8.3.21 Dem Service Use Case: software-component wants information about DTC clearance 810
13.8.3.22 Dem Service Use Case: software-component wants to reset a degradation state 812
13.8.4 Diagnostic Communication Needs 812
13.8.4.1 Dcm Service Use Case: read/write current values by Client Server Interface 818
13.8.4.2 Dcm Service Use Case: read/write current values of specific DID by Client Server Interface 818
13.8.4.3 Dcm Service Use Case: read/write current values by Sender Receiver Interface or Nv Data Interface . . . 819
13.8.4.4 Dcm Service Use Case: read/write entire DID by Sender Receiver Interface or Nv Data Interface . . . 820
13.8.4.5 Dcm Service Use Case: start/stop or request routine results 821
13.8.4.6 Dcm Service Use Case: IO control by Client Server Interface 822
13.8.4.7 Dcm Service Use Case: IO control by Sender Receiver Interface 822
13.8.4.8 Dcm Service Use Case: Access to protocol, session and security information 825
13.8.4.9 Dcm Service Use Case: Verify the access to security level 826
13.8.4.10 Dcm Service Use Case: multiple testers access one ECU 826
13.8.4.11 Dcm Service Use Case: Service Request Notification 827
13.8.4.12 Dcm Service Use Case: read/write and IOCtrl current values by Client Server Interface 828

13.8.4.13 Dcm Service Use Case: A software-component acts as a "file server" to a diagnostic tester 828
13.8.5 OBD related Needs 829
13.8.5.1 Dem Service Use Case: In-Use-Monitor Performance Ratio calculation 832
13.8.5.2 Dcm Service Use Case: read parameter identifier via diagnostic services by Client Server Interface 833
13.8.5.3 Dcm Service Use Case: read parameter identifier via diagnostic services by Sender Receiver Interface . . 833
13.8.5.4 Dcm Service Use Case: Request vehicle information 834
13.8.5.5 Dem Service Use Case: Read DTR data from SW-C for OBD Service $06 834
13.8.5.6 Dcm Service Use Case: request control of on-board system, test or component 835
13.8.5.7 Dem Service Use Case: In-Use-Monitoring Performance Ratio Denominator interface 836
13.8.6 Diagnostics over IP 837
13.8.6.1 DoIP Service Use Case: GID synchronization can be necessary if the ECU is DoIP Gid synchronization master 840
13.8.6.2 DoIP Service Use Case: Vehicle information is broadcast or can be requested by the tester 841
13.8.6.3 DoIP Service Use Case: Tester could also request the power status with respect to diagnostics 841
13.8.6.4 DoIP Service Use Case: Routing activation mechanism is used which can lead to additional impact regarding authentication or confirmation 842
13.8.6.5 DoIP Service Use Case: a DoIP entity needs to be informed when an external tester is attached or activated. 842
13.8.6.6 Service Use Case: Set and reset Warning Indicator Request bit 843
13.8.6.7 DoIP Service Use Case: Atomic Software-Component provides the further action byte to the DoIP Service Component 844
13.8.7 Miscellaneous Diagnostic Service Use-Cases 845
13.8.7.1 Dcm Service Use Case: DiagnosticSessionControl . 845
13.8.7.2 Dcm Service Use Case: EcuReset 845
13.8.7.3 Dcm Service Use Case: EcuReset ModeRapidPowerShutDown 846
13.8.7.4 Dcm Service Use Case: CommunicationControl . . . 846
13.8.7.5 Dcm Service Use Case: ControlDTCSetting 847
13.8.7.6 Dcm Service Use Case: SecurityAccess 847
13.8.7.7 Service Use Case: Atomic Software-Component implements a Hardware Shutdown 848
13.8.7.8 Service Use Case: Upload and download of data . . 848

13.9 Diagnostic Log and Trace Dependency . . . 849
13.9.1 Dlt use Case:Application software component transmits debug information . . . 850
13.10 Synchronized Time-Base Manager Dependency . . . 851
13.10.1 StbM Use Case: start timer and potentially get notified about its expiration . . . 852
13.10.2 StbM Use Case: Software-Components wants to get notifications of status changes . . . 852
13.10.3 StbM Use Case: Process time snapshot obtained from global time slave for diagnostics purposes . . . 853
13.10.4 StbM Use Case: Software-component represents a global time master . . . 854
13.10.5 StbM Use Case: Software-component represents a global time slave . . . 854
13.10.6 StbM Use Case: Software-component analyzes predictions about the time synchronization process . . . 855
13.10.7 StbM Use Case: Software-Components wants to initiate the cloning of a time base . . . 856
13.11 Secure On-Board Communication . . . 856
13.11.1 SecOc Use Case: obtain the verification status of secure communication . . . 857
13.11.2 SecOc Use Case: software component retires from secure communication for a given period . . . 858
13.11.3 SecOc Use Case: deliver freshness to SecOC I . . . 858
13.11.4 SecOc Use Case: deliver freshness to SecOC II . . . 859
13.11.5 SecOc Use Case: deliver freshness to SecOC III . . . 859
13.11.6 SecOc Use Case: enable the sending of Pdus even if computation of the MAC is not possible . . . 860
13.11.7 SecOc Use Case: Receive notification about an authentication attempt . . . 860
13.12 J1939 Communication . . . 861
13.12.1 J1939RM Use Case: AtomicSwComponentType sends requests to the bus . . . 862
13.12.2 J1939RM Use Case: AtomicSwComponentType accepts requests from the bus . . . 863
13.12.3 J1939Dcm wants to retrieve calibration verification numbers from an application software-component . . . 863
13.13 Error Tracer . . . 864
13.13.1 Error Tracer Use Case: Default Error Tracer Service use Case: report failure . . . 865
13.14 Vehicle-2-X Facilities . . . 866
13.14.1 V2xFac Use Case: Application software component provides Vehicle specific data to the V2X-Stack for CAM transmission 867
13.14.2 V2xFac Use Case: Application software component triggers transmission of DENM message . . . 867
13.15 Vehicle-2-X Management . . . 868

13.15.1 V2xM Use Case: Application software component provides Vehicle specific data to the V2X-Stack for Position and Time information . . . 869
13.15.2 V2xM Use Case: Application software component needs V2X specific data from the V2X Manager . . . 869
13.15.3 V2xM Use Case: Application software component has soft-control over Pseudonym-Change within V2X Manager . . . 870
13.15.4 V2xM Use Case: Application software component has the ability to do Verification-on-Demand . . . 871
13.15.5 V2xM Use Case: Application software component do location based calculations . . . 871
13.16 Vehicle-2-X DataManager . . . 872
13.16.1 V2x DM Use Case: Application software component receives V2X data . . . 873
13.17 Hardware Test Manager . . . 874
13.17.1 HtssM Service Use Case: Query results of hardware tests . 874
13.18 Intrusion Detection System Manager . . . 875
13.18.1 IdsM Service Use Case: AtomicSwComponentType reports security event . . . 876
13.18.2 IdsM Service Use Case: AtomicSwComponentType reports security event using Smart Sensor API . . . 876
13.18.3 IdsM Service Use Case: AtomicSwComponentType provides time stamp to IdsM . . . 877

14 Rapid Prototyping Scenarios 878

14.1 Definition of Rapid Prototyping Scenario . . . 878
14.2 Usage of RptContainers on M1 . . . 881
14.3 Usage of atpSplitable for RptContainers on M1 . . . 883
14.4 Modifications of the Meta-Model for supporting the RPT scenario . . . 883
14.5 Extended Buffer Access Method . . . 886
14.5.1 RP Preparation . . . 887
14.5.2 Service Points . . . 891
14.5.2.1 Service Functions . . . 892

A Glossary 895

B Supported Special Use Cases 899

B.1 Asymmetric Data Transformation between a Software-Component and a Complex Driver . . . 899
B.1.1 Overview . . . 899
B.1.2 Modeling Aspects . . . 900

C History of Constraints and Specification Items 902

C.1 Constraint History of this Document according to AUTOSAR R4.0.1 . . 902
C.1.1 Changed Constraints in R4.0.1 . . . 902
C.1.2 Added Constraints in R4.0.1 . . . 902
C.1.3 Deleted Constraints in R4.0.1 . . . 907

C.2 Constraint History of this Document according to AUTOSAR R4.0.2 . . 907
C.2.1 Changed Constraints in R4.0.2 907
C.2.2 Added Constraints in R4.0.2 908
C.2.3 Deleted Constraints in R4.0.2 909
C.3 Constraint History of this Document according to AUTOSAR R4.0.3 . . 909
C.3.1 Changed Constraints in R4.0.3 909
C.3.2 Added Constraints in R4.0.3 909
C.3.3 Added Specification Items in R4.0.3 912
C.3.4 Deleted Constraints in R4.0.3 927
C.3.5 Deleted Specification Items in R4.0.3 928
C.4 Constraint History of this Document according to AUTOSAR R4.1.1 . . 928
C.4.1 Changed Constraints in R4.1.1 928
C.4.2 Added Constraints in R4.1.1 929
C.4.3 Changed Specification Items in R4.1.1 932
C.4.4 Added Specification Items in R4.1.1 933
C.4.5 Deleted Constraints in R4.1.1 936
C.4.6 Deleted Specification Items in R4.1.1 936
C.5 Constraint History of this Document according to AUTOSAR R4.1.2 . . 937
C.5.1 Changed Constraints in R4.1.2 937
C.5.2 Added Constraints in R4.1.2 937
C.5.3 Changed Specification Items in R4.1.2 938
C.5.4 Added Specification Items in R4.1.2 939
C.5.5 Deleted Constraints in R4.1.2 939
C.5.6 Deleted Specification Items in R4.1.2 940
C.6 Constraint History of this Document according to AUTOSAR R4.1.3 . . 940
C.6.1 Added Traceables in R4.1.3 940
C.6.2 Changed Traceables in R4.1.3 940
C.6.3 Deleted Traceables in R4.1.3 940
C.6.4 Added Constraints in R4.1.3 941
C.6.5 Changed Constraints in R4.1.3 941
C.6.6 Deleted Constraints in R4.1.3 941
C.7 Constraint History of this Document according to AUTOSAR R4.2.1 . . 942
C.7.1 Added Traceables in R4.2.1 942
C.7.2 Changed Traceables in R4.2.1 944
C.7.3 Deleted Traceables in R4.2.1 945
C.7.4 Added Constraints in R4.2.1 945
C.7.5 Changed Constraints in R4.2.1 946
C.7.6 Deleted Constraints in R4.2.1 947
C.8 Constraint History of this Document according to AUTOSAR R4.2.2 . . 948
C.8.1 Added Traceables in R4.2.2 948
C.8.2 Changed Traceables in R4.2.2 949
C.8.3 Deleted Traceables in R4.2.2 949
C.8.4 Added Constraints in R4.2.2 950
C.8.5 Changed Constraints in R4.2.2 951
C.8.6 Deleted Constraints in R4.2.2 952
C.9 Constraint History of this Document according to AUTOSAR R4.3.0 . . 952

C.9.1 Added Traceables in R4.3.0 . . . 952
C.9.2 Changed Traceables in R4.3.0 . . . 955
C.9.3 Deleted Traceables in R4.3.0 . . . 956
C.9.4 Added Constraints in R4.3.0 . . . 957
C.9.5 Changed Constraints in R4.3.0 . . . 959
C.9.6 Deleted Constraints in R4.3.0 . . . 960
C.10 Constraint History of this Document according to AUTOSAR R4.3.1 . . . 960
C.10.1 Added Traceables in R4.3.1 . . . 960
C.10.2 Changed Traceables in R4.3.1 . . . 961
C.10.3 Deleted Traceables in R4.3.1 . . . 961
C.10.4 Added Constraints in R4.3.1 . . . 961
C.10.5 Changed Constraints in R4.3.1 . . . 962
C.10.6 Deleted Constraints in R4.3.1 . . . 963
C.11 Constraint History of this Document according to AUTOSAR R4.4.0 . . . 963
C.11.1 Added Traceables in R4.4.0 . . . 963
C.11.2 Changed Traceables in R4.4.0 . . . 965
C.11.3 Deleted Traceables in R4.4.0 . . . 967
C.11.4 Added Constraints in R4.4.0 . . . 967
C.11.5 Changed Constraints in R4.4.0 . . . 968
C.11.6 Deleted Constraints in R4.4.0 . . . 969
C.12 Constraint History of this Document according to AUTOSAR R19-11 . . . 969
C.12.1 Added Traceables in R19-11 . . . 969
C.12.2 Changed Traceables in R19-11 . . . 970
C.12.3 Deleted Traceables in R19-11 . . . 971
C.12.4 Added Constraints in R19-11 . . . 971
C.12.5 Changed Constraints in R19-11 . . . 972
C.12.6 Deleted Constraints in R19-11 . . . 973
C.13 Constraint History of this Document according to AUTOSAR R20-11 . . . 973
C.13.1 Added Traceables in R20-11 . . . 973
C.13.2 Changed Traceables in R20-11 . . . 974
C.13.3 Deleted Traceables in R20-11 . . . 974
C.13.4 Added Constraints in R20-11 . . . 975
C.13.5 Changed Constraints in R20-11 . . . 979
C.13.6 Deleted Constraints in R20-11 . . . 980
C.14 Constraint History of this Document according to AUTOSAR R21-11 . . . 980
C.14.1 Added Traceables in R21-11 . . . 980
C.14.2 Changed Traceables in R21-11 . . . 981
C.14.3 Deleted Traceables in R21-11 . . . 981
C.14.4 Added Constraints in R21-11 . . . 982
C.14.5 Changed Constraints in R21-11 . . . 982
C.14.6 Deleted Constraints in R21-11 . . . 1000
C.15 Constraint History of this Document according to AUTOSAR R22-11 . . . 1000
C.15.1 Added Traceables in R22-11 . . . 1000
C.15.2 Changed Traceables in R22-11 . . . 1002
C.15.3 Deleted Traceables in R22-11 . . . 1003
C.15.4 Added Constraints in R22-11 . . . 1003

C.15.5 Changed Constraints in R22-11 1004
C.15.6 Deleted Constraints in R22-11 1005

D Modeling of InstanceRef 1007

D.1 Introduction 1007
D.2 Modeling 1008
D.2.1 Components and Compositions 1008
D.2.2 Definition of implicit Communication Behavior 1022
D.2.3 Internal Behavior 1028

E Examples 1034

E.1 Examples for the Definition of variable-size Arrays 1034

F Mentioned Class Tables 1038

G Upstream Mapping 1082

G.1 Introduction 1082
G.2 NvM 1082
G.3 Com 1090
G.4 WdgM 1105
G.5 Dcm 1106
G.6 Dem 1149
G.7 BswM 1156
G.8 MemMap 1161
G.9 RTE 1163
G.10 ECUC 1168
G.11 OS 1174
G.12 SecOC 1175

H Splitable Elements in the Scope of this Document 1176

I Variation Points in the Scope of this Document 1181

References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[2] Specification of RTE Software
AUTOSAR_SWS_RTE

[3] Virtual Functional Bus
AUTOSAR_EXP_VFB

[4] Methodology for Classic Platform
AUTOSAR_TR_Methodology

[5] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

[6] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[7] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions

[8] Requirements on Timing Extensions
AUTOSAR_RS_TimingExtensions

[9] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[10] System Template
AUTOSAR_TPS_SystemTemplate

[11] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[12] Requirements on Software Component Template
AUTOSAR_RS_SoftwareComponentTemplate

[13] Supplementary material of general blueprints for AUTOSAR
AUTOSAR_TR_GeneralBlueprintsSupplement

[14] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[15] Information technology – Universal Coded Character Set (UCS)
http://www.iso.org

[16] Specification of I/O Hardware Abstraction
AUTOSAR_SWS_IOHardwareAbstraction

[17] ISO 17356-4: Road vehicles – Open interface for embedded automotive applications – Part 4: OSEK/VDX Communication (COM)

[18] Specification of Module E2E Transformer
AUTOSAR_SWS_E2ETransformer

[19] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_SWS_E2ELibrary

[20] Specification of Communication Manager
AUTOSAR_SWS_COMManager

[21] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager

[22] Specification of Communication
AUTOSAR_SWS_COM

[23] Diagnostic Extract Template
AUTOSAR_TPS_DiagnosticExtractTemplate

[24] Log And Trace Extract Template
AUTOSAR_TPS_LogAndTraceExtract

[25] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes

[26] ISO/IEC 9899:1999
http://www.iso.org

[27] AUTOSAR XML Schema Production Rules
AUTOSAR_TPS_XMLSchemaProductionRules

[28] ASAM MCD-2 MC (ASAP2 / A2L)
http://www.asam.net
ASAM_AE_MCD-2_MC_BS_V1-7-1.pdf

[29] ASAM MCD 2 Harmonized Data Objects Version 1.1
harmonized-data-objects-V1.1.pdf

[30] Collection of blueprints for AUTOSAR M1 models
AUTOSAR_MOD_GeneralBlueprints

[31] ISO 26262:2018 (all parts) – Road vehicles – Functional Safety
http://www.iso.org

[32] ASAM AE Calibration Data Format V2.0.0
http://www.asam.net
ASAM-AE-CDF-V2_0_0-Users-Guide.pdf

[33] Specification of Operating System
AUTOSAR_SWS_OS

[34] Specification of ECU Configuration Parameters (XML)
AUTOSAR_MOD_ECUConfigurationParameters

[35] Guidelines for the use of the C language in critical systems, ISBN 978-1-906400-

10-1
MISRA_C_2012.pdf

[36] Glossary
AUTOSAR_TR_Glossary

[37] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager

[38] ASAM AE Functional Specification Exchange Format V1.0.0
http://www.asam.net
AE-FSX_V1.0.0.pdf

[39] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager

[40] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager

[41] Specification of Function Inhibition Manager
AUTOSAR_SWS_FunctionInhibitionManager

[42] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager

[43] Specification of Diagnostic Communication Manager
AUTOSAR_SWS_DiagnosticCommunicationManager

[44] Road vehicles – Diagnostic communication over Internet Protocol (DoIP)
http://www.iso.org

[45] Specification of Diagnostic Log and Trace
AUTOSAR_SWS_DiagnosticLogAndTrace

[46] Specification of Synchronized Time-Base Manager
AUTOSAR_SWS_SynchronizedTimeBaseManager

[47] Specification of Secure Onboard Communication
AUTOSAR_SWS_SecureOnboardCommunication

[48] Specification of a Request Manager for SAE J1939
AUTOSAR_SWS_SAEJ1939RequestManager

[49] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer

[50] Specification of Vehicle-2-X Facilities
AUTOSAR_SWS_V2XFacilities

[51] Specification of Vehicle-2-X Management
AUTOSAR_SWS_V2XManagement

[52] Software Process Engineering Meta-Model Specification
http://www.omg.org/spec/SPEM/2.0/

[53] Specification of SOME/IP Transformer
AUTOSAR_SWS_SOMEIPTransformer

1 Introduction

1.1 Overview

This document contains the specification of the AUTOSAR Software-Component Template. Actually, it has been created as a supplement to the formal definition of the Software-Component Template by means of the AUTOSAR meta-model. In other words, this document in addition to the formal specification provides introductory description and rationale for the part of the AUTOSAR meta-model relevant for the definition of software-components.

In this context, the term software-component refers to a formally described piece of software existing that needs the AUTOSAR RTE [2] for execution.

Please note that the general ideas behind the semantics of application software-components have been described in the specification of the Virtual Functional Bus [3]. The latter, however, represents conceptual work that strongly influences but does not totally govern the formal definition of software-components.

Note further that this document does not provide any “best practice” recommendations of software-component modeling nor does it require or enforce a certain methodology. Note however, that the methodology aspect is covered by the specification of the AUTOSAR methodology [4].

Although it is beyond any doubt reasonable to use a suitable AUTOSAR Authoring Tool for dealing with AUTOSAR software-components, this specification does not make any assumptions nor does it give recommendations regarding the tooling.

1.2 Scope

As already mentioned in chapter 1.1, the Scope of this document is the description of AUTOSAR software-components. This work covers the following three aspects:

- A general description of SwComponentTypes using PortPrototypes and PortInterfaces, i.e. this document defines the SwComponentType as an entity which can be described through PortPrototypes which provide or require PortInterfaces.
- A description of CompositionSwComponentTypes which are sub-systems consisting out of connected instances of software-components, i.e. software-components may be defined in the form of hierarchical subsystems which in turn consist of software-components again. The description of such hierarchical structures is in scope of this document.
- A description of AtomicSwComponentType which is implemented as a piece of software that can be mapped to an AUTOSAR ECU.
  An AtomicSwComponentType therefore shows up in the ECU Software Archi-

tecture depicted in Figure 1.1. In this figure, the green (vertically striped) and blue (diagonally striped) borders show the aspects that are described by the Software-Component Template.

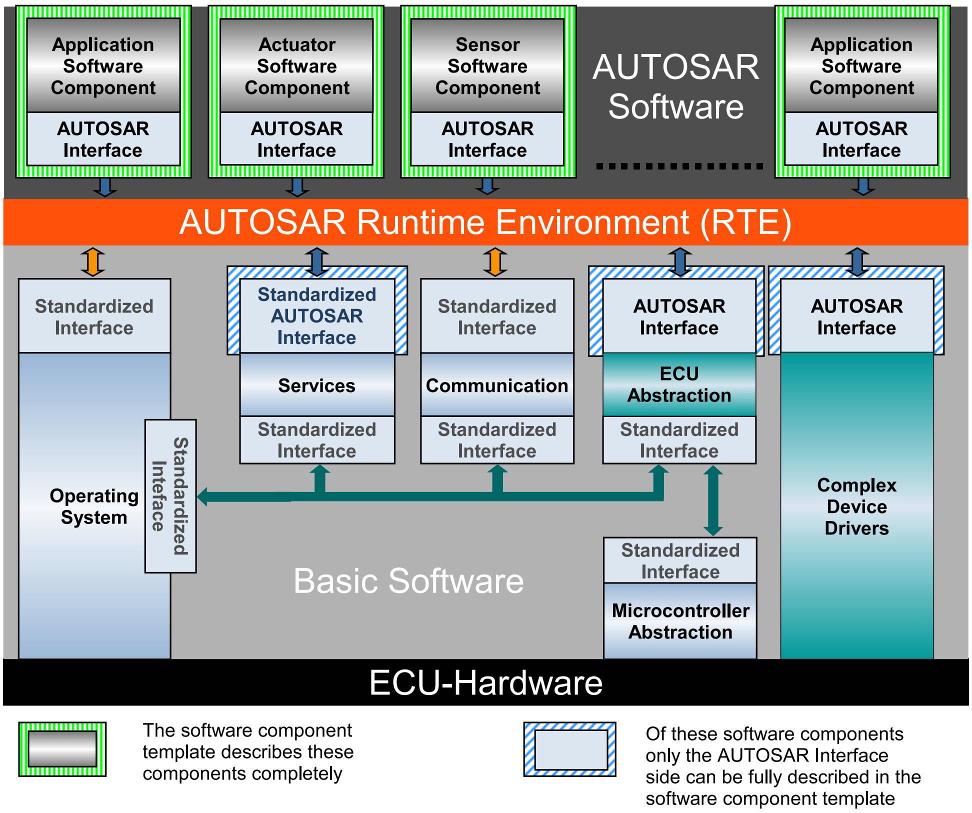


Figure 1.1: Scope of this document in the ECU SW Architecture [5]

Aspects of AUTOSAR Basic Software not relevant for the RTE are out of scope; these are covered by the Basic Software Module Description Template [6].

Also, the document does not cover aspects of timing analysis with respect to the execution of AUTOSAR software-components. This issue is explained in the Specification of Timing Extensions [7] as well as the corresponding requirement specification [8].

1.3 Organization of the Meta-Model

Figure 1.2 sketches the overall structure of the meta-model which formally defines the vocabulary required to describe AUTOSAR software-components. As the diagram points out, other template specifications (e.g. ECU Resource Template [9] and System Template [10]) also use the same modeling approach in order to define an overall consistent model of AUTOSAR software description.

The dashed arrows in the diagram describe dependencies in terms of import-relationships between the packages within the meta-model. For example, the package

SWComponentTemplate imports meta-classes defined in the packages GenericStructure [11] and ECUResourceTemplate [9].

Please note that this specification document will (with some well-defined exceptions) mostly discuss meta-model elements defined in the package SWComponentTemplate.

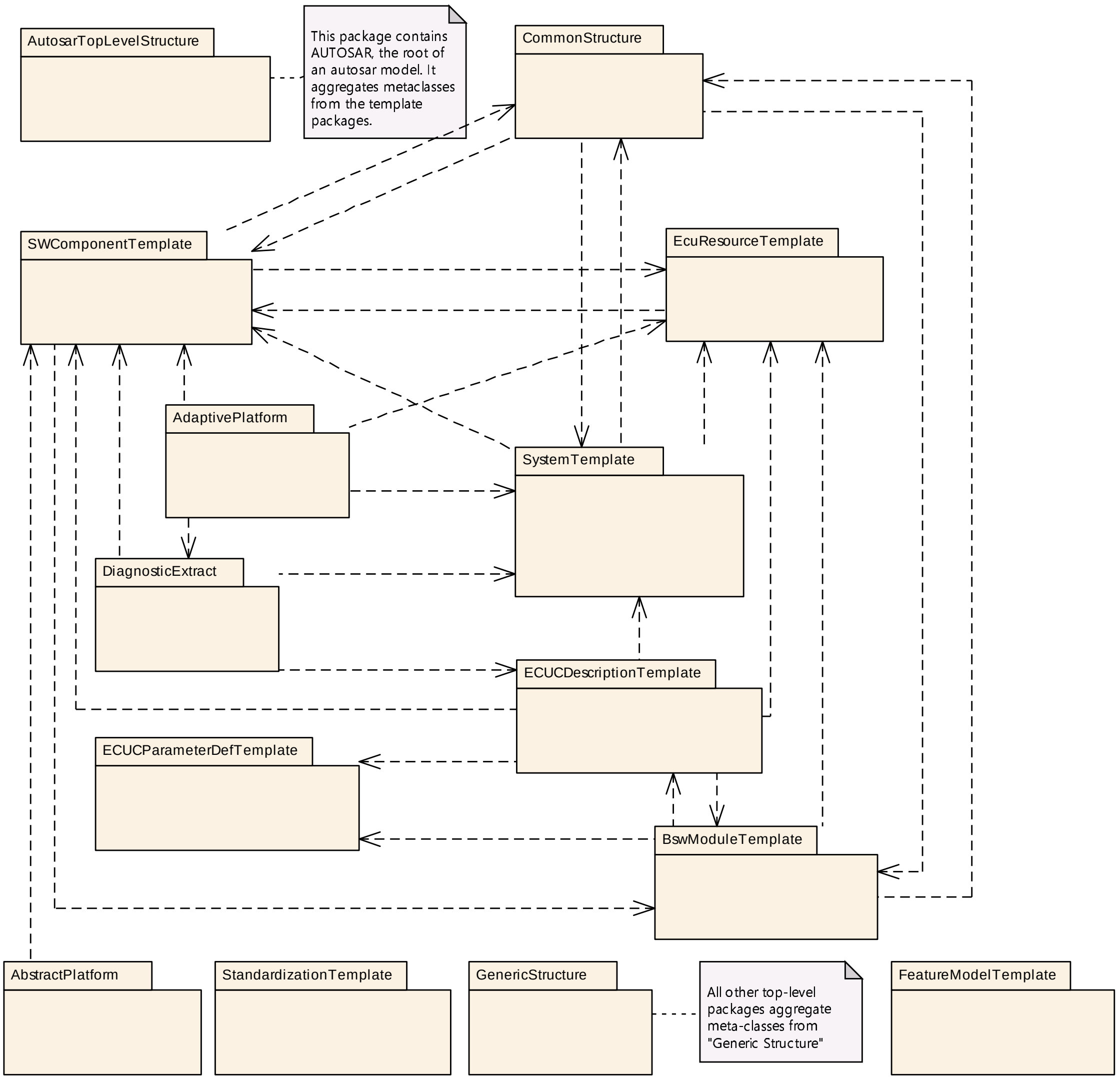


Figure 1.2: Structure of the meta-model

For clarification, please note that the package GenericStructure contains some fundamental infrastructure meta-classes and common patterns that are described in [11]. As these are used by all other template specification the dependency associations are not depicted in the diagram for the sake of clarity.

1.4 Structure of the Template

AUTOSAR software components are described on three distinctive levels, as shown in Figure 1.3.

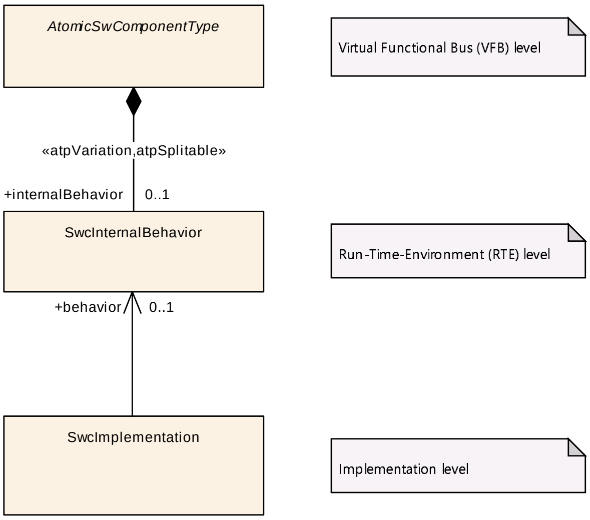


Figure 1.3: The description of a software component is done on three levels

1.4.1 Description of Software Components on VFB Level

The highest (most abstract) description level is the `Virtual Functional Bus` [3]. In this document `SwComponentType`s are described with the means of `DataTypes`, `PortInterfaces`s, `PortPrototype`s, and connections between them. At this level, the fundamental communication properties of components and their communication relationships among each other are expressed.

In the diagram depicted in Figure 1.3, this aspect is expressed by means of the description of `AtomicSwComponentType`[1].

1.4.2 Description of Software Components on RTE Level

The middle level allows for behavior description of a given `AtomicSwComponentType`. This so-called `SwcInternalBehavior` is expressed according to AUTOSAR RTE concepts, e.g. `RTEEvent`s and in terms of schedulable units, so-called `RunnableEntity`s.

For instance, for a `ClientServerOperation` defined in the scope of a particular `ClientServerInterface` on the VFB, the behavior specifies which `RunnableEntity` is activated as a consequence of the invocation of the specific `ClientServerOperation`.

[1]To avoid clutter and require additional up-front information about the meta-model, `CompositionSwComponentType`s have not been added to the diagram.

As sketched by Figure 1.3, there may be zero or one SwcInternalBehaviors aggregated by a given AtomicSwComponentType. In response to the existence of the stereotype ≪atpSplitable≫ at the aggregation it is possible to distribute the aggregation over several physical files.

1.4.3 Descriptions of Software Components on Implementation Level

The lowest level of description specifies the implementation (i.e. in terms of the AUTOSAR meta-model: the SwcImplementation) of a given SwcInternalBehavior description. More precisely, the RunnableEntitys of such a behavior are mapped to code (source code or object code).

There may be different SwcImplementations that reference a specific SwcInternalBehavior description, e.g. in different programming languages, or with differently optimized code.

Please note that Implementation has been described in previous versions of this document. In response to the evolution of the AUTOSAR concept the description of the Implementation aspect has been moved to the “CommonStructure” (see Figure 1.2) because it is also used for creating the Basic Software Module Description Template [6].

However, the SwcImplementation still remains in the scope of this document as it exclusively covers aspects of software-components rather than basic software modules.

1.5 Abbreviations

The following table contains a list of abbreviations used in the scope of this document along with the spelled-out meaning of each of the abbreviations.

Abbreviation	*Meaning*
API	Application Programming Interface
BOM	Byte Order Mark
CAN	Controller Area Network
CSE	Codes for Scaling Units
DCM	Diagnostics Communication Manager
DCY	Driving Cycle
DEM	Diagnostics Event Manager
DID	Diagnostic Identifier
DTC	Diagnostic Trouble Code
DoIp	Diagnostics over IP
ECU	Electrical Control Unit
EPROM	Erasable Programmable Read-Only Memory

▽

△

Abbreviation	Meaning
EEPROM	Electrically Erasable Programmable Read-Only Memory
FID	Function Identifier
GID	Group Identifier
ID	Identifier
IO	Input/Output
IP	Internet Protocol
IUMPR	In-Use Monitor Performance Ratio
ISO	International Standardization Organization
MAC	Message Authentication Code
MCAL	Micro-Controller Abstraction
LIN	Local Interconnect Network
MCD	Measurement, Calibration, Diagnostics
NM	Network Management
NV	Non-Volatile
OBD	On-Board Diagnostic
OEM	Original Equipment Manufacturer
OS	Operating System
PDU	Protocol Data Unit
PID	Parameter Identifier
PTO	Power Take Off
RA	Routing Activation
RAM	Random Access Memory
ROM	Read-Only Memory
RPT	Rapid Prototyping
RTE	Runtime Environment
SWC	Software Component
TID	Test Identifier
UDS	Unified Diagnostic Services
UML	Unified Modeling Language
VFB	Virtual Functional Bus
WWH-OBD	World-Wide Harmonized On-Board Diagnostics
XML	Extensible Markup Language
XSD	XML Schema Definition

Table 1.1: Abbreviations used in the scope of this Document

1.6 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

Please note that constraints are not supposed to be enforceable at any given time in an AUTOSAR workflow. During the development of a model, constraints may legitimately be violated because an incomplete model will obviously show inconsistencies.

However, at specific points in the workflow, constraints shall be enforced as a safeguard against misconfiguration.

The points in the workflow where constraints shall be enforced, sometimes also known as the "binding time" of the constraint, are different for each model category, e.g. on the classic platform, the constraints defined for software-components are typically enforced prior to the generation of the RTE while the constraints against the definition of an Ecu extract shall be applied when the Ecu configuration for the Com stack is created.

For each document, possible binding times of constraints are defined and the binding times are typically mentioned in the constraint themselves to give a proper orientation for implementers of AUTOSAR authoring tools.

Let `AUTOSAR` be an example of a typical class table. The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Type: The type of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attribute is aggregated in the class (`aggr` aggregation), an UML attribute in the class (`attr` primitive attribute), or just referenced by it (`ref` reference). Instance references are also indicated (`iref` instance reference) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

Please note that the chapters that start with a letter instead of a numerical value represent the appendix of the document. The purpose of the appendix is to support the explanation of certain aspects of the document and does not represent binding conventions of the standard.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([1]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([1]).

1.7 Imposition Times of Constraints

The constraints formulated in this document have different *actual* imposition times which denote the steps in the workflow when the respective constraint has to be imposed.

Some imposition times “include” other imposition times, an example for this relation is discussed in the table 1.2.

The imposition times that are considered applicable in the scope of this document[2] are listed in Table 1.2.

Please note that the imposition times are intentionally rendered as technical terms such that it is possible to link back from each constraint to the definition of the affected imposition time in Table 1.2.

This document has been created to apply primarily for the *AUTOSAR classic platform* and therefore the discussed imposition times also apply exclusively to the *AUTOSAR classic platform*.

Some constraints, however, *may* also be meaningful in the context of other imposition times, applicable for other *AUTOSAR platforms*.

[2]Different imposition times may be defined in the context of other AUTOSAR standard documents

Imposition Time	Description
at the time when the contract phase generation is executed	This imposition time is aimed at the time when a software-component is ready for generating the contract phase header files such that the implementation of the software-component can be started.
at the time when the RTE is generated	This imposition time denotes the step in the workflow where the model is considered complete such that the generation of the RTE can be executed. *At the time when the RTE is generated*, all constraints that need to be imposed *at the time when the contract phase generation is executed* **and** those that are imposed *at any time in the workflow* also need to be observed. In other words, a constraint that is imposed *at the time when the contract phase generation is executed* **shall also be imposed** *at the time when the RTE is generated*.
at any time in the workflow	This means that the constraint is invariant of the imposition time and therefore universally applicable. Some model configurations *never* make sense and therefore need to be restricted as early as possible in order to avoid the situation where obviously non-sensical model content is unjustifiably tolerated until some step in the workflow. And then (considerable) effort has to be spent for cleaning up the model.
at the time when the creation of the CompositionSwComponentType is finished	This imposition time applies to the creation of compositions of software-components. This imposition time is considered *optional*. In other words, there may be use cases to deliver CompositionSwComponentTypes that violate constraints with this imposition time to another party. But it may also make sense in some cases to make sure, that a CompositionSwComponentType that is going to be delivered to another party fulfills the constraints associated with this binding time.

Table 1.2: Imposition Times considered in the scope of this document

From the appearance of an imposition time that only applies to the *AUTOSAR classic platform* in the text of a constraint in this document, it **shall not be concluded that the constraint is exclusively applicable to the *AUTOSAR classic platform***.

1.8 Requirements Tracing

Requirements against this document are exclusively stated in the corresponding requirements document [12].

The following table references the requirements specified in [12] and provides information about individual specification items that fulfill a given requirement.

Requirement	Description	Satisfied by
[RS_SWCT_00010]	The Software Component Template shall support inter- and intra-ECU-communication mechanisms with high reliability	[TPS_SWCT_01025] [TPS_SWCT_01026] [TPS_SWCT_01027] [TPS_SWCT_01069] [TPS_SWCT_01070] [TPS_SWCT_01111] [TPS_SWCT_01516] [TPS_SWCT_01573]
[RS_SWCT_00020]	The Software Component Template shall provide open and standardized software interfaces for intra-ECU and inter-ECU communication	[TPS_SWCT_01002]
[RS_SWCT_00030]	The Software Component Template shall provide complete interfaces to application software and basic software modules	[TPS_SWCT_01002] [TPS_SWCT_01003] [TPS_SWCT_01004] [TPS_SWCT_01668] [TPS_SWCT_01672] [TPS_SWCT_01678] [TPS_SWCT_01679] [TPS_SWCT_01694] [TPS_SWCT_01703] [TPS_SWCT_01704] [TPS_SWCT_01705] [TPS_SWCT_01716] [TPS_SWCT_01717] [TPS_SWCT_01718] [TPS_SWCT_01726] [TPS_SWCT_01728] [TPS_SWCT_01730] [TPS_SWCT_01731] [TPS_SWCT_01732] [TPS_SWCT_01733] [TPS_SWCT_01734] [TPS_SWCT_01735] [TPS_SWCT_01740] [TPS_SWCT_01741] [TPS_SWCT_01742] [TPS_SWCT_01743] [TPS_SWCT_01745] [TPS_SWCT_01763] [TPS_SWCT_01776] [TPS_SWCT_01777] [TPS_SWCT_01778] [TPS_SWCT_01779] [TPS_SWCT_01780] [TPS_SWCT_01781] [TPS_SWCT_01782] [TPS_SWCT_01783] [TPS_SWCT_01784] [TPS_SWCT_01810] [TPS_SWCT_01813] [TPS_SWCT_01814] [TPS_SWCT_01815] [TPS_SWCT_01816] [TPS_SWCT_01817] [TPS_SWCT_01818] [TPS_SWCT_01819] [TPS_SWCT_01826] [TPS_SWCT_01827] [TPS_SWCT_01829] [TPS_SWCT_01830] [TPS_SWCT_01832] [TPS_SWCT_01854] [TPS_SWCT_01869] [TPS_SWCT_02019] [TPS_SWCT_02020] [TPS_SWCT_02021] [TPS_SWCT_02022] [TPS_SWCT_02024] [TPS_SWCT_02025] [TPS_SWCT_02026] [TPS_SWCT_02027] [TPS_SWCT_02028] [TPS_SWCT_02031] [TPS_SWCT_02032] [TPS_SWCT_02506]
[RS_SWCT_00070]	The Software Component Template shall provide an abstraction of the application software from hardware	[TPS_SWCT_01030] [TPS_SWCT_01097] [TPS_SWCT_01098]
[RS_SWCT_00080]	The Software Component Template shall provide an independence of application software from in-vehicle communication technologies	[TPS_SWCT_01025] [TPS_SWCT_01026] [TPS_SWCT_01027] [TPS_SWCT_01069] [TPS_SWCT_01070] [TPS_SWCT_01516]
[RS_SWCT_00090]	The Software Component Template should provide an independence of application software from operating systems	[TPS_SWCT_01030] [TPS_SWCT_01097] [TPS_SWCT_01098]
[RS_SWCT_00110]	The Software Component Template shall provide a functional interface view of the entire system	[TPS_SWCT_01025] [TPS_SWCT_01026] [TPS_SWCT_01027] [TPS_SWCT_01069] [TPS_SWCT_01070] [TPS_SWCT_01516]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_00120]	The Software Component Template shall provide protection/unlock mechanisms for software through appropriate services in the infrastructure	[TPS_SWCT_01031] [TPS_SWCT_01049] [TPS_SWCT_01050] [TPS_SWCT_01051] [TPS_SWCT_01052] [TPS_SWCT_01053] [TPS_SWCT_01054] [TPS_SWCT_01055] [TPS_SWCT_01321] [TPS_SWCT_01592] [TPS_SWCT_01713] [TPS_SWCT_01714]
[RS_SWCT_00150]	The Software Component Template shall provide means to protect SW-Components from malicious SW-Components	[TPS_SWCT_01002]
[RS_SWCT_00160]	The Software Component Template shall provide means to achieve compositionality	[TPS_SWCT_01002]
[RS_SWCT_00170]	The Software Component Template shall provide diagnostics means during runtime, for production and services purposes	[TPS_SWCT_01028] [TPS_SWCT_01029] [TPS_SWCT_01129] [TPS_SWCT_01132] [TPS_SWCT_01134] [TPS_SWCT_01135] [TPS_SWCT_01136] [TPS_SWCT_01137] [TPS_SWCT_01138] [TPS_SWCT_01139] [TPS_SWCT_01140] [TPS_SWCT_01425] [TPS_SWCT_01426] [TPS_SWCT_01427] [TPS_SWCT_01453] [TPS_SWCT_01582] [TPS_SWCT_01627] [TPS_SWCT_01628] [TPS_SWCT_01629] [TPS_SWCT_01630] [TPS_SWCT_01631] [TPS_SWCT_01632] [TPS_SWCT_01633] [TPS_SWCT_01634] [TPS_SWCT_01639] [TPS_SWCT_01640] [TPS_SWCT_01654] [TPS_SWCT_01655] [TPS_SWCT_01656] [TPS_SWCT_01657] [TPS_SWCT_01690] [TPS_SWCT_01691] [TPS_SWCT_01706] [TPS_SWCT_01707] [TPS_SWCT_01708] [TPS_SWCT_01709] [TPS_SWCT_01711] [TPS_SWCT_01712] [TPS_SWCT_01715] [TPS_SWCT_01739] [TPS_SWCT_01765] [TPS_SWCT_01766] [TPS_SWCT_01767] [TPS_SWCT_01789] [TPS_SWCT_01790] [TPS_SWCT_01791] [TPS_SWCT_01808] [TPS_SWCT_01853] [TPS_SWCT_01862] [TPS_SWCT_01870] [TPS_SWCT_02002] [TPS_SWCT_02003] [TPS_SWCT_02004] [TPS_SWCT_02005] [TPS_SWCT_02007] [TPS_SWCT_02008] [TPS_SWCT_02009] [TPS_SWCT_02010] [TPS_SWCT_02011] [TPS_SWCT_02012] [TPS_SWCT_02013] [TPS_SWCT_02015] [TPS_SWCT_02016] [TPS_SWCT_02505]
[RS_SWCT_00190]	The Software Component Template shall support hierarchical design methods	[TPS_SWCT_01032] [TPS_SWCT_01033] [TPS_SWCT_01034] [TPS_SWCT_01035] [TPS_SWCT_01036] [TPS_SWCT_01037]
[RS_SWCT_00200]	Definitions of relations between SW components are exhaustive and formal	[TPS_SWCT_01002] [TPS_SWCT_01322] [TPS_SWCT_01323] [TPS_SWCT_01325] [TPS_SWCT_01326] [TPS_SWCT_01328] [TPS_SWCT_01329] [TPS_SWCT_01330] [TPS_SWCT_01331] [TPS_SWCT_01333] [TPS_SWCT_01334] [TPS_SWCT_01335] [TPS_SWCT_01336] [TPS_SWCT_01337] [TPS_SWCT_01338] [TPS_SWCT_01339] [TPS_SWCT_01340] [TPS_SWCT_01341] [TPS_SWCT_01342] [TPS_SWCT_01343] [TPS_SWCT_01345] [TPS_SWCT_01346] [TPS_SWCT_01347] [TPS_SWCT_01348] [TPS_SWCT_01349] [TPS_SWCT_01350] [TPS_SWCT_01351] [TPS_SWCT_01352] [TPS_SWCT_01353] [TPS_SWCT_01557] [TPS_SWCT_01558] [TPS_SWCT_01567] [TPS_SWCT_01663]
[RS_SWCT_00210]	SW components are protected from illegal access	[TPS_SWCT_01002]
[RS_SWCT_00220]	The Software Component Template shall support management of vehicle diversity	[TPS_SWCT_01038] [TPS_SWCT_01040] [TPS_SWCT_01041] [TPS_SWCT_01042] [TPS_SWCT_01447]
[RS_SWCT_00230]	The Software Component Template shall provide the ability to define naming conventions for public symbols	[TPS_SWCT_01000] [TPS_SWCT_01001] [TPS_SWCT_01635]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_02000]	The Software Component Template shall support a top-down hierarchical design	[TPS_SWCT_01032] [TPS_SWCT_01033] [TPS_SWCT_01034] [TPS_SWCT_01035] [TPS_SWCT_01036] [TPS_SWCT_01037]
[RS_SWCT_02010]	Interfaces of atomic software-components shall be supported	[TPS_SWCT_01002]
[RS_SWCT_02020]	Bottom-up design of CompositionTypes shall be supported	[TPS_SWCT_01032] [TPS_SWCT_01033] [TPS_SWCT_01034] [TPS_SWCT_01035] [TPS_SWCT_01036] [TPS_SWCT_01037]
[RS_SWCT_02030]	Specification of Communications shall be supported	[TPS_SWCT_01002] [TPS_SWCT_01025] [TPS_SWCT_01026] [TPS_SWCT_01027] [TPS_SWCT_01087] [TPS_SWCT_01106] [TPS_SWCT_01114] [TPS_SWCT_01115] [TPS_SWCT_01116] [TPS_SWCT_01117] [TPS_SWCT_01118] [TPS_SWCT_01119] [TPS_SWCT_01120] [TPS_SWCT_01121] [TPS_SWCT_01122] [TPS_SWCT_01124] [TPS_SWCT_01196] [TPS_SWCT_01198] [TPS_SWCT_01218] [TPS_SWCT_01454] [TPS_SWCT_01516] [TPS_SWCT_01801]
[RS_SWCT_02060]	Interaction with basic software shall be considered	[TPS_SWCT_01005] [TPS_SWCT_01043] [TPS_SWCT_01044] [TPS_SWCT_01045] [TPS_SWCT_01046] [TPS_SWCT_01556] [TPS_SWCT_01660] [TPS_SWCT_01661] [TPS_SWCT_01689] [TPS_SWCT_01693] [TPS_SWCT_01833]
[RS_SWCT_02080]	Designing a Sensor Actuator Component shall be supported	[TPS_SWCT_01047] [TPS_SWCT_01048]
[RS_SWCT_02090]	Data-consistency for communication among RunnableEntities shall be supported	[TPS_SWCT_01031] [TPS_SWCT_01049] [TPS_SWCT_01050] [TPS_SWCT_01051] [TPS_SWCT_01052] [TPS_SWCT_01053] [TPS_SWCT_01054] [TPS_SWCT_01055] [TPS_SWCT_01637] [TPS_SWCT_01713] [TPS_SWCT_01714]
[RS_SWCT_02100]	Definition of physical units shall be supported	[TPS_SWCT_01056] [TPS_SWCT_01057] [TPS_SWCT_01058] [TPS_SWCT_01059] [TPS_SWCT_01060] [TPS_SWCT_01061] [TPS_SWCT_01068] [TPS_SWCT_01736] [TPS_SWCT_01737]
[RS_SWCT_02110]	Definition of comments shall be supported	[TPS_SWCT_01062] [TPS_SWCT_01203] [TPS_SWCT_01204] [TPS_SWCT_01205] [TPS_SWCT_01206] [TPS_SWCT_01207] [TPS_SWCT_01208] [TPS_SWCT_01209] [TPS_SWCT_01211] [TPS_SWCT_01212] [TPS_SWCT_01214] [TPS_SWCT_01215] [TPS_SWCT_01216] [TPS_SWCT_01217] [TPS_SWCT_01524]
[RS_SWCT_03000]	The SW-Component template shall support compositions	[TPS_SWCT_01032] [TPS_SWCT_01033] [TPS_SWCT_01034] [TPS_SWCT_01035] [TPS_SWCT_01036] [TPS_SWCT_01037]
[RS_SWCT_03010]	The SW-Component template shall support interfaces	[TPS_SWCT_01025] [TPS_SWCT_01026] [TPS_SWCT_01069] [TPS_SWCT_01070] [TPS_SWCT_01516]
[RS_SWCT_03040]	The SW-Component template shall support description of the behavior	[TPS_SWCT_01022] [TPS_SWCT_01030] [TPS_SWCT_01075] [TPS_SWCT_01098] [TPS_SWCT_01108] [TPS_SWCT_01153] [TPS_SWCT_01154] [TPS_SWCT_01155] [TPS_SWCT_01156] [TPS_SWCT_01157] [TPS_SWCT_01302] [TPS_SWCT_01303] [TPS_SWCT_01304] [TPS_SWCT_01305] [TPS_SWCT_01306] [TPS_SWCT_01307] [TPS_SWCT_01309] [TPS_SWCT_01310] [TPS_SWCT_01312] [TPS_SWCT_01313] [TPS_SWCT_01314] [TPS_SWCT_01315] [TPS_SWCT_01317] [TPS_SWCT_01318] [TPS_SWCT_01320] [TPS_SWCT_01324] [TPS_SWCT_01354] [TPS_SWCT_01355] [TPS_SWCT_01356] [TPS_SWCT_01357] [TPS_SWCT_01358] [TPS_SWCT_01359] [TPS_SWCT_01360] [TPS_SWCT_01361] [TPS_SWCT_01363] [TPS_SWCT_01366] [TPS_SWCT_01367] [TPS_SWCT_01368] [TPS_SWCT_01369] [TPS_SWCT_01469] [TPS_SWCT_01483] [TPS_SWCT_01519] [TPS_SWCT_01520] [TPS_SWCT_01521] [TPS_SWCT_01522] [TPS_SWCT_01523] [TPS_SWCT_01687] [TPS_SWCT_01849] [TPS_SWCT_03500] [TPS_SWCT_03501]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_03045]	The SW-Component template shall allow enabling of RTE-Feature to get the activating RTE-Event of Runnable Entity	[TPS_SWCT_01469]
[RS_SWCT_03046]	The SW-Component template shall support instance specific RTE-Events	[TPS_SWCT_02507]
[RS_SWCT_03050]	The SW-Component template shall support the definition of schedulability	[TPS_SWCT_01030] [TPS_SWCT_01097] [TPS_SWCT_01098]
[RS_SWCT_03055]	The SW-Component template shall support optional configuration of ExclusiveArea usage within RunnableEntities	[TPS_SWCT_01457] [TPS_SWCT_01458] [TPS_SWCT_01459] [TPS_SWCT_01460]
[RS_SWCT_03065]	The Software Component Template shall support the definition of implicit communication behavior	[TPS_SWCT_01466] [TPS_SWCT_01470] [TPS_SWCT_01471] [TPS_SWCT_01472] [TPS_SWCT_01473] [TPS_SWCT_01474] [TPS_SWCT_01475] [TPS_SWCT_01476] [TPS_SWCT_01479] [TPS_SWCT_01480] [TPS_SWCT_01481] [TPS_SWCT_01482] [TPS_SWCT_01509] [TPS_SWCT_01625]
[RS_SWCT_03090]	The SW-Component template shall support the definition of needed and usable sensors and actuators	[TPS_SWCT_01047] [TPS_SWCT_01048]
[RS_SWCT_03100]	The SW-Component template shall support variant handling	[TPS_SWCT_01040] [TPS_SWCT_01041] [TPS_SWCT_01042] [TPS_SWCT_01370] [TPS_SWCT_01371] [TPS_SWCT_01372] [TPS_SWCT_01373] [TPS_SWCT_01448] [TPS_SWCT_01858]
[RS_SWCT_03110]	The SW-Component template shall support modes	[TPS_SWCT_01071] [TPS_SWCT_01153] [TPS_SWCT_01154] [TPS_SWCT_01190] [TPS_SWCT_01376] [TPS_SWCT_01377] [TPS_SWCT_01378] [TPS_SWCT_01379] [TPS_SWCT_01380] [TPS_SWCT_01381] [TPS_SWCT_01382] [TPS_SWCT_01383] [TPS_SWCT_01384] [TPS_SWCT_01385] [TPS_SWCT_01388] [TPS_SWCT_01530] [TPS_SWCT_01531] [TPS_SWCT_01532] [TPS_SWCT_01533] [TPS_SWCT_01534] [TPS_SWCT_01535] [TPS_SWCT_01536] [TPS_SWCT_01541] [TPS_SWCT_01542] [TPS_SWCT_01552] [TPS_SWCT_01553] [TPS_SWCT_01554] [TPS_SWCT_01555] [TPS_SWCT_01581] [TPS_SWCT_01664] [TPS_SWCT_03502] [TPS_SWCT_03503] [TPS_SWCT_03504] [TPS_SWCT_03505]
[RS_SWCT_03115]	The SW-Component template shall support mapping of mode declarations	[TPS_SWCT_01464] [TPS_SWCT_01465] [TPS_SWCT_01545]
[RS_SWCT_03120]	The SW-Component template shall support dependency on modes	[TPS_SWCT_01077]
[RS_SWCT_03130]	The SW-Component template shall support connections between PortInterfaces	[TPS_SWCT_01079] [TPS_SWCT_01080] [TPS_SWCT_01081] [TPS_SWCT_01082] [TPS_SWCT_01083] [TPS_SWCT_01084] [TPS_SWCT_01113] [TPS_SWCT_01507] [TPS_SWCT_01515] [TPS_SWCT_01573] [TPS_SWCT_01843]
[RS_SWCT_03135]	The SW-Component template shall support record type subsetting	[TPS_SWCT_01023] [TPS_SWCT_01024] [TPS_SWCT_01551]
[RS_SWCT_03136]	The SW-Component template shall support record type subsetting with primitive types	[TPS_SWCT_01195]
[RS_SWCT_03140]	The SW-Component template shall support conditional existence of PortPrototypes	[TPS_SWCT_01038]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_03141]	The SW-Component template shall support the conditional existence of data element prototypes, operation prototypes, parameter prototypes in an interface	[TPS_SWCT_01106]
[RS_SWCT_03142]	The SW-Component template shall support the conditional existence of Component Prototypes	[TPS_SWCT_01038]
[RS_SWCT_03143]	The SW-Component template shall support the conditional existence of Connector Prototypes	[TPS_SWCT_01040]
[RS_SWCT_03144]	The SW-Component template shall support a configurable size of arrays	[TPS_SWCT_01076] [TPS_SWCT_01078]
[RS_SWCT_03148]	Attributes swMinAxisPoints and swMaxAxisPoints shall be adjustable by an System Constant Definition	[TPS_SWCT_01107] [TPS_SWCT_01181] [TPS_SWCT_01839]
[RS_SWCT_03149]	The SW-Component template shall support the conditional existence of RunnableEntitys	[TPS_SWCT_01085]
[RS_SWCT_03150]	The SW-Component template shall support the conditional existence of RTEEvents	[TPS_SWCT_01085]
[RS_SWCT_03151]	The SW-Component template shall support the conditional existence of InterRunnable Variables	[TPS_SWCT_01085]
[RS_SWCT_03152]	The SW-Component template shall support the conditional accessibility for measurement	[TPS_SWCT_01130]
[RS_SWCT_03153]	The SW-Component template shall support the conditional existence of parameter prototypes	[TPS_SWCT_01085]
[RS_SWCT_03154]	The SW-Component template shall support conditional ports for software components	[TPS_SWCT_01038]
[RS_SWCT_03155]	The SW-Component template shall support interfaces with different resolutions	[TPS_SWCT_01099] [TPS_SWCT_01100] [TPS_SWCT_01101] [TPS_SWCT_01102] [TPS_SWCT_01103] [TPS_SWCT_01104] [TPS_SWCT_01105]
[RS_SWCT_03170]	The SW-Component template shall support fixed data exchange	[TPS_SWCT_01102] [TPS_SWCT_01103] [TPS_SWCT_01104]
[RS_SWCT_03175]	The SW-Component template shall support the definition of calibration datasets	[TPS_SWCT_01177] [TPS_SWCT_01178] [TPS_SWCT_01188] [TPS_SWCT_01793] [TPS_SWCT_01794]
[RS_SWCT_03180]	The SW-Component template shall support SAE J1939 Protocol Features	[TPS_SWCT_01076] [TPS_SWCT_01673] [TPS_SWCT_01674] [TPS_SWCT_01809]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_03181]	The SW-Component template shall support arrays of variable number of elements within the maximum size	[TPS_SWCT_01076] [TPS_SWCT_01127] [TPS_SWCT_01495] [TPS_SWCT_01601] [TPS_SWCT_01602] [TPS_SWCT_01604] [TPS_SWCT_01605] [TPS_SWCT_01606] [TPS_SWCT_01607] [TPS_SWCT_01608] [TPS_SWCT_01610] [TPS_SWCT_01612] [TPS_SWCT_01613] [TPS_SWCT_01614] [TPS_SWCT_01615] [TPS_SWCT_01617] [TPS_SWCT_01618] [TPS_SWCT_01619] [TPS_SWCT_01620] [TPS_SWCT_01621] [TPS_SWCT_01622] [TPS_SWCT_01623] [TPS_SWCT_01636] [TPS_SWCT_01641] [TPS_SWCT_01642] [TPS_SWCT_01644] [TPS_SWCT_01645] [TPS_SWCT_01647] [TPS_SWCT_01648] [TPS_SWCT_01649] [TPS_SWCT_01650] [TPS_SWCT_01793] [TPS_SWCT_01794]
[RS_SWCT_03182]	The SW-Component template shall support byte arrays of variable number of elements	[TPS_SWCT_01127]
[RS_SWCT_03190]	The SW-Component template shall support the ability to publish/specify the diagnostic capabilities and its resources of an SWC	[TPS_SWCT_01028] [TPS_SWCT_01029] [TPS_SWCT_01129] [TPS_SWCT_01132] [TPS_SWCT_01134] [TPS_SWCT_01135] [TPS_SWCT_01136] [TPS_SWCT_01137] [TPS_SWCT_01138] [TPS_SWCT_01139] [TPS_SWCT_01140] [TPS_SWCT_01421] [TPS_SWCT_01422] [TPS_SWCT_01425] [TPS_SWCT_01426] [TPS_SWCT_01427] [TPS_SWCT_01453] [TPS_SWCT_01537] [TPS_SWCT_01538] [TPS_SWCT_01539] [TPS_SWCT_01540] [TPS_SWCT_01544] [TPS_SWCT_01546] [TPS_SWCT_01547] [TPS_SWCT_01577] [TPS_SWCT_01578] [TPS_SWCT_01582] [TPS_SWCT_01627] [TPS_SWCT_01628] [TPS_SWCT_01629] [TPS_SWCT_01630] [TPS_SWCT_01631] [TPS_SWCT_01632] [TPS_SWCT_01633] [TPS_SWCT_01634] [TPS_SWCT_01639] [TPS_SWCT_01640] [TPS_SWCT_01654] [TPS_SWCT_01655] [TPS_SWCT_01656] [TPS_SWCT_01657] [TPS_SWCT_01680] [TPS_SWCT_01690] [TPS_SWCT_01691] [TPS_SWCT_01706] [TPS_SWCT_01707] [TPS_SWCT_01708] [TPS_SWCT_01709] [TPS_SWCT_01711] [TPS_SWCT_01712] [TPS_SWCT_01715] [TPS_SWCT_01739] [TPS_SWCT_01746] [TPS_SWCT_01765] [TPS_SWCT_01766] [TPS_SWCT_01767] [TPS_SWCT_01769] [TPS_SWCT_01789] [TPS_SWCT_01790] [TPS_SWCT_01791] [TPS_SWCT_01808] [TPS_SWCT_01853] [TPS_SWCT_01862] [TPS_SWCT_01870] [TPS_SWCT_02002] [TPS_SWCT_02003] [TPS_SWCT_02004] [TPS_SWCT_02005] [TPS_SWCT_02007] [TPS_SWCT_02008] [TPS_SWCT_02009] [TPS_SWCT_02010] [TPS_SWCT_02011] [TPS_SWCT_02012] [TPS_SWCT_02013] [TPS_SWCT_02015] [TPS_SWCT_02016] [TPS_SWCT_02505]
[RS_SWCT_03200]	The SW-Component template shall support vehicle and application mode management	[TPS_SWCT_01008] [TPS_SWCT_01009] [TPS_SWCT_01010] [TPS_SWCT_01011] [TPS_SWCT_01016] [TPS_SWCT_01017] [TPS_SWCT_01018] [TPS_SWCT_01019] [TPS_SWCT_01020] [TPS_SWCT_01021] [TPS_SWCT_01063] [TPS_SWCT_01064] [TPS_SWCT_01065] [TPS_SWCT_01066] [TPS_SWCT_01067] [TPS_SWCT_01071] [TPS_SWCT_01126] [TPS_SWCT_01450] [TPS_SWCT_01451] [TPS_SWCT_01552] [TPS_SWCT_01553] [TPS_SWCT_01554] [TPS_SWCT_01572] [TPS_SWCT_01581] [TPS_SWCT_01664] [TPS_SWCT_01811] [TPS_SWCT_03502] [TPS_SWCT_03503] [TPS_SWCT_03504] [TPS_SWCT_03505]
[RS_SWCT_03201]	The SW-Component template shall support Portgroups	[TPS_SWCT_01063] [TPS_SWCT_01064] [TPS_SWCT_01065] [TPS_SWCT_01066] [TPS_SWCT_01096] [TPS_SWCT_01126] [TPS_SWCT_01169] [TPS_SWCT_01173] [TPS_SWCT_01174]
[RS_SWCT_03202]	The SW-Component template shall support enabling SWCs to request dedicated modes	[TPS_SWCT_01086] [TPS_SWCT_01201] [TPS_SWCT_01353] [TPS_SWCT_01554] [TPS_SWCT_01572] [TPS_SWCT_03502] [TPS_SWCT_03504]
[RS_SWCT_03203]	The SW-Component template shall support propagation of mode information	[TPS_SWCT_01086] [TPS_SWCT_01087] [TPS_SWCT_01200] [TPS_SWCT_01201] [TPS_SWCT_01202] [TPS_SWCT_01552] [TPS_SWCT_01553] [TPS_SWCT_01566] [TPS_SWCT_01664] [TPS_SWCT_03503] [TPS_SWCT_03505]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_03210]	The SW-Component template shall support integrity and scaling at ports	[TPS_SWCT_01023] [TPS_SWCT_01024] [TPS_SWCT_01099] [TPS_SWCT_01100] [TPS_SWCT_01101] [TPS_SWCT_01102] [TPS_SWCT_01103] [TPS_SWCT_01104] [TPS_SWCT_01105] [TPS_SWCT_01158] [TPS_SWCT_01159] [TPS_SWCT_01160] [TPS_SWCT_01161] [TPS_SWCT_01162] [TPS_SWCT_01163] [TPS_SWCT_01164] [TPS_SWCT_01165] [TPS_SWCT_01166] [TPS_SWCT_01167] [TPS_SWCT_01168] [TPS_SWCT_01449] [TPS_SWCT_01543] [TPS_SWCT_01549] [TPS_SWCT_01550] [TPS_SWCT_01551] [TPS_SWCT_01560] [TPS_SWCT_01561] [TPS_SWCT_01583] [TPS_SWCT_01768]
[RS_SWCT_03215]	The SW-Component template shall define the need to add application data type on top of implementation data type	[TPS_SWCT_01072] [TPS_SWCT_01073] [TPS_SWCT_01074] [TPS_SWCT_01076] [TPS_SWCT_01078] [TPS_SWCT_01189] [TPS_SWCT_01229] [TPS_SWCT_01231] [TPS_SWCT_01235] [TPS_SWCT_01236] [TPS_SWCT_01247] [TPS_SWCT_01256] [TPS_SWCT_01295] [TPS_SWCT_01296] [TPS_SWCT_01298] [TPS_SWCT_01299] [TPS_SWCT_01300] [TPS_SWCT_01489] [TPS_SWCT_01837]
[RS_SWCT_03216]	The SW-Component template shall support application data type	[TPS_SWCT_01072] [TPS_SWCT_01073] [TPS_SWCT_01179] [TPS_SWCT_01180] [TPS_SWCT_01181] [TPS_SWCT_01183] [TPS_SWCT_01184] [TPS_SWCT_01185] [TPS_SWCT_01189] [TPS_SWCT_01191] [TPS_SWCT_01229] [TPS_SWCT_01230] [TPS_SWCT_01231] [TPS_SWCT_01235] [TPS_SWCT_01236] [TPS_SWCT_01237] [TPS_SWCT_01240] [TPS_SWCT_01241] [TPS_SWCT_01242] [TPS_SWCT_01243] [TPS_SWCT_01244] [TPS_SWCT_01245] [TPS_SWCT_01247] [TPS_SWCT_01249] [TPS_SWCT_01256] [TPS_SWCT_01486] [TPS_SWCT_01562] [TPS_SWCT_01564] [TPS_SWCT_01565] [TPS_SWCT_01760] [TPS_SWCT_01834] [TPS_SWCT_01835] [TPS_SWCT_01839]
[RS_SWCT_03217]	The SW-Component template shall support implementation data type	[TPS_SWCT_01006] [TPS_SWCT_01007] [TPS_SWCT_01072] [TPS_SWCT_01074] [TPS_SWCT_01183] [TPS_SWCT_01184] [TPS_SWCT_01189] [TPS_SWCT_01191] [TPS_SWCT_01229] [TPS_SWCT_01231] [TPS_SWCT_01232] [TPS_SWCT_01233] [TPS_SWCT_01235] [TPS_SWCT_01236] [TPS_SWCT_01237] [TPS_SWCT_01248] [TPS_SWCT_01250] [TPS_SWCT_01251] [TPS_SWCT_01252] [TPS_SWCT_01253] [TPS_SWCT_01254] [TPS_SWCT_01255] [TPS_SWCT_01257] [TPS_SWCT_01258] [TPS_SWCT_01259] [TPS_SWCT_01442] [TPS_SWCT_01443] [TPS_SWCT_01478] [TPS_SWCT_01564] [TPS_SWCT_01565] [TPS_SWCT_01610] [TPS_SWCT_01612] [TPS_SWCT_01613] [TPS_SWCT_01614] [TPS_SWCT_01615] [TPS_SWCT_01617] [TPS_SWCT_01618] [TPS_SWCT_01619] [TPS_SWCT_01620] [TPS_SWCT_01621] [TPS_SWCT_01622] [TPS_SWCT_01647] [TPS_SWCT_01648] [TPS_SWCT_01649] [TPS_SWCT_01650] [TPS_SWCT_01700] [TPS_SWCT_01701] [TPS_SWCT_01702] [TPS_SWCT_01759] [TPS_SWCT_01772] [TPS_SWCT_01773]
[RS_SWCT_03218]	The SW-Component template shall support data types for primitive data mapping	[TPS_SWCT_01477]
[RS_SWCT_03220]	The SW-Component template shall allow communication attributes on compositions	[TPS_SWCT_01088] [TPS_SWCT_01568]
[RS_SWCT_03221]	The SW-Component template shall allow port specific configuration of data transformation properties	[TPS_SWCT_01222] [TPS_SWCT_01594] [TPS_SWCT_01595] [TPS_SWCT_01596] [TPS_SWCT_01597] [TPS_SWCT_01598] [TPS_SWCT_01599] [TPS_SWCT_01600] [TPS_SWCT_01626] [TPS_SWCT_01812] [TPS_SWCT_03500] [TPS_SWCT_03501]
[RS_SWCT_03222]	The SW-Component template shall support error notification for transformed data communication	[TPS_SWCT_01616] [TPS_SWCT_01624] [TPS_SWCT_01626]

▽

△

Requirement	Description	Satisfied by
[RS_SWCT_03225]	The SW-Component template shall support an enhanced non-volatile (NV) memory interface	[TPS_SWCT_01141] [TPS_SWCT_01142] [TPS_SWCT_01143] [TPS_SWCT_01227] [TPS_SWCT_01228] [TPS_SWCT_01584] [TPS_SWCT_01585] [TPS_SWCT_01586] [TPS_SWCT_01587] [TPS_SWCT_01588] [TPS_SWCT_01589] [TPS_SWCT_01590] [TPS_SWCT_01662] [TPS_SWCT_01665] [TPS_SWCT_01666] [TPS_SWCT_01675] [TPS_SWCT_01754] [TPS_SWCT_01755] [TPS_SWCT_01795] [TPS_SWCT_01805] [TPS_SWCT_01806] [TPS_SWCT_01807] [TPS_SWCT_02501] [TPS_SWCT_02502] [TPS_SWCT_02503] [TPS_SWCT_02504]
[RS_SWCT_03230]	The SW-Component template shall support documentation of M1 artifacts	[TPS_SWCT_01062] [TPS_SWCT_01699]
[RS_SWCT_03240]	The SW-Component template shall support end-to-end communication protection	[TPS_SWCT_01089] [TPS_SWCT_01090] [TPS_SWCT_01091] [TPS_SWCT_01092] [TPS_SWCT_01093] [TPS_SWCT_01094] [TPS_SWCT_01095] [TPS_SWCT_01508] [TPS_SWCT_01529] [TPS_SWCT_01850] [TPS_SWCT_01851] [TPS_SWCT_01852]
[RS_SWCT_03241]	The SW-Component template shall support partial networking	[TPS_SWCT_01169] [TPS_SWCT_01170] [TPS_SWCT_01171] [TPS_SWCT_01172] [TPS_SWCT_01173] [TPS_SWCT_01174] [TPS_SWCT_01175]
[RS_SWCT_03250]	The SW-Component template shall support bidirectional communication	[TPS_SWCT_01112] [TPS_SWCT_01113] [TPS_SWCT_01454] [TPS_SWCT_01455] [TPS_SWCT_01514] [TPS_SWCT_01573]
[RS_SWCT_03260]	The SW-Component template shall support rule-based initialization of arrays	[TPS_SWCT_01484] [TPS_SWCT_01485] [TPS_SWCT_01493] [TPS_SWCT_01494] [TPS_SWCT_01495] [TPS_SWCT_01528] [TPS_SWCT_01609] [TPS_SWCT_01692]
[RS_SWCT_03270]	The SW-Component template shall support overriding the activation period time on instance level	[TPS_SWCT_02507]
[RS_SWCT_03280]	The SW-Component template shall support the description of bypass points and bypass scenarios	[TPS_SWCT_01719] [TPS_SWCT_01720] [TPS_SWCT_01721] [TPS_SWCT_01722] [TPS_SWCT_01723] [TPS_SWCT_01724] [TPS_SWCT_02046] [TPS_SWCT_02047] [TPS_SWCT_02048] [TPS_SWCT_02049] [TPS_SWCT_02050] [TPS_SWCT_02051] [TPS_SWCT_02052]
[RS_SWCT_03281]	The SW-Component template shall support post-build hooking tools for rapid prototyping	[TPS_SWCT_02047]
[RS_SWCT_03282]	The SW-Component template shall support the description of service points and rapid prototyping scenarios	[TPS_SWCT_02046] [TPS_SWCT_02047]
[RS_SWCT_03290]	The SW-Component template shall support the initialization of runnables without usage of mode management	[TPS_SWCT_01525]
[RS_SWCT_03310]	The SW-Component template shall support Diagnostics over IP	[TPS_SWCT_01537] [TPS_SWCT_01538] [TPS_SWCT_01539] [TPS_SWCT_01544] [TPS_SWCT_01546] [TPS_SWCT_01547] [TPS_SWCT_01746]
[RS_SWCT_03320]	The SW-Component template shall support the definition of optional elements for communication	[TPS_SWCT_01771] [TPS_SWCT_01772] [TPS_SWCT_01773] [TPS_SWCT_01774] [TPS_SWCT_01775] [TPS_SWCT_01785] [TPS_SWCT_01786] [TPS_SWCT_01821] [TPS_SWCT_01822] [TPS_SWCT_01823]

Table 1.3: RequirementsTracing

2 Conceptual Aspects

2.1 Introduction

For the sake of a compact description of relevant meta-model elements the discussion and explanation of conceptual aspects has been concentrated in this chapter.

Reading this chapter is not a pre-requisite for understanding the subsequent chapters. It just provides a central place for the detailed description of conceptual aspects used in various other chapters of this document.

The actual explanation of the concept of a software-component starts in chapter 3.

2.2 Measurement and Calibration

2.2.1 Basic Approach of Measurement and Calibration

While performing the calibration process using an MCD tool (Measurement, Calibration, and Diagnostic) the calibration engineer needs to have a specific insight to the data within the CPU at runtime.

This insight is provided by access to ECU internal variables (also called measurements) as well as calibration parameters (sometimes also called characteristic value). For more details, please refer to [TPS_SWCT_01418]

The description of measurement variables and calibration parameters is basically the same. In AUTOSAR both appear finally as `DataPrototype`s.

2.2.2 Calibration Parameters Overview

A Calibration Parameter is a parameter which characterizes the dynamics of a control algorithm. From a software implementation point of view, it is a variable with only read-access during the normal operation of an ECU. Characteristics are specialized `DataPrototype` entities in terms of its associated type but are used similarly.

[TPS_SWCT_01418] Ways to define a calibration parameter ⌈This means that Calibration Parameters can be defined

- individually for a `SwComponentPrototype` in the `SwcInternalBehavior` of a `SwComponentType` via an aggregation of an `ParameterDataPrototype` in the role of `perInstanceParameter` (similar to `PerInstanceMemory`).
- sharing between all `SwComponentPrototype`s of the same `SwComponentType` in its `SwcInternalBehavior` via an aggregation of an `ParameterDataPrototype` in the role of `sharedParameter` or `constantMemory`.

- for several SwComponentPrototypes (using the port-/interface-concept with ParameterInterfaces).

⌋()

Please note:

- The definition of perInstanceParameter is further described in chapter 2.2.3.3.
- Chapter 2.2.3.2 provides more information about the definition of sharedParameter or constantMemory.
- For more information regarding the definition of ParameterInterface, please refer to chapter 2.2.3.1.

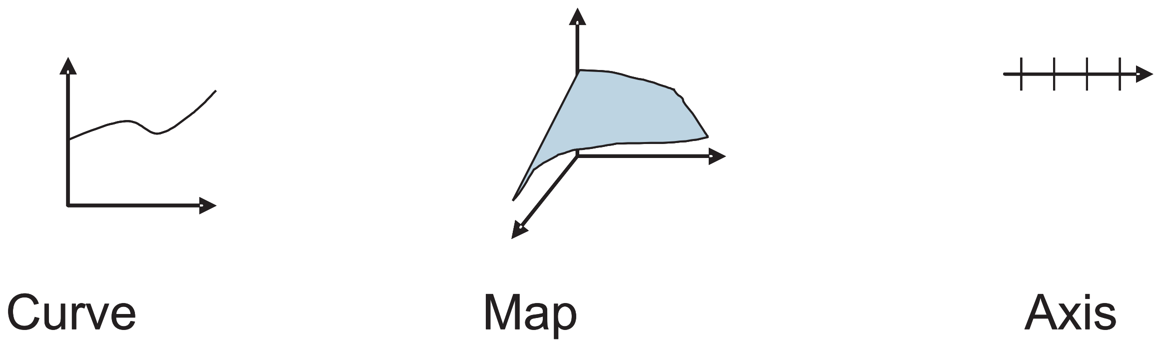


Figure 2.1: Some Categories of calibration parameters

Note: the structure of various calibration objects is visualized in [13].

2.2.3 Using Calibration Parameters

As mentioned above, a ParameterDataPrototype can be used in the context of SwcInternalBehavior as well as in the context of PortPrototypes.

2.2.3.1 Sharing Calibration Parameters within Compositions

To provide calibration parameters for being visible in other SwComponentTypes, a dedicated ParameterSwComponentType (see Figure 3.4) that inherits from SwComponentType has to be used as a SwComponentPrototype within a CompositionSwComponentType.

[TPS_SWCT_01420] SwComponentType requiring access to shared calibration parameters needs RPortPrototype typed by a ParameterInterface ⌈Every SwComponentType requiring access to shared calibration parameters will have an RPortPrototype typed by a ParameterInterface. The definition of this shared calibration access in the context of a CompositionSwComponentType will be defined by creating a SwConnector between the relevant SwComponentPrototypes.⌋()

Class	**ParameterSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ParameterSwComponentType defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected SwComponentPrototypes **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
constantMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular ParameterSwComponentType **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantMapping
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular ParameterSwComponentType **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
instantiationDataDefProps	InstantiationDataDefProps	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationDataDefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 2.1: ParameterSwComponentType

Class	**ParameterInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A parameter interface declares a number of parameter and characteristic values to be exchanged between parameter components and software components. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
parameter	ParameterDataPrototype	*	aggr	The ParameterDataPrototype of this ParameterInterface.

Table 2.2: ParameterInterface

[TPS_SWCT_01421] `ParameterInterface` is not restricted to parameters which can actually can be calibrated ⌈Note that a `ParameterInterface` is not restricted to parameters which can actually can be calibrated. It can be used whenever there shall be no write access to the data during normal operation of the software, i.e. only constant data are visible over the interface.⌋*(RS_SWCT_03190)*

The compatibility rules for `ParameterInterface`s are described in chapter 6.4; the compatibility rules for `ParameterDataPrototype`s are described in chapter 6.4.4.

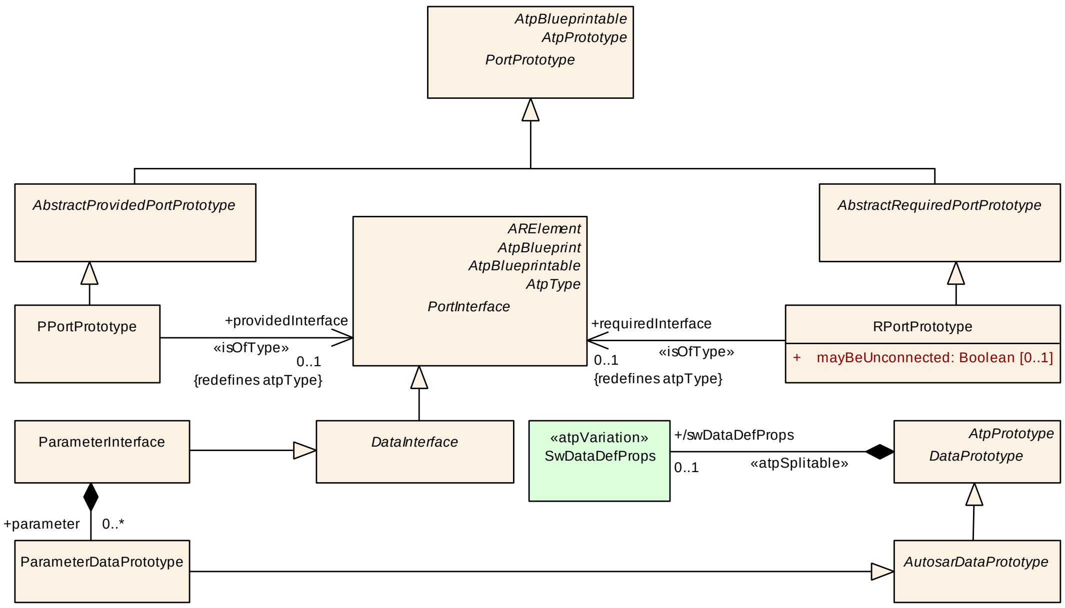


Figure 2.2: ParameterInterface

[TPS_SWCT_01422] Delegation of PortPrototypes typed by a ParameterInterface ⌈Access to shared calibration parameters can be provided and required even over CompositionSwComponentTypes using DelegationSwConnectors and AssemblySwConnectors.

This means that each access to calibration parameters between SwComponentPrototypes is explicitly visible.

If a SwConnector spans after the mapping of SwComponentPrototypes over two different ECUs the system generation process has to ensure the proper allocation of the ParameterDataPrototype while the calibration system has to cope with setting the parameter synchronously on the affected ECUs.⌋*(RS_SWCT_03190)*

2.2.3.2 Sharing Calibration Parameters between SwComponentPrototypes of the Same SwComponentType

To share calibration parameters between several SwComponentPrototypes of the same SwComponentType, a ParameterDataPrototype is attached to an SwcInternalBehavior in sharedParameter role (see [TPS_SWCT_01418]).

When the SwcInternalBehavior is aggregated by an AtomicSwComponentType the actual calibration parameters of the ParameterDataPrototype is the same for all SwComponentPrototypes.

[TPS_SWCT_01423] ParameterDataPrototype aggregated in the role constantMemory ⌈Additionally, it is possible to describe the implementation of shared characteristic values via a ParameterDataPrototype which is attached to an SwcInternalBehavior in the role constantMemory.

In contrast to the ParameterDataPrototype in sharedParameter role this kind of memory is not instantiated by the RTE. This supports more efficient implementations (especially for software components provided as object code) by avoidance of the additional indirection caused by the RTE's component data structure.⌋()

Further on this kind of memory reduces the dependencies of the software-component's implementation to generated RTE code which is appreciated for safety related functionalities.

Nevertheless, the information about these characteristic values has to be taken into account for the A2L file generation.

A typical example for this kind of sharing code between instances is dealing with two lambda sensors in multiple cylinder-bank engines, where (at least) two SwComponentPrototypes for each lambda sensor will use the very same Calibration Parameters.

2.2.3.3 Providing Instance Individual Characteristic Data

[TPS_SWCT_01424] ParameterDataPrototype aggregated in the role perInstanceParameter ⌈To provide instance individual calibration parameters a ParameterDataPrototype is owned by a SwcInternalBehavior in perInstanceParameter role.

When the SwcInternalBehavior is attached to an AtomicSwComponentType, the actual calibration values are specific for each SwComponentPrototype.⌋()

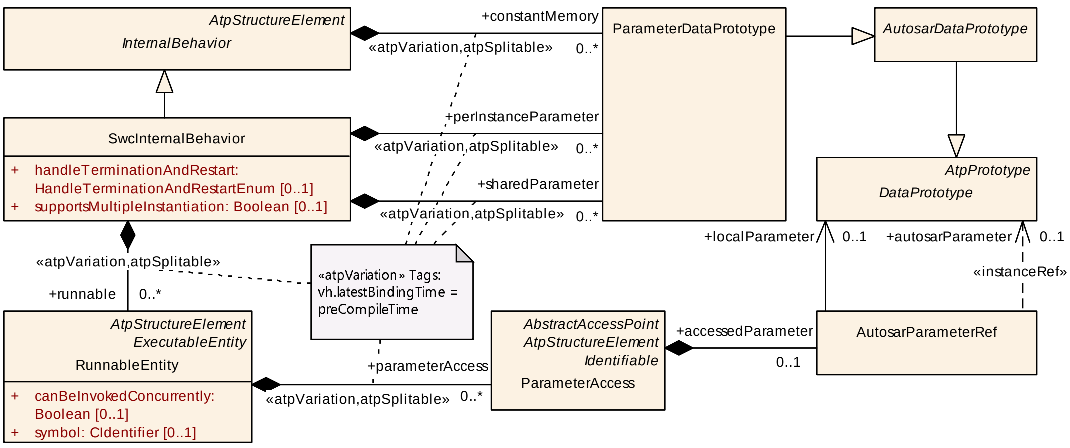


Figure 2.3: ParameterDataPrototypes in internal behavior

2.3 Runtime and Data Consistency Aspects

2.3.1 Background: the Issues

This section gives some background information and lists possible strategies concerning the implementation of the RunnableEntitys and the RTE with respect to efficient communication between the RunnableEntitys.

The communication among RunnableEntitys can very efficiently be implemented by means of “sharing memory”[1].

This is technically feasible because it is always guaranteed that the RunnableEntitys within an AtomicSwComponentType are always gathered at a specific processing unit (in other words: distribution is not an option).

Note that the purpose of communication among the RunnableEntitys is to establish a data flow scheme. The latter is a very popular pattern in the application of control theory to automotive embedded systems. So if “global variables” are used for establishing internal communication among RunnableEntitys they acquire the semantics of so called state-messages.

Nevertheless, directly sharing memory between RunnableEntitys requires a serious problem to be solved: the guarantee of data consistency among communicating RunnableEntitys. The RunnableEntitys will indeed be mapped to tasks so that one RunnableEntity of an AtomicSwComponentType may be preempted by a different RunnableEntity of the same AtomicSwComponentType.

Please note that a purist approach to achieving data consistency not only applies to single accesses of concurrently accessed variables. Rather, it would not be permitted that the value of a concurrently accessed variable (with state-message semantics) is unintentionally changed during the run-time of a RunnableEntity.

The following paragraphs describe some common strategies that can be used to ensure the required data-consistency. We do not attempt to describe the pros or cons of these approaches.

2.3.1.1 Mutual Exclusion with Semaphores

Multi-threaded operating systems provide mutexes (mutual exclusion semaphores) that protect access to an exclusive resource that is used from within several tasks.

The RTE could use these OS-provided mutexes to make sure that the RunnableEntitys sharing a memory-space would never run concurrently. The RTE would make sure the task running the RunnableEntity has taken an appropriate mutex before accessing the memory shared between the RunnableEntitys.

[1] Please note that the term “sharing memory” can be interpreted on different levels. It is e.g. in the C language possible to use variables with external linkage (a.k.a. “global variables”, although this term is not officially defined by the C language) for the purpose of inter-Runnable communication.

2.3.1.2 Interrupt Disabling

Another alternative would be the disabling of interrupts during the run-time of RunnableEntitys or at least for a period in time identical to the interval from the first to the last usage of a concurrently accessed variable in a RunnableEntity. This approach could lead to seriously non-deterministic execution timing.

2.3.1.3 Priority Ceiling

Priority ceiling allows for a non-blocking protection of shared resources. Provided that the priority scheme is static, the AUTOSAR OS is capable of temporarily raising the priority of a task that attempts to access a shared resource to the highest priority of all tasks that would ever attempt to access the resource.

By this means is technically impossible that a task in temporary possession of a resource is ever preempted by a task that attempts to access the resource as well.

2.3.1.4 Implicit Communication by Means of Variable Copies

Another alternative is the usage of copies of concurrently accessed variables with state message semantics. Note that this approach directly corresponds to the semantics of "implicit" sender-receiver communication (see 7.5.1.2).

This means in particular that for a concurrently used variable a copy is created on which a RunnableEntity entity can work without any danger of data inconsistency.

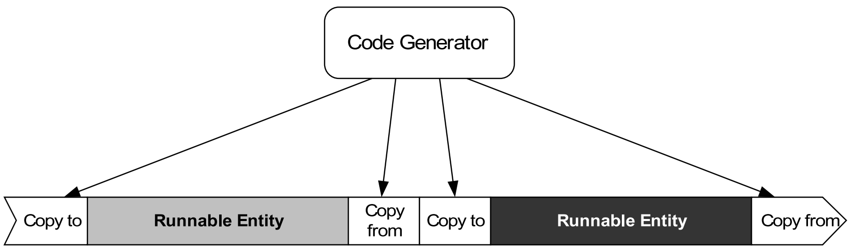


Figure 2.4: Generation of copy routines around RunnableEntitys

This concept requires additional code to write the value of the concurrently accessed variable to the copy before the RunnableEntity that accesses the variable is executed. The value of the copy shall be written back to the concurrently accessed variable after the RunnableEntity has been terminated.

This concept is sketched in Figure 2.4. Since it would be too expensive and error-prone to manually care about the copy routines it would be a good idea to leave the creation of the additional code to a suitable code generator.

The additional copy routines as sketched in Figure 2.4 already protect the particular RunnableEntitys from unintended changes of concurrently accessed variables. It would, however, be possible to further optimize the process by reducing the additional code at the beginning and end of each task (see Figure 2.5).

2.3.2 Data Consistency at Runtime

In addition, copy routines will only be inserted where appropriate, e.g. a copy routine for writing the value of a copy back to the concurrently accessed variable will only be inserted if the RunnableEntity has write-access to the concurrently used variable.

Please note that the copy routines have to temporarily make sure that the copy process is not interrupted in order to be capable of consistently copying the values from and to the concurrently accessed variable.

These periods, however, are supposed to be very short compared with the overall run-time consumption of the RunnableEntity and thus would not have a significant impact on the runtime behavior.

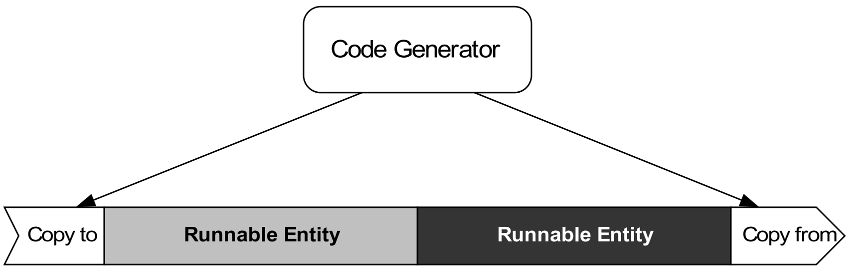


Figure 2.5: Optimized insertion of copy routines

Further optimization criteria can be applied, for example: it would be perfectly safe to avoid the creation of copies for RunnableEntitys that are scheduled in the task with the highest priority of all tasks that (via contained RunnableEntitys) access a certain concurrently accessed variable.

In order to keep the application code free of any dependencies from the code generation, access to concurrently accessed variables will be guarded by macros that are later resolved by the code generator.

The presence of the guard macros directly supports reuse on the level of source code. Reuse on the level of object code is only possible if the scheduling scenario (in terms of the assignment of RunnableEntitys to priority levels) does not change.

This concept can only be implemented properly with the aid of a code generator if the variables in question can be identified. In other words: the description of an AtomicSwComponentType has to expose all concurrently accessed variables to the outside world.

2.3.3 Modeling Aspects of Data Consistency

The intrinsic meaning of the terms “explicit communication” and “implicit communication” is explained in section 7.5.1.1. It would be fair to say that the distinction between implicit and explicit communication establishes a usage pattern in the application domain, i.e. in the world of the developer of AUTOSAR software-components and their implementation.

There is another facet to this subject, however, namely the question of how this pattern is implemented in the meta-model. With respect to the application of the pattern for port-based communication the details can be found in section 7.5.1.2, more specifically in section 7.5.1.3. The consideration of the internal communication based on so-called “inter-runnable variables” is described in section 7.4.2.

By reading the respective text sections it becomes apparent that the two applications of the pattern are modeled differently. The port-based communication uses the VariableAccess to formalize different roles of accessing communication elements. Some roles used for this purpose imply explicit communication (e.g. dataSendPoint) and some represent implicit communication (e.g. dataWriteAccess).

The important thing about using the VariableAccess, however, is that the modeling of communication roles is abstracted from the actual communication elements and represents a uniform (meaning: it can refer to the target directly or by a so-called InstanceRef) modeling approach that is applied for all use cases[2].

Admittedly, this is handled differently for the internal communication. Here, the additional layer of abstraction is not used (although it would have been technically feasible to do so) with respect to the clear separation of “inter-runnable variables with implicit behavior” and “inter-runnable variables with explicit behavior” in the RTE.

The implementation of different communication roles (i.e. implicit vs. explicit) is done by directly aggregating VariableDataPrototype in the roles explicitInterRunnableVariable and implicitInterRunnableVariable.

On the other hand, access to internal communication **never** requires the usage of an InstanceRef and therefore the abstraction might be considered unnecessary overhead that blows up the M1 model.

2.4 Variant Handling in the Software Component Template

The Software Component Template supports the creation of *Variant*s in a subset of its model elements. The full list of model elements that support variation can be found in the appendix.

[2]On a related note, even for non-communication related data access the same pattern applies implemented by ParameterAccess

[TPS_SWCT_01038] Support for Variant Handling in the in Software Component Template ⌈The Variant Handling support in the in Software Component Template is mainly driven by the purpose to describe a variable system on Virtual Functional Bus[3] level by varying

- the existence of SwComponentPrototypes
- the existence of SwConnectors
- the existence of Chapters of SwComponentDocumentation
- the existence of PortPrototypes

⌋*(RS_SWCT_00220, RS_SWCT_03140, RS_SWCT_03142, RS_SWCT_03154)*

This approach supports adjusting the number and kind of software-component instances as well as their interconnection in a particular system variant.

[TPS_SWCT_01447] Applicable binding times for model elements in the scope of the Software Component Template ⌈The first three cases are supporting *Post-Build* binding. For the existence of PortPrototypes only preCompileTime is supported as latest Binding Time.⌋*(RS_SWCT_00220)*

[TPS_SWCT_01040] SwConnector exists depending on a *PostBuild* condition ⌈A SwConnector which exists depending on a *PostBuild* condition has an impact on the behavior of API function calls that apply on a PortPrototype to which the SwConnector is attached.

If the SwConnector does not exist, the behavior of the RTE API functions need to take this into account.

This means that the RTE implementation of this PortPrototype resembles the behavior of an unconnected PortPrototype.⌋*(RS_SWCT_00220, RS_SWCT_03100, RS_SWCT_03143)*

Please find more details in the specification of the RTE [2].

[TPS_SWCT_01041] API functions of not existing SwConnector are still part of the software-component's implementation ⌈If SwConnectors do not exist, the corresponding API functions are still part of the software-component's implementation. It is not possible to remove the API functions in a *PostBuild* step.

Therefore, the latest reasonable Binding Time for the conditional existence of a PortPrototype is preCompileTime.⌋*(RS_SWCT_00220, RS_SWCT_03100)*

[TPS_SWCT_01085] Variation on the behavior level ⌈In addition to variation of the VFB-related model elements, the description of variant software-component implementations is supported. Please note that this requires a broad support of variability in the *Internal Behavior*.

The identified main use case are

- the existence of RunnableEntitys

- the existence of RTEEvents
- the existence of VariableDataPrototypes in the roles implicitInterRunnableVariable and explicitInterRunnableVariable
- the existence of ParameterDataPrototypes in the roles perInstanceParameter, sharedParameter, and constantMemory

⌋*(RS_SWCT_03149, RS_SWCT_03150, RS_SWCT_03151, RS_SWCT_03153)*

For the same reason that applies on the existence of PortPrototype the latest Binding Time of these kinds of variability is preCompileTime.

In the meta-model, all locations that may exhibit variability are marked with the stereotype ≪atpVariation≫. This allows the definition of possible variation points. Tagged Values are used to specify additional information, for example the latest binding time.

[TPS_SWCT_01042] Four types of locations in the meta-model which may exhibit variability ⌈There are four types of locations in the meta-model which may exhibit variability:

- Aggregations
- Associations
- Attribute Values
- Classes providing property sets

⌋*(RS_SWCT_00220, RS_SWCT_03100)*

The reasons for the attachment of the stereotype ≪atpVariation≫ to certain model elements and the consequences for other model elements are explained in class tables in the following chapters.

More details about the AUTOSAR Variant Handling Concept can be found in the AUTOSAR Generic Structure Template [11].

2.5 Communication Specification of Composition Component Types

[TPS_SWCT_01088] ComSpecs defined by CompositionSwComponentTypes ⌈It shall be possible to attach ComSpecs to PortPrototypes owned by CompositionSwComponentTypes.⌋*(RS_SWCT_03220)*

2.5.1 Rationale

ComSpecs attached to a PortPrototype owned by an AtomicSwComponentType have a direct impact on the generation of the RTE. The RTE Generator, on the other hand, does not consider the existence of CompositionSwComponentTypes.

Nevertheless, there are some cases where the definition of a ComSpec attached to a PortPrototype owned by a CompositionSwComponentType does make sense.

That is, in case an OEM wants to submit the definition of a CompositionSwComponentType to a supplier for adding more details and implementing the behavior the OEM might want to point out that from the OEM's point of view sender initValues and receiver initValues apply for the elements of PortInterfaces used to type the delegation PortPrototypes.

The idea is that the supplier takes over the initValues attached to the delegation PortPrototypes and *copies* them to the PortPrototypes owned by SwComponentPrototypes of the CompositionSwComponentType.

[TPS_SWCT_01568] Consideration of RPortComSpec or PPortComSpec depending on the ownership ⌈The RTE Generator shall take the attributes of the RPortComSpec or PPortComSpec of the PortPrototypes owned by AtomicSwComponentTypes or ParameterSwComponentType and ignore the attributes of the RPortComSpec or PPortComSpec attached to PortPrototypes owned by CompositionSwComponentType.⌋*(RS_SWCT_03220)*

Therefore, the initValues of the delegation PortPrototype would be taken as *mere templates* for the detailing of PortPrototypes connected to the delegation PortPrototypes.

It is not required that the initValues of delegated PortPrototype and a PortPrototype connected by means of a DelegationSwConnector match.

Although this would certainly make sense in many cases, it is eventually still left to the supplier to decide on the specific initValues applicable inside the CompositionSwComponentType.

On the other hand, a requirement that the initValues defined on the surface of CompositionSwComponentType and the inside of the CompositionSwComponentType shall be consistent in any case might effectively prevent reuse of existing AtomicSwComponentTypes.

Please note that the ability to define a ComSpec in the context of a CompositionSwComponentType implies that it shall be possible to define mappings of ApplicationDataTypes used in a PortInterface to their corresponding ImplementationDataTypes.

For this purpose the CompositionSwComponentType owns a DataTypeMappingSet in the role dataTypeMapping and a ConstantSpecificationMappingSet in the role constantValueMapping.

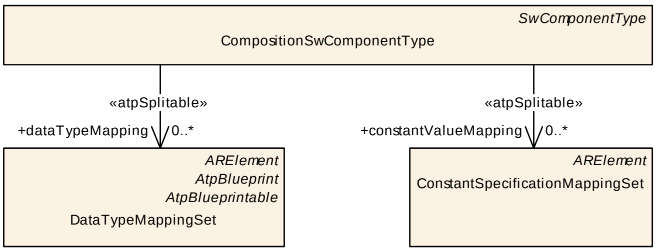


Figure 2.6: Specification of data type mapping for CompositionSwComponentType

2.6 PRPortPrototype

In some cases SwComponentTypes need to read and write the same piece of data. One of the most prominent examples for this use case is the NvBlockSwComponentType that factually ready and writes blocks of NvRAM.

Without the ability to combine read and write semantics in a kind of PortPrototype that supports both read and write semantics, work-arounds have to be implemented that come with a certain footprint on memory and processing time.

2.6.1 Use Case 1

Without the ability to define a combined read and write semantics the definition of an RPortPrototype and a PPortPrototype is required for reading and writing the applicable data.

Technically, this read and write access is related to the same data item in an NVRAM Block. This requires a consistent connection of the PortPrototypes between an NvBlockSwComponentType and ApplicationSwComponentType as well as a consistent mapping of the corresponding RPortPrototype and a PPortPrototype of the NvBlockSwComponentType and the related element of the ramBlock.

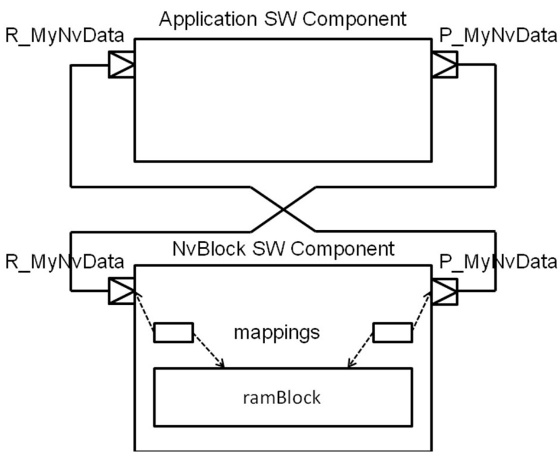


Figure 2.7: Use Case 1 for the existence of PRPortPrototype

2.6.2 Use Case 2

It may happen that a SwComponentType need to consume the same data that it produces. If the only way to achieve this was the connection of a PPortPrototype to an RPortPrototype of the same SwComponentType then the creator of the SwComponentType cannot enforce this connection as it is created on a higher level of abstraction in the context of a CompositionSwComponentType.

In other words, it is impossible to fully specify the semantics of the otherwise self-contained SwComponentType.

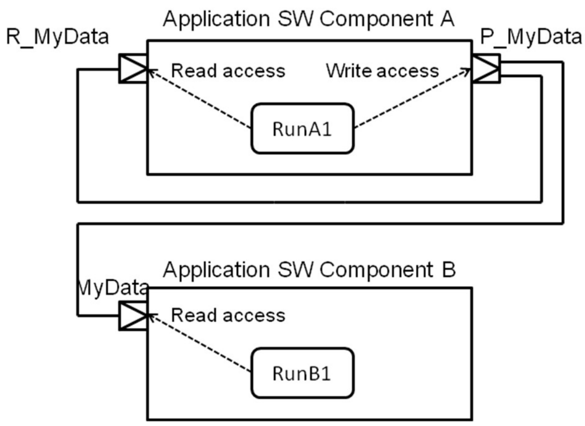


Figure 2.8: Use Case 2 for the existence of PRPortPrototype

This means that only in the in best case one buffer for the data is needed. But depending on the mapping RunnableEntitys to OS tasks additional buffers may need to be allocated by the RTE to fully implement the implicit communication pattern.

As an alternative, the ApplicationSwComponentType could utilize inter-runnable variables but unfortunately this inhibits any optimization in the RTE and will consume additional RAM. In contrast to the previous approach at least two buffers are needed.

2.6.3 Use Case 3

In this scenario, several ApplicationSwComponentTypes are iterating over the same large set of data. This means each ApplicationSwComponentType implements one out of many steps of a complex data processing algorithm applied to the same piece of data.

For example, this scenario may apply for video signal processing in camera applications. Typically, such applications will **not** be distributed over several ECUs.

It is clear that in this case the allocation of several buffers in the RTE is required to implement the individual connections between the ApplicationSwComponentTypes. In most cases, the processing has to be executed at a certain point in time in a dedicated order.

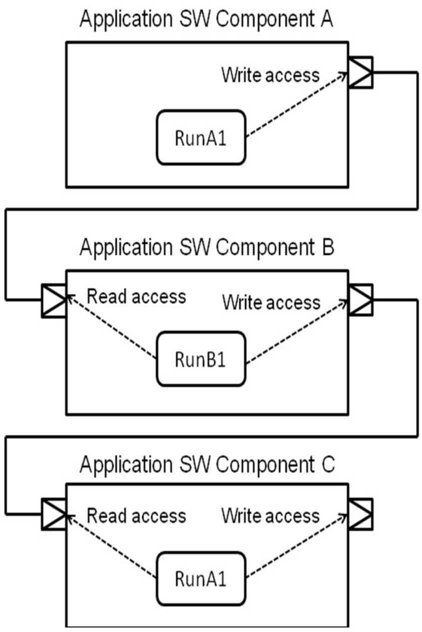


Figure 2.9: Use Case 3 for the existence of PRPortPrototype

2.6.4 Solution

The solution to the above-mentioned use cases is the ability to define a PortPrototype that can read and write the same piece of data. This solves both the described problem of resource consumption and the problem of having to define multiple PortPrototypes as outlets for same piece of data item.

The technical details of the definition of PRPortPrototype are explained in chapters 3.2.2 and 4.2.1.

2.7 Variable-size Array Data Types

2.7.1 Overview and Use cases

AUTOSAR supports the definition of array data types where the size of the actual payload varies at run-time. As far as the configuration is concerned, it is possible to specify a maximum number of array elements that shall not be exceeded at run-time.

In order to properly understand the approach, it is necessary to understand that the support for Variable-Size Array Data Types has been introduced in two waves that each had a different motivation.

2.7.1.1 "Old-world" dynamic-size Arrays

In the first wave, the support for Variable-Size Array Data Types was limited to data types that basically boil down to an array where the base type is an unsigned integer data type with a length of exactly one byte.

The main use cases for this scenario are derived from diagnostics requirements as well as support for the J1939 communication protocol.

In both cases the actual length of a Variable-Size Array Data Type could be determined from the context, i.e. either by the diagnostic basic-software module or by the implementation of the J1939 TP.

For the lack of a better terminology, this specification distinguishes between "old-world" dynamic-size arrays and "new-world" Variable-Size Array Data Types. It will be necessary to clearly define the characteristics that allow for a disambiguation between the "old-world" dynamic-size arrays and "new-world" Variable-Size Array Data Types.

[TPS_SWCT_01641] Definition of an "old-world" dynamic-size array data type by means of an ApplicationArrayDataType ⌈An ApplicationArrayDataType that **doesn't define** attribute dynamicArraySizeProfile **and** that aggregates an ApplicationArrayElement where attribute arraySizeSemantics exists and is set to the value variableSize shall be considered an "old-world" dynamic-size array data type.⌋*(RS_SWCT_03181)*

Please note that [TPS_SWCT_01641] can't go any deeper into the specifics of the given data type because it is intentionally focused on ApplicationDataTypes. There are use cases where the distinction between "old-world" dynamic-size arrays and "new-world" Variable-Size Array Data Types shall be done in the absence of a corresponding ImplementationDataType.

In general, the disambiguation becomes multi-faceted (but not necessarily easier) if the definition of a corresponding ImplementationDataType is available (see [TPS_SWCT_01642]).

[TPS_SWCT_01642] Definition of an "old-world" dynamic-size array data type by means of an ImplementationDataType ⌈An ImplementationDataType that (after all type references are resolved) fulfills all the following conditions shall be considered an "old-world" dynamic-size array data type:

- The value of attribute category is set to ARRAY
- The ImplementationDataType doesn't define the attribute dynamicArraySizeProfile
- The ImplementationDataType aggregates a subElement where
  - attribute arraySizeSemantics exists and is set to the value variableSize
  - attribute arraySizeHandling does not exist

- One of the following conditions applies:
  - subElement.category is set to VALUE or TYPE_REFERENCE that eventually boils down to VALUE and the attribute subElement.swDataDefProps.baseType.baseTypeDefinition.baseTypeSize is set to the value 8 and the attribute baseTypeEncoding is set to NONE.
  - subElement.category is set to TYPE_REFERENCE and the attribute subElement.swDataDefProps.implementationDataType literally represents the Platform Data Type named uint8.
  - subElement.category is set to TYPE_REFERENCE and the attribute subElement.swDataDefProps.implementationDataType.shortName is set to uint8 and subElement.swDataDefProps.baseType.baseTypeDefinition.nativeDeclaration does not exist.

⌋*(RS_SWCT_03181)*

By and large, the defining characteristics for "old-world" dynamic-size arrays is the **absence** of a definition of the attribute ApplicationArrayDataType.dynamicArraySizeProfile or ImplementationDataType.dynamicArraySizeProfile.

By regulation of [constr_1387], "old-world" dynamic-size arrays are not supported for transmission by means of a data transformer. The only supported kind of Variable-Size Array Data Type that can be transmitted using a data transformer is the "new-world" variable-size arrays.

2.7.1.2 "New-world" variable-size Arrays

In contrast to this, the second wave of support for Variable-Size Array Data Types was motivated by the application software layer itself.

Here, the situation is entirely different because the actual size cannot be determined by any context software module. The application itself is responsible for maintaining the proper length of a Variable-Size Array Data Type at run-time.

As a consequence, the specification of the actual array size at run-time needs to be reflected by the structure of the data types used for hosting the Variable-Size Array Data Type.

[TPS_SWCT_01644] Definition of a "new-world" variable-size array data type by means of an ApplicationArrayDataType ⌈An ApplicationArrayDataType that fulfills all the following conditions shall be considered an "new-world" dynamic-size array data type.

- The ApplicationArrayDataType **defines** attribute ApplicationArrayDataType.dynamicArraySizeProfile.
- ApplicationArrayDataType aggregates an ApplicationArrayElement that **defines** attribute ApplicationArrayElement.arraySizeHandling.

⌋(RS_SWCT_03181)

[TPS_SWCT_01645] Definition of a "new-world" variable-size array data type by means of an ImplementationDataType ⌈An ImplementationDataType that fulfills all the following conditions shall be considered an "new-world" dynamic-size array data type.

- The ImplementationDataType **defines** attribute ImplementationDataType.dynamicArraySizeProfile.
- ImplementationDataType aggregates an ImplementationDataTypeElement that **defines** attribute ImplementationDataTypeElement.arraySizeHandling.

⌋(RS_SWCT_03181)

In contrast to the first use case described above, the application-motivated Variable-Size Array Data Type cannot be limited in terms of the base type of the array data type, i.e. limiting the underlying data type to an unsigned integer data type with a length of exactly one byte is not an option.

On top of that, several possible structures of Variable-Size Array Data Types have been required. This aspect is depicted in Figure 2.10.

[TPS_SWCT_01636] Definition of profiles for the definition of Variable-Size Array Data Types ⌈The possible variants for Variable-Size Array Data Types are:

Linear The data type of the elements of the Variable-Size Array Data Type itself does not consist of a Variable-Size Array Data Type.

This case corresponds to the possible value **VSA_LINEAR** of attribute dynamicArraySizeProfile.

Square The data type of the elements of the Variable-Size Array Data Type itself consists of Variable-Size Array Data Types where the maximum number of elements in all "second order" arrays is **identical** to the maximum number of elements in the "first order" array.

This case corresponds to the possible value **VSA_SQUARE** of attribute dynamicArraySizeProfile.

Rectangular The data type of the elements of the Variable-Size Array Data Type itself consists of Variable-Size Array Data Types where the maximum number of elements in "second order" arrays is **identical** but this value is typically **not identical**[3] to the maximum number of elements in the "first order" array.

This case corresponds to the possible value **VSA_RECTANGULAR** of attribute dynamicArraySizeProfile.

[3] If it was, the case boils down to the rectangular scenario tagged (b).

Fully Flexible The data type of the elements of the Variable-Size Array Data Type itself consists of Variable-Size Array Data Types where the maximum number of elements in "second order" arrays is **not necessarily identical** with each other and (obviously) **not necessarily identical** to the maximum number of elements in the "first order" array.

This case corresponds to the possible value **VSA_FULLY_FLEXIBLE** of attribute dynamicArraySizeProfile.

⌋*(RS_SWCT_03181)*

The described cases directly correspond to the portrayal of different kinds of variable-size arrays in Figure 2.10:

- The value VSA_LINEAR corresponds to the tag (a).
- The value VSA_SQUARE corresponds to the tag (b).
- The value VSA_RECTANGULAR corresponds to the tag (c).
- The value VSA_FULLY_FLEXIBLE corresponds to the tag (d).

Please note that the leaf elements in a Variable-Size Array Data Type doesn't have to be primitive data types. As mentioned before, it is possible to define multiple-dimension Variable-Size Array Data Types.

The "terminal" elements can be recognized as such in that they don't establish further Variable-Size Array Data Types.

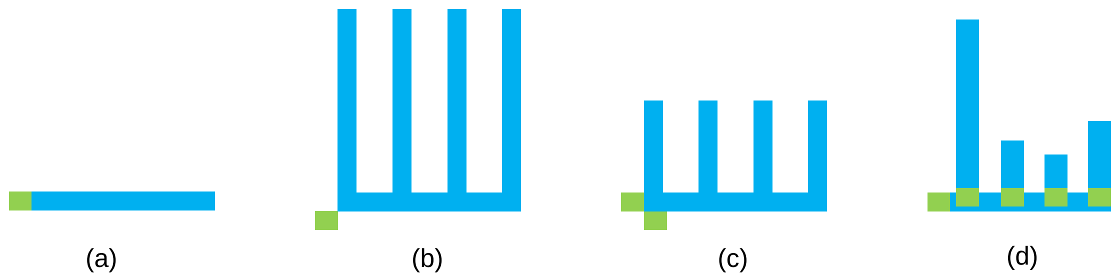


Figure 2.10: Structural variety of array data types with variable size

Please note further that the modeling of Variable-Size Array Data Types is a complex step governed by a collection of rules and constraints.

It is the expressed intent of this specification to keep the complexity of the rule set as low as possible while still providing the user with a powerful modeling framework.

The major consequence of this conclusion is to keep the modeling as straightforward as possible; in other words: intentionally cut away certain modeling variants for which acceptable workarounds within the modeling framework itself exist.

One concrete example for such a restriction is that for ImplementationDataTypes, Variable-Size Array Data Types can only be defined on the level of an AutosarDataType.

It is intentionally not supported to define a Variable-Size Array Data Type on the level of an ImplementationDataTypeElement because the intended semantics can be realized by assigning the value TYPE_REFERENCE to the ImplementationDataTypeElement.category and then let it reference to another ImplementationDataType that in turn implements the Variable-Size Array Data Type.

2.7.2 Modeling Aspects regarding Application Data Types

In the context of the AUTOSAR layered data type concept, the level of ApplicationDataTypes is not concerned about the structure of how the Variable-Size Array Data Types.

In other words, aspects of the implementation of this kind of data type is intentionally abstracted as much as possible in order to support the idea behind the definition of ApplicationDataTypes as a concept that is independent of an implementation to the applicable degree.

Consequently, the support for Variable-Size Array Data Types on the level of ApplicationDataTypes requires the addition of a couple of additional attributes. Details can be found in chapter 5.2.4.2.

If a Variable-Size Array Data Type is modeled on the level of ApplicationDataType it is necessary to also provide a companion ImplementationDataType as well as a DataTypeMap that refers to both the ApplicationDataType and the ImplementationDataType.

The contrary is **not applicable**, i.e. it is possible to define a Variable-Size Array Data Type with only an ImplementationDataType, see [TPS_SWCT_01622].

2.7.3 Modeling Aspects regarding Implementation Data Types

On the other hand, the data type used for the actual hosting of the Variable-Size Array Data Type corresponds directly to the level of the ImplementationDataType.

Here, it is possible to define how an ImplementationDataType can be used to define a Variable-Size Array Data Type.

The definition of ImplementationDataType in the AUTOSAR meta-model comes with a certain level of generic nature the support for Variable-Size Array Data Types on this level comes as a mixture of dedicated attributes in the meta-model and a set of recipes how to support different use cases of Variable-Size Array Data Types.

This means that the definition of ImplementationDataTypes for the purpose of creating Variable-Size Array Data Types only has a chance to take off if the structure of these data types is replicated in different implementations of AUTOSAR software.

Therefore, AUTOSAR defines a common way of how ImplementationDataTypes for the purpose of creating Variable-Size Array Data Types shall be defined such that the ImplementationDataType shall be of category STRUCTURE with the following sub-elements:

1. A numerical value that determines the actual size. This element shall be called the Size Indicator throughout this document.
2. An array of the base-type of the Variable-Size Array Data Type that implements the payload of the Variable-Size Array Data Type. The dimension of the array shall be defined such that the intended maximum number of elements fits in.

A Size Indicator of a Variable-Size Array Data Type holds the number of valid elements of the array. This information is necessary for the RTE to handle the array efficiently.

On the sender-side this indicator is actively updated by the software-component, which is the only instance that knows how many elements of the array are valid.

So the number of valid elements and the Size Indicator have to be kept consistent by the application. When the software-component sends the data over to the RTE, the RTE hands the data over to the transformer.

The transformer may evaluate the Size Indicator (depends on the transformer) and only work on the valid array elements. The output of the transformer can vary in length and only contain necessary data. Therefore, it can be more resource saving.

On the receiver side, the last transformer in the execution order restores the data elements of the array and the value of the Size Indicator. This output is handed over by the RTE to the software-component. The application is now aware of the number of valid elements in the array.

The details of how ImplementationDataTypes need to be modeled for the implementation of Variable-Size Array Data Types can be found in chapter 5.2.5 and a couple of examples is available in the appendix E.1.

2.8 Optional Elements in Structures

2.8.1 Background

The *AUTOSAR classic platform* supports the usage of a TLV[4] data encoding on the SOME/IP transport layer. TLV is typically used where at least a part of the transmitted data is only *optionally* existing and filled with meaningful values.

In other words: an optional part of a data structure may exist and carry meaningful values in one instance of data transmission and be completely missing in another instance of the data transmission.

The receiving software needs to be able to identify whether the optional part exists and read its value accordingly.

The receiving software also needs to be able to still execute meaningfully if the optional part of such a data structure does not exist in the specific communication instance.

Consequently, it is necessary to be able to precisely identify the parts of a data structure that may become optional for specific instances of data transmission.

In terms of the AUTOSAR meta-model, the identification could - in principle - be attached at various levels of abstraction:

AutosarDataType In this case the optionality that is only needed for communication purposes would still be existing in all other usages of data types. This seems unbalanced.

Admittedly, the definition of different optionality configurations for the same data type may lead to the existence of a bunch of structurally identical data types that only vary in terms of optionality. The existence of variation points may help to mitigate this effect, though.

PortInterface In this case the optionality is defined where it is actually required. However, different optionality could - in principle - be defined for DataPrototypes typed by the same AutosarDataType.

This would lead to an increased effort for the definition of C data types in the context of the same PortInterface.

Additional constraints have been identified in the context of the definition of RTE APIs of the AUTOSAR classic platform that finally render this option as not viable.

ComSpec In this case (for more information please refer to section 4.5) the definition of optionality would even be more specific in comparison to the definition of optionality on the level of PortInterfaces.

[4]This abbreviation stands for tag-length-value

On top of that, the task to define optionality in the vast majority of cases is done by an OEM, whereas the model definition on the level of ComSpec requires the existence of SwComponentTypes and this definition is in many cases in the domain of a supplier.

As a result of this consideration, AUTOSAR has opted for implementation of the concept of defining the optionality on the level of the AutosarDataType.

3 Overview: Software Components, Ports, and Interfaces

3.1 Introduction

The detailed introduction of all aspects of the `Software Component Template` in one move is considered too complex. This chapter therefore provides an overview of the main conceptual aspects of software components, ports and interfaces.

The overview will then be broken down into further details in chapter 4.

One of the goals of the AUTOSAR concept is the support of re-usability on the level of application software.

In other words: it should be possible to re-use existing artifacts to create further model elements instead of being forced to create every single modeling detail from scratch.

One of the consequences of this approach is the application of the so-called type-prototype pattern [11].

The type-prototype pattern is created by using references from prototype to type that are decorated with the ≪isOfType≫. The existence of such references is governed by [constr_2633], see [11].

Among other things, this concept allows for creating hierarchical structures of software-components with arbitrary complexity. However, the creation of hierarchical structures itself does not have an impact on the run-time behavior of the overall system.

The actual behavior is completely defined within the individual software-components.

This conclusion is backed by the understanding that software-components are developed against the so-called *Virtual Functional Bus* (VFB), an abstract communication channel without direct dependency on ECUs and communication buses.

The VFB does not provide any means for expressing a hierarchy of software-components.

Of course, the usage of the VFB has further consequences on the design of software-components which shall not directly call the operating system or the communication hardware.

As a result, software-components can be deployed to actual ECUs at a rather late stage in the development process.

In order to make the description more precise, the following text preferably uses accurate meta-model terms instead of the rather vague terminology of “composition” and “software-component”.

3.2 Software Component

3.2.1 Overview

Application software within AUTOSAR is organized in self-contained units called AtomicSwComponentTypes. Such AtomicSwComponentTypes encapsulate the implementation of their functionality and behavior and merely expose well-defined connection points, called PortPrototypes, to the outside world.

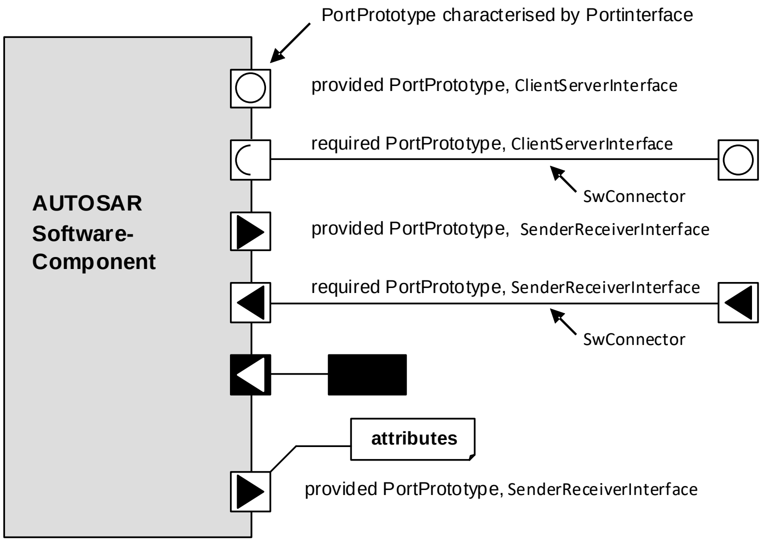


Figure 3.1: Graphical representation of software-components in AUTOSAR

The graphical appearance of AUTOSAR software-components according to [3] is depicted in Figure 3.1.

Class	***SwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of ConsistencyNeeds owned by the enclosing SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=consistencyNeeds.shortName, consistency Needs.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	***SwComponentType*** (abstract)			
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponent Type can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=port.shortName, port.variationPoint.short Label vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portGroup.shortName, portGroup.variation Point.shortLabel vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this Sw ComponentType.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which UnitGroups are relevant in the context of referencing SwComponentType.

Table 3.1: SwComponentType

3.2.2 PortPrototype

Please note that `PortPrototype`s of a `SwComponentType` are supposed to be used for attaching `SwConnector`s that establish an actual connection between `SwComponentPrototype`s (see chapter 3.3).

[TPS_SWCT_01002] `SwComponentType`s may only interact by means of their `PortPrototypes` ⌈`AtomicSwComponentType`s (and also the more general `SwComponentType`s may only interact by means of their `PortPrototype`s). Hidden communication dependencies that are *not* expressed by means of `PortPrototype`s are strictly forbidden.⌋*(RS_SWCT_00020, RS_SWCT_00030, RS_SWCT_00150, RS_SWCT_00160, RS_SWCT_00200, RS_SWCT_00210, RS_SWCT_02010, RS_SWCT_02030)*

Therefore, software-components are in theory exchangeable as long as they implement the same functionality and provide the same public communication interface to the remaining system.

[TPS_SWCT_01111] `PortPrototype`s need an additional model artifact, the `PortInterface` ⌈Please note that `PortPrototype`s actually need an additional model artifact, the PortInterface, for fully describing the details of the PortPrototype.⌋(RS_SWCT_00010)

The concept of the PortInterface as another means for establishing a high degree of re-usability is described in chapter 3.4.

Class	***PortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clientServerAnnotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPortAnnotation	DelegatedPortAnnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstractionServerAnnotation	IoHwAbstractionServerAnnotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortAnnotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPortAnnotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table 3.2: PortPrototype

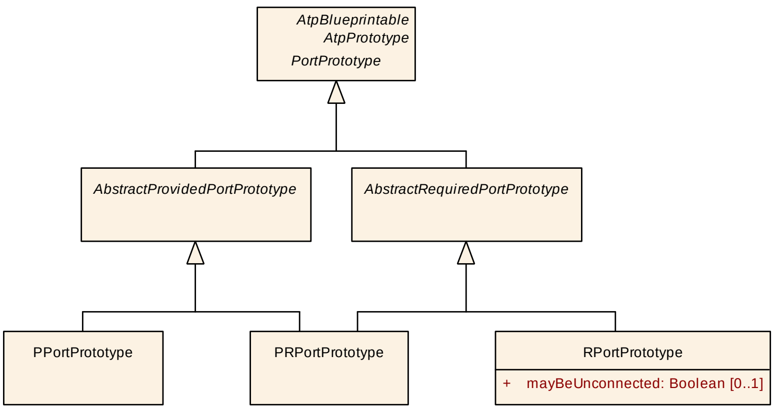


Figure 3.2: Overview of PortPrototype

[TPS_SWCT_01112] Semantics of PortPrototypes ⌈PortPrototypes can have the following semantics:

- A require-port (in technical terms: RPortPrototype) requires certain services or data.

- A provide-port (or PPortPrototype) on the other hand provides services or data.
- A provide-require-port (or PRPortPrototype) combines the ability to provide and require services or data in one entity.

⌋*(RS_SWCT_03250)*

The semantics of PortPrototype is also depicted in Figure 3.2,

[TPS_SWCT_01573] A PRPortPrototype is never considered unconnected ⌈A PRPortPrototype is never considered unconnected, even if there are no SwConnectors actually referring to it.⌋*(RS_SWCT_00010, RS_SWCT_03250, RS_SWCT_03130)*

Please note that [TPS_SWCT_01573] represents the immediate consequence of the semantics defined in [TPS_SWCT_01112].

[TPS_SWCT_01113] Connecting two PortPrototypes ⌈Two SwComponentPrototypes are eventually connected by hooking up a PPortPrototype or PRPortPrototype of one SwComponentPrototype to a compatible RPortPrototype or PRPortPrototype of the other SwComponentPrototypes.⌋*(RS_SWCT_03130, RS_SWCT_03250)*

Please find more information concerning the definition of “compatibility” in section 6.

Class	***AbstractRequiredPortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This abstract class provides the ability to become a required PortPrototype.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Subclasses	PRPortPrototype, RPortPrototype			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
requiredComSpec	RPortComSpec	*	aggr	Required communication attributes, one for each interface element.

Table 3.3: AbstractRequiredPortPrototype

Class	***AbstractProvidedPortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This abstract class provides the ability to become a provided PortPrototype.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Subclasses	PPortPrototype, PRPortPrototype			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
providedComSpec	PPortComSpec	*	aggr	Provided communication attributes per interface element (data element or operation).

Table 3.4: AbstractProvidedPortPrototype

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
mayBe Unconnected	Boolean	0..1	attr	If set to true, this attribute indicates that the enclosing RPortPrototype may be left unconnected and that this aspect has explicitly been considered in the software-component's design.
required Interface	PortInterface	0..1	tref	The interface that this port requires. **Stereotypes:** isOfType

Table 3.5: RPortPrototype

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
provided Interface	PortInterface	0..1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table 3.6: PPortPrototype

Class	**PRPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This kind of PortPrototype can take the role of both a required and a provided PortPrototype.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
provided Required Interface	PortInterface	0..1	tref	This represents the PortInterface used to type the PRPort Prototype **Stereotypes:** isOfType

Table 3.7: PRPortPrototype

[TPS_SWCT_01096] **PortGroup** ⌈PortPrototypes can be logically grouped into PortGroups. This mechanism is used for implementing mode management features.⌋ *(RS_SWCT_03201)*

Further explanations about the semantics of meta-class PortGroup can be found in chapter 4.6.

There are cases where an RPortPrototype is intentionally left unconnected. Such a scenario would typically be reported if the RTE Generator is executed in strict mode.

By means of setting attribute RPortPrototype.mayBeUnconnected, the designer of the respective software-component has the ability to express that the "open" RPortPrototype is part of the final design and shall not be reported, even in strict mode.

The attribute can be used in combination of an RPortPrototype and any kind of PortInterface.

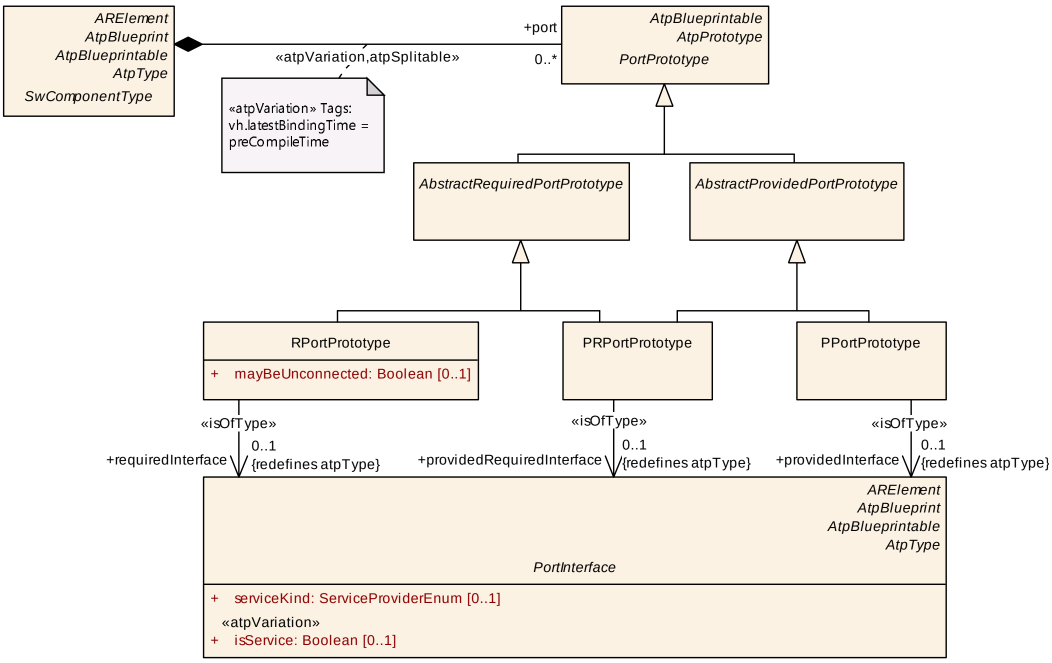


Figure 3.3: Components and Ports

Note that the attribute applies to the entire RPortPrototype, not to individual elements of the applicable PortInterface.

In other words, the attribute does not affect the case, where e.g. only one dataElement of several sender/receiver RPortPrototype is left unconsidered, as described by [TPS_SWCT_01101].

Please note that the usage of RPortPrototype.mayBeUnconnected is potentially dangerous because it removes a warning and this can be harmful if the suppression of a legitimate warning were done by mistake.

It is therefore advised to handle the existence of RPortPrototype.mayBeUnconnected with care.

3.2.3 AtomicSwComponentType

[TPS_SWCT_01108] Added value of an AtomicSwComponentType ⌈As mentioned before, the term AtomicSwComponentType is a specific form of the general concept of the SwComponentType. The added value of an AtomicSwComponentType is that it can aggregate an InternalBehavior⌋*(RS_SWCT_03040)*

More information regarding the semantics of InternalBehavior can be found in chapter 7.

[TPS_SWCT_01109] Adding the SwcInternalBehavior in a later process step
⌈The aggregation of SwcInternalBehavior is stereotyped ≪atpSplitable≫ to allow for adding the SwcInternalBehavior in a later process step. In other words, it is possible to completely develop the VFB view of a software-component and later add more details like InternalBehavior.⌋*()*

Class	***AtomicSwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponentType, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponentType, ServiceSwComponentType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is <<atpSplitable>>. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior.shortName, internalBehavior.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbolProps.shortName

Table 3.8: AtomicSwComponentType

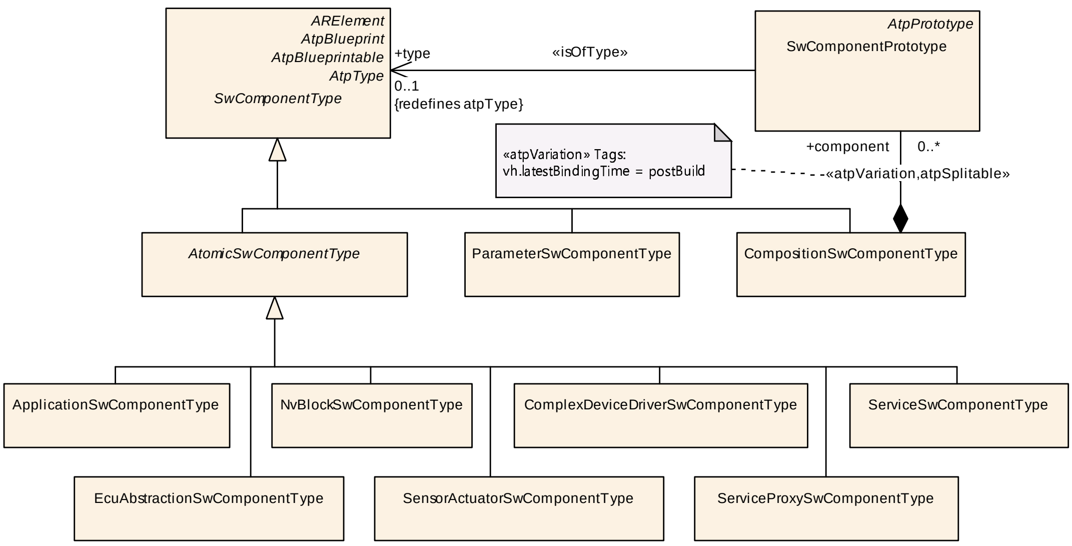


Figure 3.4: Overview of Component Types

There are several specialized SwComponentTypes to describe specific software-components used in the different parts of the AUTOSAR Layered Architecture [5]. Further details are mentioned in chapter 10 and 11.

The ApplicationSwComponentType is a specialization of AtomicSwComponentType for representing hardware-independent application software. The ParameterSwComponentType is a specialization of SwComponentType that can - in contrast to AtomicSwComponentType - not aggregate SwcInternalBehavior.

The purpose of the NvBlockSwComponentType is described in detail in section 11.4.2. The ServiceSwComponentType is described in section 11.2. Further on, the EcuAbstractionSwComponentType and the ComplexDeviceDriverSwComponentType are discussed in detail in section 10.

A description of the ServiceProxySwComponentType can be found in section 11.3 while the SensorActuatorSwComponentType is described in section 10.4.

Class	ApplicationSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 3.9: ApplicationSwComponentType

3.2.4 ParameterSwComponentType

[constr_1092] Restrictions for the ParameterSwComponentType ⌈The following restrictions exist for a ParameterSwComponentType:

- it shall never aggregate a SwcInternalBehavior and
- the only aggregated PortPrototypes shall be PPortPrototypes of type ParameterInterface.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01847] Define role-specific data properties of elements of composite data types used for the definition of calibration parameters ⌈A ParameterSwComponentType shall have the ability to aggregate InstantiationDataDefProps. By this means it is possible to define role-specific data properties of elements of composite data types used for the definition of calibration parameters in the scope of a ParameterSwComponentType.⌋*()*

For more information about this aspect please refer to section 7.5.4.

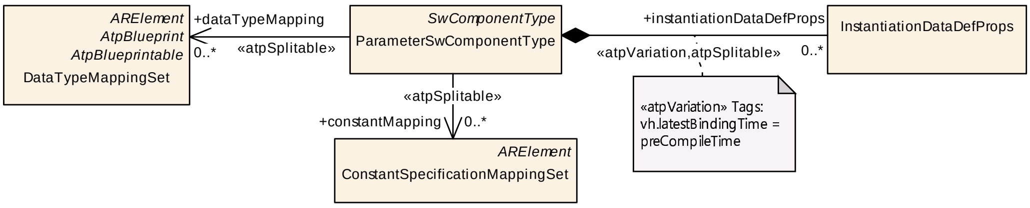


Figure 3.5: Details of ParameterSwComponentType

3.2.5 Symbolic Name of a Software-Component

Please note that an AtomicSwComponentType manifests itself in the source code of an RTE into which an instance of the AtomicSwComponentType is deployed. This implies potential naming conflicts if instances of AtomicSwComponentType that have identical shortNames are deployed into a specific RTE.

[TPS_SWCT_01110] Symbolic name of a software-component ⌈To mitigate this potential hazard it is possible to provide the AtomicSwComponentType along with an accompanying symbolic name that can be used for resolving the name clash. The symbolic name is provided by means of the attribute symbol of the meta-class SymbolProps owned by AtomicSwComponentType in the role symbolProps.⌋*()*

Please note that more information about the symbolic name provided by means of the attribute symbol of the meta-class SymbolProps owned by AtomicSwComponentType in the role symbolProps can be found in Figure 3.6.

For more detailed information about how SymbolProps can be used to mitigate name clashes occurring during the integration of software-components on an AUTOSAR ECU, please refer to [4].

[TPS_SWCT_01000] Usage of attribute symbol of the symbolProps ⌈In particular, the RTE generator shall take over the value of the attribute symbol of the symbolProps owned by a given AtomicSwComponentType. If and only if symbolProps is not defined the RTE generator shall take the shortName of the AtomicSwComponentType. For the generation of symbols for RunnableEntitys [TPS_SWCT_01001] shall be observed.⌋*(RS_SWCT_00230)*

[TPS_SWCT_01001] Prefix symbols generated for the RunnableEntity ⌈If and only if the attribute symbol of a symbolProps owned by an AtomicSwComponentType exists, its value shall also be taken for prefixing the symbols generated for the RunnableEntitys owned by the AtomicSwComponentType.⌋*(RS_SWCT_00230)*

Note: if symbolProps is not defined the behavior of the RTE generator is fully backwards compatible, i.e. existing implementations of RunnableEntitys do not have to be touched in order to conform with this version of the AUTOSAR standard.

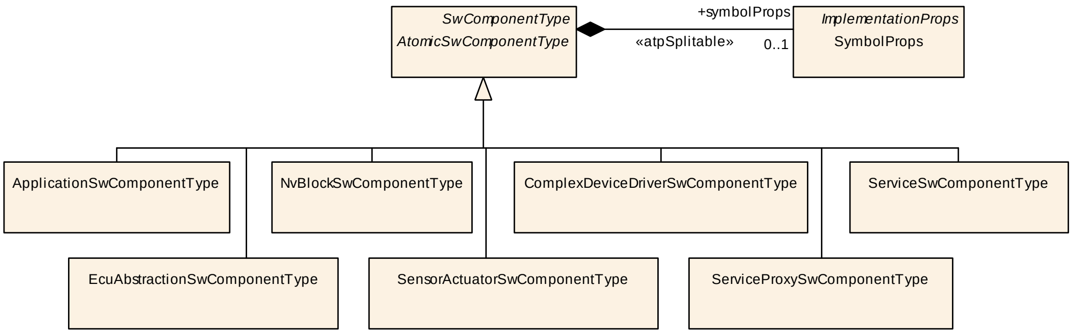


Figure 3.6: Overview of AtomicSwComponentType

This is a further measure to mitigate the risk of potential name clashes in the RTE code.

[TPS_SWCT_01635] Naming conventions may support the effectiveness of SymbolProps ⌈Of course, there is a residual risk that even in the presence of SymbolProps name clashes may occur.

Therefore, the definition of naming conventions may facilitate the avoidance of name clashes to the further degree.⌋*(RS_SWCT_00230)*

3.3 Composition

3.3.1 Overview

[TPS_SWCT_01032] CompositionSwComponentType ⌈The purpose of an AUTOSAR CompositionSwComponentType is to allow the encapsulation of specific functionality by aggregating existing software-components.⌋*(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

[TPS_SWCT_01033] Nested definition of CompositionSwComponentTypes ⌈Since a CompositionSwComponentType is also a SwComponentType, it again may be aggregated in further CompositionSwComponentTypes.⌋*(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

This recursive relation is formally expressed in Figure 3.7.

It is important to understand that while compositions allow for (sub-) system abstraction, they are solely an *architectural element for the implementation of model scalability*. They simply group existing software-components and thereby take away complexity when viewing or designing logical software architecture.

Therefore, the definition of CompositionSwComponentTypes has no effect on how software-components interact with the Virtual Functional Bus (VFB). CompositionSwComponentTypes do not add any new functionality to what is already provided by the software-components they aggregate.

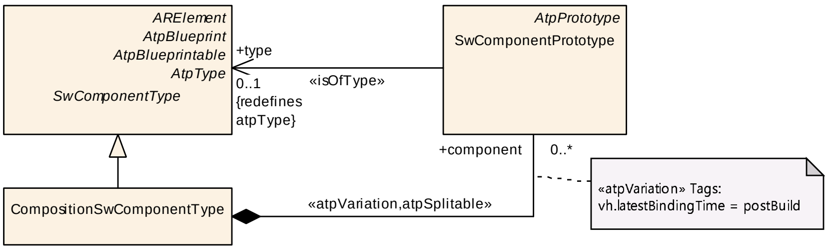


Figure 3.7: The recursive relation of software-components and compositions

[TPS_SWCT_01034] `CompositionSwComponentType`s do not have any binary footprint ⌈As the main consequence, `CompositionSwComponentType`s do not have any binary footprint in the ECU software.⌋*(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

3.3.2 SwComponentPrototype

[TPS_SWCT_01035] `CompositionSwComponentType` aggregates `SwComponentPrototype`s ⌈In terms of the AUTOSAR meta-model, a composition of software-components realized by the meta-class `CompositionSwComponentType` aggregates `SwComponentPrototype`s which in turn are typed by a `SwComponentType`.⌋ *(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

Please note that a `CompositionSwComponentType` is also a `SwComponentType`.

[TPS_SWCT_01036] `SwComponentPrototype` implements a specific role ⌈Therefore, a `SwComponentPrototype` implements the usage of a `SwComponentType` in a specific *role*.⌋*(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

[TPS_SWCT_01037] arbitrary numbers of `SwComponentPrototype`s can be created ⌈In general, arbitrary numbers of `SwComponentPrototype`s that refer to specific `SwComponentType`s can be created.⌋*(RS_SWCT_00190, RS_SWCT_02000, RS_SWCT_02020, RS_SWCT_03000)*

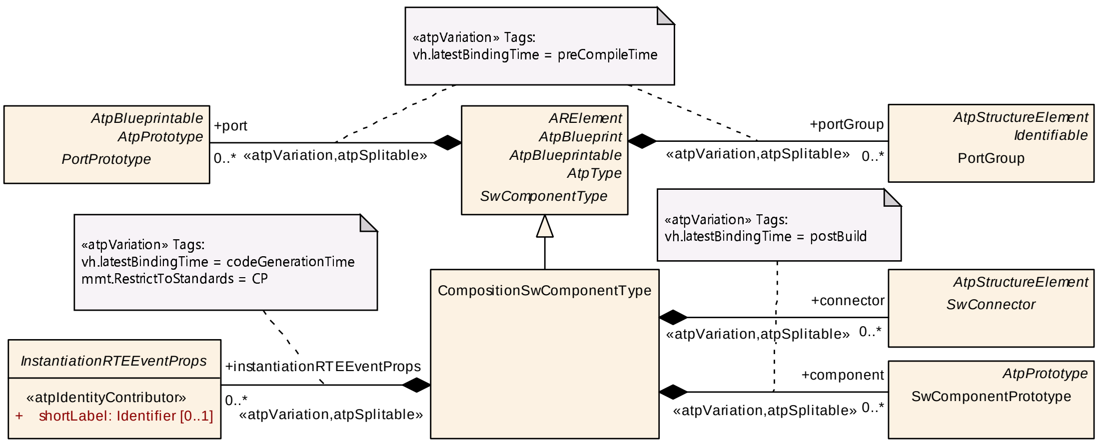


Figure 3.8: Composition and the meta-classes aggregated

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by Sw ComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means, hierarchical structures of software-components can be created. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
component	SwComponent Prototype	*	aggr	The instantiated components that are part of this composition. The aggregation of SwComponentPrototype is subject to variability with the purpose to support the conditional existence of a SwComponentPrototype. Please be aware: if the conditional existence of Sw ComponentPrototypes is resolved post-build the deselected SwComponentPrototypes are still contained in the ECUs build but the instances are inactive in that they are not scheduled by the RTE. The aggregation is marked as atpSplitable in order to allow the addition of service components to the ECU extract during the ECU integration. The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=component.shortName, component.variation Point.shortLabel vh.latestBindingTime=postBuild
connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySw Connectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variation Point.shortLabel vh.latestBindingTime=postBuild
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortCom Spec. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping

△

Class	CompositionSwComponentType			
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementors mainly because the RTE generator is not concerned about the CompositionSwComponent Types. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
instantiation RTEEventProps	InstantiationRTEEvent Props	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationRTEEventProps.shortLabel, instantiationRTEEventProps.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table 3.10: CompositionSwComponentType

Class	SwComponentPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.component			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType

Table 3.11: SwComponentPrototype

Example: a SwComponentPrototype “LeftDoorControl” fulfills the role of implementing the SwComponentType “DoorControl” for the left door of a vehicle while the SwComponentPrototype “RightDoorControl” fulfills the role of the SwComponentType “DoorControl” for the right door.

[TPS_SWCT_01080] Delegation ports ⌈Note that being a SwComponentType, a CompositionSwComponentType also exposes PortPrototypes to the outside world. However, the PortPrototypes are only delegated and do not play the same role as PortPrototypes attached to AtomicSwComponentTypes.⌋*(RS_SWCT_03130)*

[TPS_SWCT_01081] Implications of being a delegation port ⌈Being a PortPrototype attached to a CompositionSwComponentType has the following implications:

- The delegation has to follow the rules for basic compatibility.

- By creating PortPrototypes on the surface of a specific CompositionSwComponentType it is explicitly decided whether the contents of an "inner" port contained in the CompositionSwComponentType is exposed to the outside world.

⌋*(RS_SWCT_03130)*

Please note that the rules for compatibility are described in chapter 6.

Please note further that the semantics of the delegation of PortPrototypes are similar to encapsulation mechanisms like public and private members in object-oriented programming languages.

One implication of the concept of CompositionSwComponentType is that the application software of an entire vehicle eventually is represented by one CompositionSwComponentType. This so-called top-level composition has a special role in the context of the AUTOSAR System Template [10].

However, please note that a top-level composition might have (unconnected) PortPrototypes in order to allow for reuse as part of another system.

[constr_1035] Recursive definition of CompositionSwComponentType ⌈The recursive definition of a CompositionSwComponentType that eventually contains a SwComponentPrototype typed by the same CompositionSwComponentType shall not be feasible **at any time in the workflow**.⌋*()*

3.3.3 Connectors

[TPS_SWCT_01079] SwConnector ⌈Note that CompositionSwComponentType also aggregates the abstract meta-class SwConnector for connecting the contained SwComponentPrototypes among each other.⌋*(RS_SWCT_03130)*

More information can be found in Figure 3.8.

CompositionSwComponentTypes contain three kinds of SwConnectors:

- **[TPS_SWCT_01082] AssemblySwConnector** ⌈AssemblySwConnectors interconnect PortPrototypes of SwComponentPrototypes that are part of the CompositionSwComponentType.⌋*(RS_SWCT_03130)*
- **[TPS_SWCT_01083] DelegationSwConnector** ⌈DelegationSwConnectors connect from "inner" PortPrototypes to delegated "outer" PortPrototypes.⌋*(RS_SWCT_03130)*

  [TPS_SWCT_01084] Outer PortPrototype is referenced by multiple DelegationSwConnectors ⌈In the case that an outer PortPrototype is referenced by multiple DelegationSwConnectors the semantic is the multiplication of the AssemblySwConnectors referencing the outer PortPrototypes.⌋ *(RS_SWCT_03130)*

- PassThroughSwConnector, see [TPS_SWCT_01507].

[constr_1086] SwConnector between two specific PortPrototypes ⌈Each pair of PortPrototypes can only be connected by one and only one SwConnector **at the time when the RTE is generated**.⌋*()*

In other words, it is not supported to create two different SwConnectors that connect the same pair of PortPrototypes.

[TPS_SWCT_01638] Existence of SwConnector between two PRPortPrototypes ⌈[constr_1086] applies also in the case that two PRPortPrototypes are connected with each other. In particular, the roles

- AssemblySwConnector.requester
- AssemblySwConnector.provider
- PassThroughSwConnector.providedOuterPort
- PassThroughSwConnector.requiredOuterPort

do **not** establish a direction in this case.⌋*()*

For clarification, [TPS_SWCT_01638] means that the SwConnector represents the ability for bi-directional communication between the two PRPortPrototypes.

[constr_1087] AssemblySwConnector inside CompositionSwComponentType ⌈An AssemblySwConnector can only connect PortPrototypes of SwComponentPrototypes that are owned by the same CompositionSwComponentType **at any time in the workflow**.⌋*()*

[constr_1088] DelegationSwConnector inside CompositionSwComponentType ⌈A DelegationSwConnector can only connect a PortPrototype of a SwComponentPrototype that is owned by the same CompositionSwComponentType that also owns the connected delegation PortPrototype **at any time in the workflow**.⌋*()*

In the context of attaching a DelegationSwConnector to an inner PRPortPrototype there is some ambiguity to be considered. In particular, from the formal point of view it would be feasible to use **either** a PPortInCompositionInstanceRef **or** a RPortInCompositionInstanceRef.

The ability to use one or the other meta-class arbitrarily is considered confusing. Therefore, [TPS_SWCT_01515] has been defined to remove the unnecessary degree of freedom.

[TPS_SWCT_01515] PPortInCompositionInstanceRef shall be used for attaching DelegationSwConnector to an inner PRPortPrototype ⌈For the implementation of the attachment of a DelegationSwConnector to an inner PRPortPrototype the meta-class PPortInCompositionInstanceRef shall be used.⌋*(RS_SWCT_03130)*

[constr_1100] Unconnected RPortPrototype typed by a DataInterface ⌈For any element in an unconnected RPortPrototype typed by a DataInterface, there shall be a requiredComSpec that defines an initValue **at the time when the RTE is generated.**⌋*()*

Class	***SwConnector*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	The base class for connectors between ports. Connectors have to be identifiable to allow references from the system constraint template.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	AssemblySwConnector, DelegationSwConnector, PassThroughSwConnector			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.connector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
mapping	PortInterfaceMapping	0..1	ref	Reference to a PortInterfaceMapping specifying the mapping of unequal named PortInterface elements of the two different PortInterfaces typing the two PortPrototypes which are referenced by the ConnectorPrototype.

Table 3.12: SwConnector

One specific use case for the application of SwConnectors is exemplified by the figures 3.9 and 3.11. A specific CompositionSwComponentType exists in two variants where one (more complex) variant foresees the existence of a SwComponentPrototype inside the CompositionSwComponentType (depicted by 3.9) and the other (because it is implementing a simpler semantics) does not need the SwComponentPrototype.

Class	**AssemblySwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	AssemblySwConnectors are exclusively used to connect SwComponentPrototypes in the context of a CompositionSwComponentType.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.connector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
provider	AbstractProvidedPortPrototype	0..1	iref	Instance of providing port. **InstanceRef implemented by:**PPortInCompositionInstanceRef
requester	AbstractRequiredPortPrototype	0..1	iref	Instance of requiring port. **InstanceRef implemented by:**RPortInCompositionInstanceRef

Table 3.13: AssemblySwConnector

Class	**DelegationSwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A delegation connector delegates one inner PortPrototype (a port of a component that is used inside the composition) to a outer PortPrototype of compatible type that belongs directly to the composition (a port that is owned by the composition).			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.connector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
innerPort	PortPrototype	0..1	iref	The port that belongs to the ComponentPrototype in the composition **Tags:**xml.typeElement=true **InstanceRef implemented by:**PortInCompositionTypeInstanceRef
outerPort	PortPrototype	0..1	ref	The port that is located on the outside of the CompositionType

Table 3.14: DelegationSwConnector

[constr_1860] Multiplicity of DelegationSwConnector.innerPort ⌈For each DelegationSwConnector, the reference DelegationSwConnector.innerPort shall exist **at the time when the creation of the CompositionSwComponentType is finished**.⌋ *()*

[constr_1861] Multiplicity of DelegationSwConnector.outerPort ⌈For each DelegationSwConnector, the reference DelegationSwConnector.outerPort shall exist **at the time when the creation of the CompositionSwComponentType is finished**.⌋ *()*

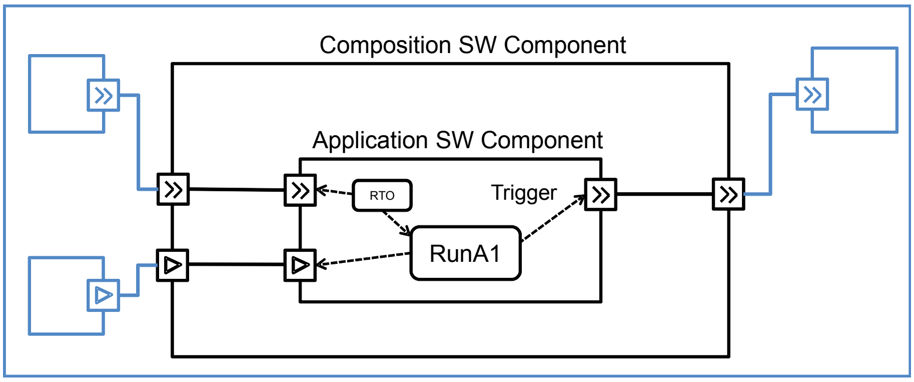


Figure 3.9: Use case for PassThroughSwConnector (I)

Without the ability to define a PassThroughSwConnector the second variant could only be implemented by defining a dummy SwComponentPrototype inside the CompositionSwComponentType. However, the dummy SwComponentPrototype would need to define RunnableEntitys that are created for the sole purpose of being able to shove the data from (e.g. for sender-receiver communication) RPortPrototypes to PPortPrototypes.

This would not only be cumbersome it would also obviously require additional resources (memory and code) at run-time. Plus, the existence of addition RunnableEntitys also unnecessarily increases the propagation delay of information flowing around inside the ECU.

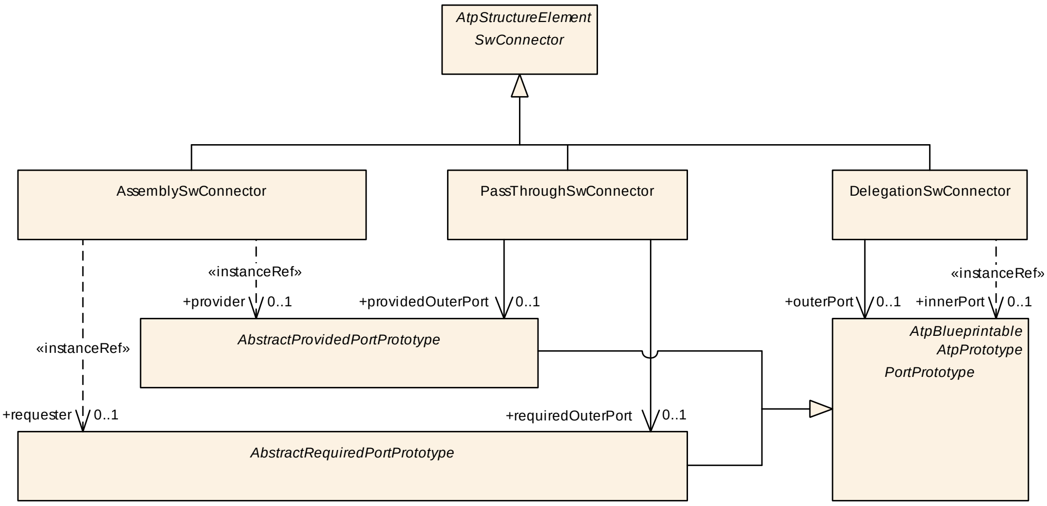


Figure 3.10: Connectors

[TPS_SWCT_01507] The role of PassThroughSwConnector ⌈PassThroughSwConnector can be taken to connect PortPrototypes owned by the same CompositionSwComponentType. In other words, PassThroughSwConnector creates a bypass inside a CompositionSwComponentType from the requiredOuterPort to the providedOuterPort (or vice versa) without involving SwComponentPrototypes.⌋*(RS_SWCT_03130)*

[constr_1252] Creation of a loop involving a PassThroughSwConnector is not allowed ⌈**at any time in the workflow**, a PassThroughSwConnector is not allowed if the required outer PortPrototype is directly or indirectly connected to the provided outer PortPrototype without the placement of a SwComponentPrototype typed by an AtomicSwComponentType in the chain of SwConnectors.⌋*()*

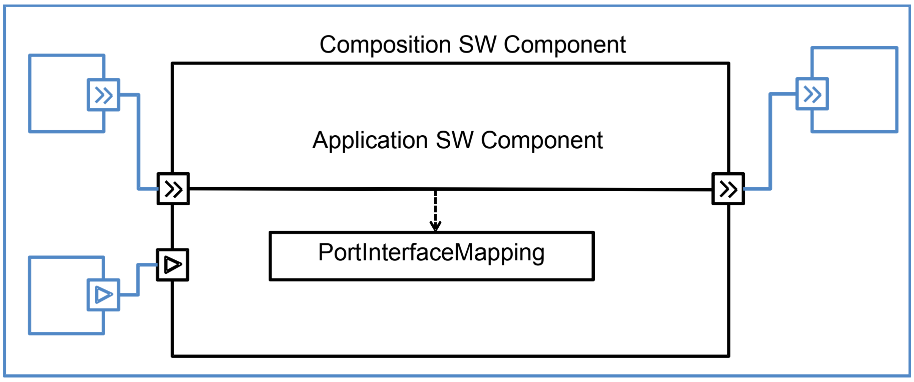


Figure 3.11: Use case for PassThroughSwConnector (II)

In other words, according to [constr_1252] it is not allowed to create a “infinite loop” by means of a PassThroughSwConnector and at least one AssemblySwConnector that connects the requiredOuterPort to the providedOuterPort.

Class	PassThroughSwConnector			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	This kind of SwConnector can be used inside a CompositionSwComponentType to connect two delegation PortPrototypes.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *SwConnector*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.connector			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
providedOuterPort	AbstractProvidedPortPrototype	0..1	ref	This represents the provided outer delegation PortPrototype of the PassThroughSwConnector.
requiredOuterPort	AbstractRequiredPortPrototype	0..1	ref	This represents the required outer delegation PortPrototype of the PassThroughSwConnector.

Table 3.15: PassThroughSwConnector

[constr_1862] Multiplicity of PassThroughSwConnector.requiredOuterPort ⌈For each PassThroughSwConnector, the reference PassThroughSwConnector.requiredOuterPort shall exist **at the time when the creation of the CompositionSwComponentType is finished**.⌋*()*

[constr_1863] Multiplicity of PassThroughSwConnector.providedOuterPort ⌈For each PassThroughSwConnector, the reference PassThroughSwConnector.providedOuterPort shall exist **at the time when the creation of the CompositionSwComponentType is finished**.⌋*()*

[TPS_SWCT_01843] Value of PassThroughSwConnector.category ⌈Meta-class PassThroughSwConnector can be used in different contexts:

- The PassThroughSwConnector is designed to implement VFB communication and therefore is considered in the generation of the RTE. In this case the attribute PassThroughSwConnector.category should either not exist or should be set to the value VFB.
- The PassThroughSwConnector is used to support the early design of a signal/service translation. At some point, the early design is replaced by actual ApplicationSwComponentTypes that finally implement the intended communication path sketched by the early design on the VFB.

  For more details about designing a signal/service translation by means of a PassThroughSwConnector, please refer to the TPS System Template [10]. To indicate the usage of the PassThroughSwConnector for the early design purpose, the value of the attribute PassThroughSwConnector.category should be set to S2S_DESIGN.

⌋*(RS_SWCT_03130)*

Please note that, on the AUTOSAR adaptive platform, the PassThroughSwConnector is also used for the design of a signal/service translation in a way that is very similar to the approach described in the TPS System Template [10]. More details can be found in the TPS Manifest Specification [14].

3.3.4 Instantiation-specific RTEEvents

[TPS_SWCT_02507] Instantiation-specific RTEEvents ⌈It is possible to specify instantiation specific properties of an RTEEvent by applying InstantiationRTEEventProps in the role instantiationRTEEventProps.

This allows to use the same ApplicationSwComponentType in different timing scenarios. Even if the scheduling is an issue of the SwcInternalBehavior, the instance specific definition of timing needs to be specified on the level of a CompositionSwComponentType.⌋*(RS_SWCT_03046, RS_SWCT_03270)*

As an example for [TPS_SWCT_02507], please consider a software-component that implements a closed-loop control algorithm.

This software-component can potentially be deployed to “slow” and “fast” control scenarios. As the actual time-base of the control algorithm is derived from the scheduling implemented in the RTE it obviously facilitates the overall design if the timing can be defined on “instance” level.

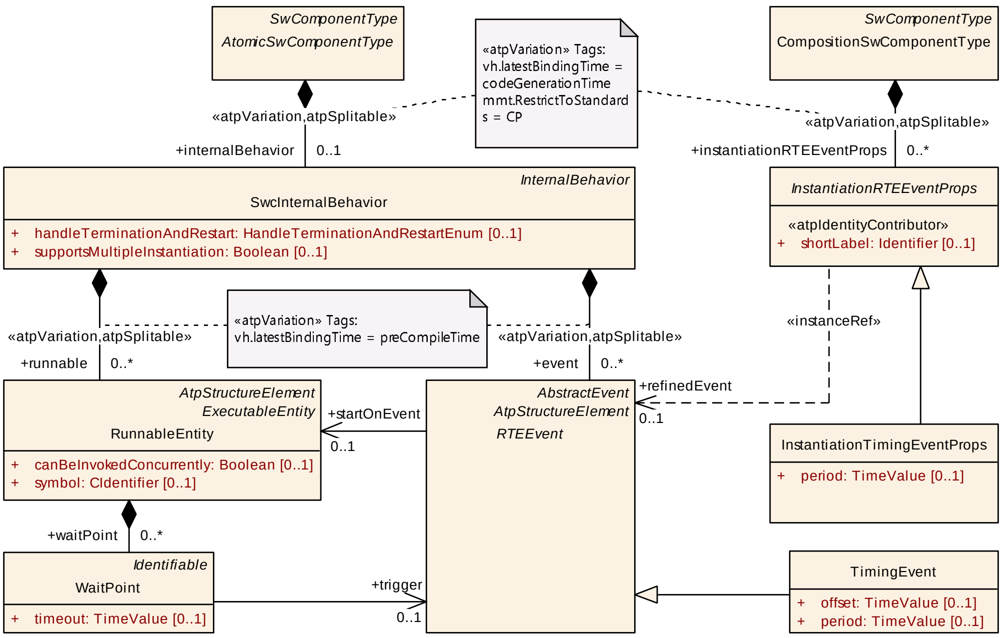


Figure 3.12: Instantiation specific Properties of RTEEvents

[constr_1233] InstantiationTimingEventProps shall only reference TimingEvent ⌈**at any time in the workflow**, an InstantiationTimingEventProps shall only reference TimingEvent in the role refinedEvent. A reference to other kinds of RTEEvents is not supported.⌋*()*

[constr_1864] Multiplicity of InstantiationRTEEventProps.refinedEvent
⌈For each InstantiationRTEEventProps, the instance-reference InstantiationRTEEventProps.refinedEvent shall exist **at the time when the RTE is generated**.⌋*()*

Please note that the attribute shortLabel only contributes to model semantics if the ability to split the definition of the aggregation CompositionSwComponentType.instantiationRTEEventProps over several physical files is actually utilized[1].

More explanation about the ability to split models over physical files can be found in [11].

Class	***InstantiationRTEEventProps*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	This meta-class represents the ability to refine the properties of RTEEvents for particular instances of a software component.			
Base	*ARObject*			
Subclasses	InstantiationTimingEventProps			
Aggregated by	CompositionSwComponentType.instantiationRTEEventProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
refinedEvent	RTEEvent	0..1	iref	This instance ref denotes the Timing Event for which the period shall be refined on an instance level. **InstanceRef implemented by:**InstanceEventInCompositionInstanceRef
shortLabel	Identifier	0..1	attr	The main purpose of the shortLabel is to contribute to the splitkey of aggregations that are <<atpSplitable>>. **Stereotypes:** atpIdentityContributor

Table 3.16: InstantiationRTEEventProps

3.4 Port Interface

[TPS_SWCT_01025] The role of PortPrototypes in the AUTOSAR architecture
⌈A PortPrototype mainly contributes the functionality of being a *connection point* to the AUTOSAR concept.

The details, i.e. with respect to what kind of information is actually transported between two PortPrototypes is defined by the PortInterface.⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_SWCT_02030, RS_SWCT_03010)*

[TPS_SWCT_01026] The role of PortInterfaces in the AUTOSAR architecture ⌈PortInterfaces are used to support a design-by-contract work-flow, i.e. a PortInterface provides means to formally verify structural and dynamic compatibility between software-components.⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_SWCT_02030, RS_SWCT_03010)*

[1] In which case the shortLabel serves as a part of the splitkey

In other words: PortInterfaces (see Figure 3.14) represent a pivotal point in the AUTOSAR concept.

Please note that a PortInterface creates a name space for the information contained. This allows for defining the details of a specific PortInterface without having to care for possible side effects on other PortInterfaces. Again, this property of the AUTOSAR concept directly supports re-usability.

[TPS_SWCT_01027] Different flavors of PortInterfaces ⌈Within the AUTOSAR concept, different flavors of PortInterfaces are defined:

- SenderReceiverInterface
- NvDataInterface
- ParameterInterface
- ModeSwitchInterface
- ClientServerInterface
- TriggerInterface

⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_SWCT_02030)*

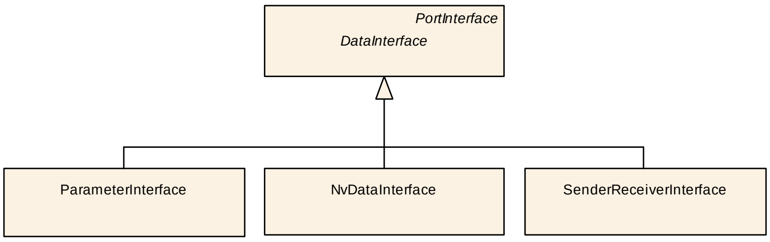


Figure 3.13: DataInterface as an abstract base class

[TPS_SWCT_01069] DataInterface is defined as abstract base class ⌈Please note that the conceptual relationship of SenderReceiverInterface, NvDataInterface, and ParameterInterface is expressed by the definition of the abstract base class DataInterface.⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_SWCT_03010)*

Please find more details about the specialization of the PortInterface concept in chapter 4.2.3 and 4.2.2.

Class	***PortInterface*** (abstract)
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	Abstract base class for an interface that is either provided or required by a port of a software component.
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*
Subclasses	ClientServerInterface, *DataInterface*, ModeSwitchInterface, TriggerInterface
Aggregated by	ARPackage.element

▽

△

Class	***PortInterface*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isService	Boolean	0..1	attr	This flag is set if the PortInterface is to be used for communication between an • ApplicationSwComponentType or • ServiceProxySwComponentType or • SensorActuatorSwComponentType or • ComplexDeviceDriverSwComponentType • ServiceSwComponentType • EcuAbstractionSwComponentType and a ServiceSwComponentType (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime
serviceKind	ServiceProviderEnum	0..1	attr	This attribute provides further details about the nature of the applied service.

Table 3.17: PortInterface

[TPS_SWCT_01070] `PortInterface` acts as a *type* for a `PortPrototype` ⌈From an abstract point of view, a `PortInterface` acts as a *type* for a `PortPrototype`. This means in particular that several `PortPrototype`s can be typed by the same `PortInterface`.⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_-SWCT_03010)*

Class	***DataInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	The purpose of this meta-class is to act as an abstract base class for subclasses that share the semantics of being concerned about data (as opposed to e.g. operations).			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	NvDataInterface, ParameterInterface, SenderReceiverInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 3.18: DataInterface

Of course, this aspect facilitates the creation of valid connections between software-components dramatically. By using a specific `PortInterface` for typing particular `PortPrototype`s the latter are eligible for being connected to each other by definition.

However, the creation of a valid connection does not need to be based on the usage of identical `PortInterface`s. It is also possible to use different, but *compatible* `PortInterface`s. The details about compatibility of `PortInterface`s are described in chapter 6.

[constr_1036] Connect kinds of PortInterfaces ⌈It shall not be possible to connect PortPrototypes typed by PortInterfaces of different kinds **at the time when the RTE is generated**.

Subclasses of DataInterface make an exception to this rule and can be used for creating connections to each other.⌋ *()*

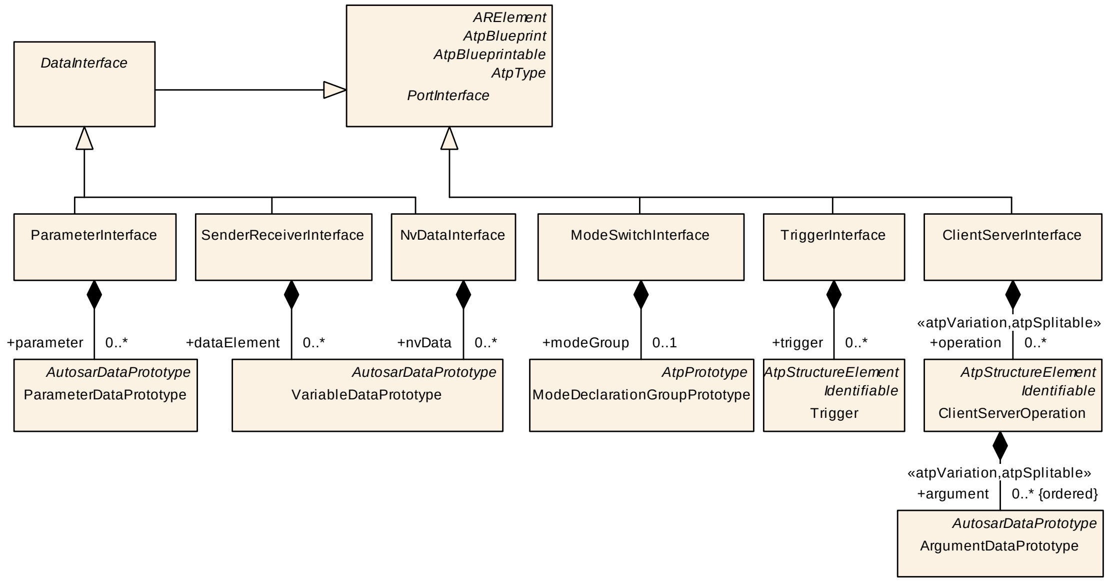


Figure 3.14: PortInterfaces in the AUTOSAR meta-model

For clarification, a connection between a PortPrototype typed by a SenderReceiverInterface and a PortPrototype typed by a ClientServerInterface shall not be possible. However, the creation of a connection between a PortPrototype typed by a SenderReceiverInterface and a PortPrototype typed by a ParameterInterface is supported.

[constr_1137] Applicability of ParameterInterface ⌈A PPortPrototype typed by a ParameterInterface can **only** be owned by a ParameterSwComponentType or a CompositionSwComponentType **at any time in the workflow**.⌋ *()*

Please note that PortInterfaces also play an important role in the context of defining so-called AUTOSAR services. In particular, by means of the attribute isService a PortInterface can define whether it is supposed to be used in the context of an AUTOSAR service and in addition to this it may define (by means of the attribute serviceKind) what kind of service is intended.

The creation of an AssemblySwConnector between PortPrototypes where the respective PortInterfaces have set attribute isService to true puts requirements on the nature of the enclosing SwComponentTypes.

For example, it does not make sense to establish a service communication between two SwComponentPrototypes that each are typed by ApplicationSwComponentTypes. For service communication, at least one of the participating SwComponentPrototypes has be to be typed by a ServiceSwComponentType.

[constr_10067] Creation of AssemblySwConnector for service communication ⌈If an AssemblySwConnector is created between two PortPrototypes and the affected PortInterfaces set the attribute isService to the value true, then at least one of the SwComponentPrototypes shall be typed by a ServiceSwComponentType. This constraint shall be imposed **at the time when the RTE is generated**.⌋*()*

The information contained in serviceKind can be used in various ways. The primary intent is to distinguish between the usage of standardized AUTOSAR services from the usage of a vendor-specific service. This information may have an impact on the development- and build process of software-components that use the PortInterface.

In addition, it is also possible to use the information contained in serviceKind for filtering the presentation of an AUTOSAR model in an AUTOSAR authoring tool and e.g. display the nature of the service PortPrototypes independently of the content of the corresponding PortInterface.

[TPS_SWCT_01003] Inconsistencies regarding the value of serviceKind and the actual implementation of the PortInterface ⌈In case of inconsistencies between the value of serviceKind and the actual implementation of the PortInterface the implementation of the PortInterface wins over the value of attribute PortInterface.serviceKind (which, for the intended purpose shall be considered an annotation rather than a semantically binding information).⌋*(RS_SWCT_00030)*

[TPS_SWCT_01004] Specific default value if serviceKind is not defined ⌈if the attribute serviceKind is not defined in the context of a specific PortInterface the default value anyStandardized shall be assumed.⌋*(RS_SWCT_00030)*

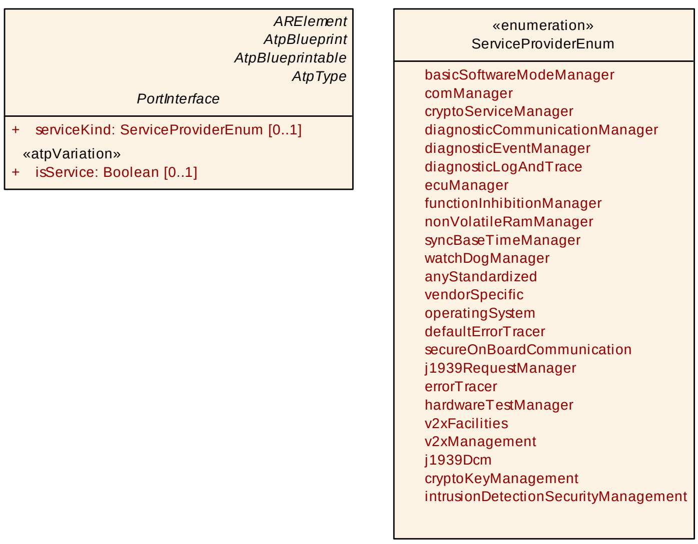


Figure 3.15: PortInterfaces and AUTOSAR services

[constr_1174] **PortInterfaces used in the context of CompositionSwComponentTypes cannot refer to AUTOSAR services** ⌈CompositionSwComponentTypes shall not own PortPrototypes typed by PortInterfaces where the attribute isService is set to true **at any time in the workflow**.⌋()

Enumeration	**ServiceProviderEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This represents a list of possible service providers
Aggregated by	*PortInterface*.serviceKind
Literal	***Description***
anyStandardized	This value means that the specific nature is either unknown or it is not important for the given purpose. This is also the default value for any attribute of type ServiceProviderEnum **Tags:**atp.EnumerationLiteralIndex=0
basicSoftwareMode Manager	The service relates to the Basic Software Mode Manager (BswM) **Tags:**atp.EnumerationLiteralIndex=1
comManager	The service relates to the COM Manager (ComM). **Tags:**atp.EnumerationLiteralIndex=2
cryptoKey Management	The service relates to the Key Manager (KeyM). **Tags:**atp.EnumerationLiteralIndex=23
cryptoService Manager	The service relates to the Crypto Service Manager (CsM). **Tags:**atp.EnumerationLiteralIndex=3
defaultErrorTracer	The service relates to the Default Error Tracer (DET) **Tags:**atp.EnumerationLiteralIndex=4
diagnostic Communication Manager	The service relates to the Diagnostic Communication Manager (DCM). **Tags:**atp.EnumerationLiteralIndex=6
diagnosticEvent Manager	The service relates to the Diagnostic Event Manager (DEM). **Tags:**atp.EnumerationLiteralIndex=7
diagnosticLogAnd Trace	The service relates to the Diagnostic Log and Trace (DLT). **Tags:**atp.EnumerationLiteralIndex=8
ecuManager	The service relates to the ECU Manager (EcuM). **Tags:**atp.EnumerationLiteralIndex=9
errorTracer	This service relates to the error tracer. **Tags:**atp.EnumerationLiteralIndex=18
functionInhibition Manager	The service relates to the Function Inhibition Manager (FIM). **Tags:**atp.EnumerationLiteralIndex=10
hardwareTest Manager	This service relates to the hardware test manager. **Tags:**atp.EnumerationLiteralIndex=19
intrusionDetection Security Management	The service relates to the intrusion detection security management (IdsM). **Tags:**atp.EnumerationLiteralIndex=24
j1939Dcm	This service relates to the J1939 Dcm. **Tags:**atp.EnumerationLiteralIndex=22
j1939Request Manager	The service relates to the J1939Rm. **Tags:**atp.EnumerationLiteralIndex=11
nonVolatileRam Manager	The service relates to the Non-Volatile RAM Manager (NvM). **Tags:**atp.EnumerationLiteralIndex=12

▽

△

Enumeration	**ServiceProviderEnum**
operatingSystem	The service relates to the Operating System (OS). **Tags:**atp.EnumerationLiteralIndex=13
secureOnBoard Communication	The service relates to the SecOc module. **Tags:**atp.EnumerationLiteralIndex=14
syncBaseTime Manager	The service relates to the Sync Time Base Manager (StbM). **Tags:**atp.EnumerationLiteralIndex=15
v2xFacilities	This service relates to the Vehicle to X facilities. **Tags:**atp.EnumerationLiteralIndex=20
v2xManagement	This service relates to the Vehicle to X management. **Tags:**atp.EnumerationLiteralIndex=21
vendorSpecific	This value denotes a vendor-specific service. **Tags:**atp.EnumerationLiteralIndex=16
watchDogManager	The service relates to the Watchdog Manager (WdgM). **Tags:**atp.EnumerationLiteralIndex=17

Table 3.19: ServiceProviderEnum

Please find more details about the relation of `PortInterface`s to AUTOSAR services in chapter 11.

4 Details: Software Components, Ports, and Interfaces

4.1 Introduction

The specification of the Virtual Functional Bus (VFB) [3] explains the main communication paradigms for communication among software-components: *client/server* for operation-based communication, and *sender/receiver* for data-based communication.

The nature of the two communication paradigms is quite different, and so is the modeling of SenderReceiverInterfaces and ClientServerInterfaces and their related meta-classes.

[TPS_SWCT_01516] PortInterface describes the static structure of information interchange ⌈PortInterfaces are limited to the description of the static structure of the exchanged information; the dynamic attributes relevant for communication are attached to PortPrototypes.⌋*(RS_SWCT_00010, RS_SWCT_00080, RS_SWCT_00110, RS_SWCT_02030, RS_SWCT_03010)*

Please note that the dynamic attributes relevant for communication are described in chapter 4.5.

4.2 Port Interface Details

4.2.1 Introduction

The usage of value encodings (for more information please refer to section 5.2.6) is limited within the context of PortInterfaces.

[constr_10383] Supported value encodings for SwBaseType in the context of PortInterfaces where attribute isService is set to false ⌈The supported value encodings for the usage within a PortInterface where attribute isService is set to false are:

- 2C: Two's complement
- IEEE754: floating-point numbers
- ISO-8859-1: single-byte coded character
- ISO-8859-2: single-byte coded character
- WINDOWS-1252: single-byte coded character
- UTF-8: UCS Transformation Format 8
- UTF-16: Character encoding for Unicode code points based on 16 bit code units [15]

- UCS-2: Universal Character Set 2
- NONE: Unsigned Integer
- BOOLEAN: This represents an integer to be interpreted as boolean.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Please note that the limitation formulated in [constr_10383] is **only applicable for non-service communication**. Service-level communication can utilize further encodings, as documented in [TPS_SWCT_01845].

[constr_1046] Applicability of [TPS_SWCT_01845] ⌈[TPS_SWCT_01845] applies **at the time when the contract phase generation is executed** **only** if the value of the attribute isService is set to false.⌋()

[constr_1295] PortInterfaces and category DATA_REFERENCE ⌈A DataPrototype defined in the context of a PortInterface used by an ApplicationSwComponentType or SensorActuatorSwComponentType that is (after potential indirections via TYPE_REFERENCE are resolved) either typed by or mapped to an ImplementationDataType of category DATA_REFERENCE shall only be used if either the provider or the requester of the information represents a ServiceSwComponentType, a ComplexDeviceDriverSwComponentType, a ParameterSwComponentType, or an NvBlockSwComponentType, or the EcuAbstractionSwComponentType.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

Note: [constr_1295] corresponds to [SWS_Rte_07670].

4.2.2 Sender Receiver Communication

4.2.2.1 Sender Receiver Interface

[TPS_SWCT_01114] SenderReceiverInterface ⌈SenderReceiverInterfaces allow for the specification of the typically asynchronous communication pattern where a sender provides data that is required by one or more receivers.

While the actual communication takes place via the respective PortPrototypes, a SenderReceiverInterface allows for formally describing what kind of information is sent and received.⌋*(RS_SWCT_02030)*

Class	**SenderReceiverInterface**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:**atp.recommendedPackage=PortInterfaces

▽

△

Class	SenderReceiverInterface			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataElement	VariableDataPrototype	*	aggr	The data elements of this SenderReceiverInterface.
invalidation Policy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement
metaDataItem Set	MetaDataItemSet	*	aggr	This aggregation defines fixed sets of meta-data items associated with dataElements of the enclosing Sender ReceiverInterface

Table 4.1: SenderReceiverInterface

A `SenderReceiverInterface` focuses on the description of information items represented by `VariableDataPrototype`s (see section 5.3).

A `VariableDataPrototype` aggregated in the role of `dataElement` represents an atomic[1] piece of information transmitted among `PortPrototype`s typed by a `SenderReceiverInterface`.

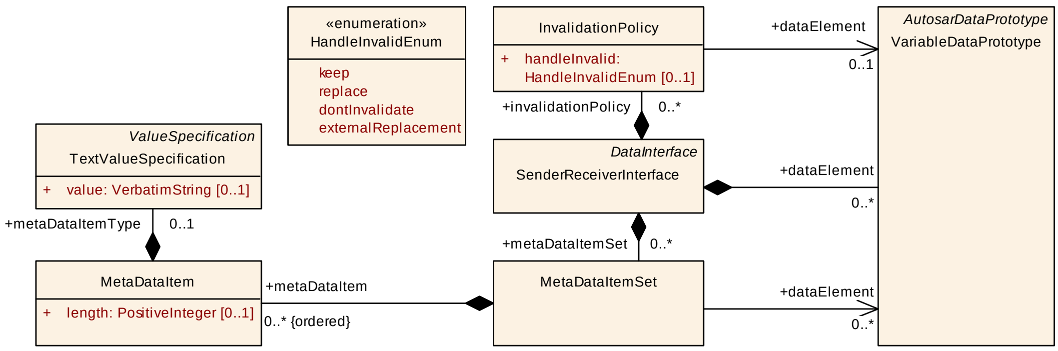


Figure 4.1: `dataElement`s of a `SenderReceiverInterface`

Note that a `SenderReceiverInterface` provides a name space for the definition of `VariableDataPrototype`s.

In terms of the AUTOSAR meta-model this aspect is indicated by the inheritance relation to `DataPrototype` (which in turn inherits from `Identifiable`). Please find more information on the creation of name spaces in [11].

[TPS_SWCT_01116] **`swImplPolicy`** ⌈The `swImplPolicy` indicates the way how a `VariableDataPrototype` shall be processed at the receiver's side. If set to `queued` the semantics is that the corresponding `VariableDataPrototype` needs to be added to a *queue* (or in other words: a FIFO data structure) from which it is later consumed by the actual receiver software-component.⌋*(RS_SWCT_02030)*

Please note that the `swImplPolicy` is described in section 5.4.

[1]Note that the term "atomic" does not have any implication on the implementation on a concrete computing platform

[constr_1200] Queued communication is not applicable for dataElements owned by PRPortPrototype ⌈The swImplPolicy shall not be set to queued for any dataElement owned by a PRPortPrototype **at any time in the workflow**.⌋*()*

[TPS_SWCT_01176] last-is-best semantics for sender-receiver communication ⌈If swImplPolicy is set to any other valid value of SwImplPolicyEnum then *last is best* semantics applies.⌋*()*

Please note that the definition of VariableDataPrototype may come very close to the reader's idea of a *signal*. However, different kinds of signals have a specific meaning in the AUTOSAR concept, especially in the context of the AUTOSAR System Template [10].

4.2.2.2 Communication Patterns for Sender Receiver Communication

[TPS_SWCT_01117] Communication patterns for sender-receiver communication ⌈PortPrototypes typed by a SenderReceiverInterface may be connected to establish a 1:n (i.e. one sender, multiple receivers) communication relationship. It is also possible to establish an n:1 (i.e. many senders, one receiver) communication pattern.⌋*(RS_SWCT_02030)*

[constr_1033] Communication scenarios for sender/receiver communication ⌈For sender/receiver communication, it is not allowed to create a communication scenario where n sender are connected to m receivers where m and n are **both** greater than 1 **at any time in the workflow**.⌋*()*

Factually, [constr_1033] is not applicable to a scenario where several PRPortPrototypes are connected by a chain of AssemblySwConnectors or PassThroughSwConnectors.

[constr_1202] Supported connections by AssemblySwConnector between PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
⌈

	RPortPrototype	PPortPrototype	PRPortPrototype
RPortPrototype	No	Yes	Yes
PPortPrototype	Yes	No	Yes
PRPortPrototype	Yes	Yes	Yes

This rule shall be imposed **at any time in the workflow**.

⌋*()*

[constr_1203] Supported connections by DelegationSwConnector between PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
⌈

innerPort	outerPort		
	RPortPrototype	PPortPrototype	PRPortPrototype
RPortPrototype	Yes	No	Yes
PPortPrototype	No	Yes	Yes
PRPortPrototype	Yes	Yes	Yes

This rule shall be imposed **at any time in the workflow**.

⌋*()*

4.2.2.3 Invalidation Policy

[TPS_SWCT_01115] invalidationPolicy ⌈An invalidationPolicy specifies whether the sending component can actively invalidate a particular dataElement and which strategy of handling the reception of invalidValue on the receiver side shall be implemented.⌋*(RS_SWCT_02030)*

Further information about the related concept of an invalidValue is provided in chapter 5.4.2

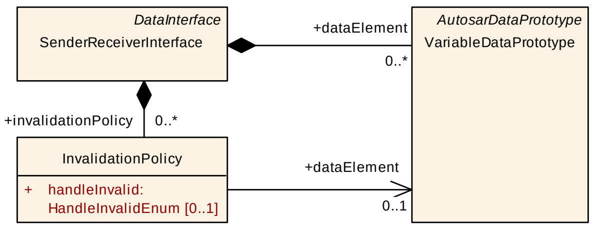


Figure 4.2: Modeling of invalidation policy of a SenderReceiverInterface

[constr_1865] Existence of InvalidationPolicy.dataElement ⌈For each InvalidationPolicy, the reference InvalidationPolicy.dataElement shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_10118] Structural consistency of the modeling of InvalidationPolicy ⌈A dataElement referenced by an InvalidationPolicy shall be owned by the SenderReceiverInterface that also owns the InvalidationPolicy **at any time in the workflow**.⌋*()*

Technically, even in the presence of [constr_10118], it would still be possible to refer to the same SenderReceiverInterface.dataElement from different InvalidationPolicys.

Therefore, a further constraint applies: for each dataElement owned by a SenderReceiverInterface, at most a single InvalidationPolicy shall be defined.

[constr_10119] SenderReceiverInterface.dataElement shall be referenced by at most one InvalidationPolicy ⌈Any SenderReceiverInterface.dataElement shall be referenced by at most one InvalidationPolicy in the role InvalidationPolicy.dataElement **at any time in the workflow.**⌋*()*

Class	**InvalidationPolicy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies whether the component can actively invalidate a particular dataElement. If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate.			
Base	*ARObject*			
Aggregated by	SenderReceiverInterface.invalidationPolicy			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dataElement	VariableDataPrototype	0..1	ref	Reference to the dataElement for which the Invalidation Policy applies.
handleInvalid	HandleInvalidEnum	0..1	attr	This attribute controls how invalidation is applied to the dataElement.

Table 4.2: InvalidationPolicy

Enumeration	**HandleInvalidEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Strategies of handling the reception of invalidValue.
Aggregated by	InvalidationPolicy.handleInvalid, ISignalPort.handleInvalid
Literal	*Description*
dontInvalidate	Invalidation is switched off. **Tags:**atp.EnumerationLiteralIndex=0
external Replacement	Replace a received invalidValue. The replacement value is sourced from the aggregation in the role replaceWith. **Tags:**atp.EnumerationLiteralIndex=1
keep	The application software is supposed to handle signal invalidation on RTE API level either by Data ReceiveErrorEvent or check of error code on read access. **Tags:**atp.EnumerationLiteralIndex=2
replace	Replace a received invalidValue. The replacement value is specified by the initValue. **Tags:**atp.EnumerationLiteralIndex=3

Table 4.3: HandleInvalidEnum

4.2.2.4 Meta-data on the Application Software Level

There are cases where information available on different levels in the AUTOSAR basic software stack need to be made available as meta-data on the application layer in order to make the overall software function properly.

One example could be a software-component that is involved with communication using the J1939 protocol.

In such a case, the semantics of the information transmitted is strongly bound to the source address and the sender needs to be able to set the source address individually.

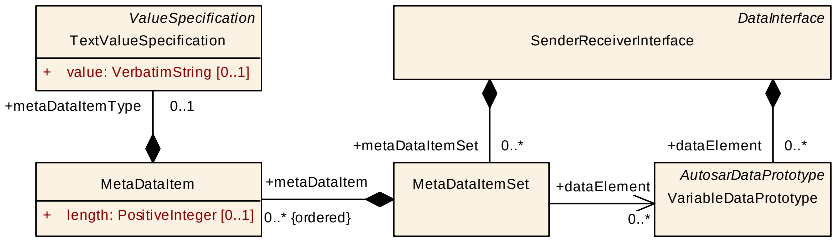


Figure 4.3: Modeling of meta-data of a SenderReceiverInterface

[TPS_SWCT_01801] Support for Meta-Data ⌈Meta-data on the application software level can only be made available in the context of a SenderReceiverInterface. No other kind of PortInterface supports the definition of meta-data.⌋*(RS_SWCT_-02030)*

[TPS_SWCT_01802] Definition of meta-data in the context of a SenderReceiverInterface ⌈The definition of meta-data in the context of a SenderReceiverInterface involves two aspects:

- The available meta-data are defined by means of an ordered aggregation of meta-class MetaDataItem at MetaDataItemSet that in turn is aggregated in the role SenderReceiverInterface.metaDataItemSet.
- The involvement of dataElements with meta-data is specified by means of the reference MetaDataItemSet.dataElement. In other words, the dataElements that are referenced by a MetaDataItemSet in the role dataElement are involved with meta-data handling.

⌋*()*

Class	**MetaDataItem**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents a single meta-data item.			
Base	*ARObject*			
Aggregated by	MetaDataItemSet.metaDataItem			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
length	PositiveInteger	0..1	attr	This attribute determines the length of the MetaDataItem at run-time.
metaDataItem Type	TextValueSpecification	0..1	aggr	This aggregation contributes the specification of the concrete meta-data item type.

Table 4.4: MetaDataItem

Class	**MetaDataItemSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the ability to define a set of meta-data items to be used in SenderReceiver Interfaces.			
Base	*ARObject*			
Aggregated by	SenderReceiverInterface.metaDataItemSet			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	MetaDataItemSet			
dataElement	VariableDataPrototype	*	ref	This reference identifies the dataElement for which the ordered list of meta-data items is defined.
metaDataItem (ordered)	MetaDataItem	*	aggr	This aggregation represents the ordered definition of meta-data items.

Table 4.5: MetaDataItemSet

[constr_1726] Ordering of MetaDataItemSet.metaDataItem ⌈The ordering of the elements of MetaDataItemSet.metaDataItem shall be **descending** with respect to the value of MetaDataItem.length, such that the MetaDataItem with the largest value of attribute length is located in the first position and the MetaDataItem with the smallest value of attribute length is located in the last position.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Although the ordering is significant by way of the meta-model, it is still true that for the creation of the ordered collection it is possible to put MetaDataItems with the same value of attribute length with arbitrary order into the collection.

In an existing collection, however, the ordering is semantically relevant even for elements with same value of attribute length.

[TPS_SWCT_01804] Standardized values of attribute MetaDataItem.metaDataItemType.value ⌈The following values of attribute MetaDataItem.metaDataItemType.value are standardized by AUTOSAR:

- PRIORITY_8
- SOURCE_ADDRESS_16
- TARGET_ADDRESS_16
- ADDRESS_EXTENSION_8
- SOCKET_CONNECTION_ID_16
- LIN_NAD_8
- CAN_ID_32
- ETHERNET_MAC_32

⌋*()*

[constr_1866] Existence of MetaDataItem.length ⌈For each MetaDataItem, attribute length shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1867] Existence of MetaDataItem.metaDataItemType ⌈For each MetaDataItem, attribute metaDataItemType shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1868] Existence of MetaDataItemSet.dataElement ⌈For each MetaDataItemSet that aggregates at least one metaDataItem, at least one reference to a dataElement shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_10120] Structural consistency of the modeling of MetaDataItemSet ⌈A dataElement referenced by an MetaDataItemSet in the role dataElement shall be owned by the SenderReceiverInterface that also owns the MetaDataItemSet **at any time in the workflow**.⌋*()*

Technically, even in the presence of [constr_10120], it would still be possible to refer to the same SenderReceiverInterface.dataElement from different MetaDataItemSets.

However, this is not allowed because the aggregation of MetaDataItem at MetaDataItemSet is qualified as ordered, and therefore it would be very difficult to semantically merge the content of different MetaDataItemSets to one set of meta data items in the implementation.

Therefore, a further constraint applies: for each dataElement owned by a SenderReceiverInterface, at most a single MetaDataItemSet shall be defined.

[constr_10121] SenderReceiverInterface.dataElement shall be referenced by at most one MetaDataItemSet ⌈Any SenderReceiverInterface.dataElement shall be referenced by at most one MetaDataItemSet in the role MetaDataItemSet.dataElement **at any time in the workflow**.⌋*()*

4.2.3 Client Server Communication

The underlying semantics of a client/server communication is that a client may initiate the execution of an operation by a server that supports the operation.

The server executes the operation and, when completed, it provides the client with the result (synchronous operation call) or else the client checks for the completion of the operation by itself (asynchronous operation call).

[constr_1037] Client shall not be connected to multiple servers ⌈A client shall not be connected to multiple servers such that an operation call would be handled by more than one server **at the time when the RTE is generated**.⌋*()*

4.2.3.1 Client Server Interface

A ClientServerInterface, to some extent, is a counterpart to the SenderReceiverInterface[2].

[2]However, different connection patterns apply, see [constr_1037]

Instead of defining pieces of information to be transferred among software-components, a ClientServerInterface defines a collection of ClientServerOperations.

Class	ClientServerInterface			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, PortInterface, Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operation	ClientServerOperation	*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=operation.shortName, operation.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table 4.6: ClientServerInterface

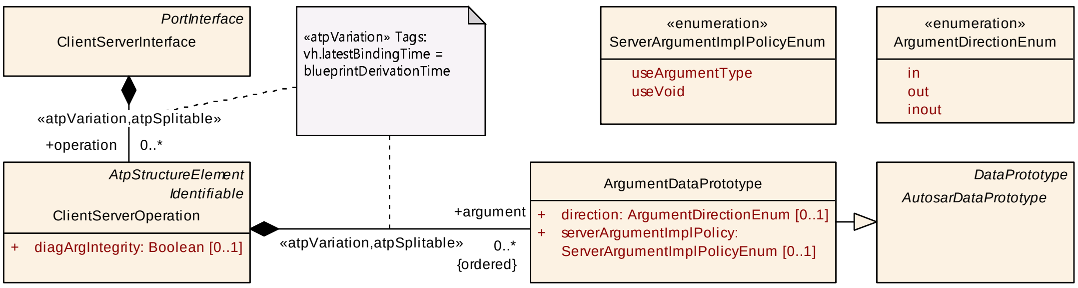


Figure 4.4: ClientServerOperations of a ClientServerInterface

[TPS_SWCT_01118] ClientServerInterface ⌈A ClientServerInterface is composed of ClientServerOperations, i.e. a ClientServerOperation cannot be reused in the context of a different ClientServerInterface.⌋*(RS_SWCT_02030)*

[TPS_SWCT_01106] ClientServerOperation ⌈A ClientServerOperation consists of 0..* ArgumentDataPrototypes. The latter may be

- passed to the operation (i.e. the direction is "in")
- passed to and returned from the operation (i.e. the direction is "inout")
- returned from the operation (i.e. the direction is "out")

The aggregation represents a variation point.⌋*(RS_SWCT_02030, RS_SWCT_03141)*

[TPS_SWCT_01844] Optional method arguments ⌈AUTOSAR does not support the existence of optional arguments within a ClientServerOperation.⌋*()*

The reason for the existence of the restriction in [TPS_SWCT_01844] on the *AUTOSAR classic platform* is that the RTE does not have an API to handle optional method arguments.

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationInterface.command, *AtpClassifier*.atpFeature, ClientServerInterface.operation, DiagnosticDataElementInterface.read, DiagnosticDataIdentifierInterface.read, DiagnosticDataIdentifierInterface.write, DiagnosticRoutineInterface.requestResult, DiagnosticRoutineInterface.start, DiagnosticRoutineInterface.stop, PhmRecoveryActionInterface.recovery, ServiceInterface.method			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=argument.shortName, argument.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
diagArgIntegrity	Boolean	0..1	attr	This attribute shall only be used in the implementation of diagnostic routines to support the case where input and output arguments are allocated in a shared buffer and might unintentionally overwrite input arguments by tentative write operations to output arguments. This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the ClientServerOperation is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments. If the attribute does not exist or is set to false the ClientServerOperation does not have to consider the usage of a shared buffer.
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table 4.7: ClientServerOperation

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ClientServerOperation.argument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
direction	ArgumentDirectionEnum	0..1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures.

Table 4.8: ArgumentDataPrototype

[TPS_SWCT_01119] Direction of ArgumentDataPrototypes ⌈To cover these cases, ArgumentDataPrototype defines an attribute direction, possible values are in (pass to operation), out (return from operation), and inout (pass to and return from operation).⌋*(RS_SWCT_02030)*

In many common programming languages (like *C*), an operation is yet another data type. This makes it for example possible to pass a reference to an operation as an argument to another operation.

This is *not* allowed in the AUTOSAR concept.

Essentially, all ArgumentDataPrototypes in a ClientServerOperation can be passed (conceptually) by value (from the client to the server and/or from the server to the client depending on the direction of the ArgumentDataPrototype).

[constr_1869] Existence of attribute ArgumentDataPrototype.direction ⌈For each ArgumentDataPrototype, attribute direction shall be defined **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01120] Client needs to provide ArgumentDataPrototypes ⌈When the client invokes an operation, it needs to provide a value for each ArgumentDataPrototype that is of direction in or inout.⌋*(RS_SWCT_02030)*

[TPS_SWCT_01121] Pass correct data type ⌈The value passed to an ArgumentDataPrototype of direction in or inout needs to be of the corresponding Datatype.⌋*(RS_SWCT_02030)*

[TPS_SWCT_01122] Synchronous call of ClientServerOperation ⌈In the case of synchronous operation call, the client expects to receive a response to the invocation of the operation.

As part of the response, it receives a value (of the correct AutosarDataType) for each ArgumentDataPrototype that is of direction out or inout.⌋*(RS_SWCT_02030)*

Enumeration	**ArgumentDirectionEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	Use cases: • Arguments in ClientServerOperation can have different directions that need to be formally indicated because they have an impact on how the function signature looks like eventually. • Arguments in BswModuleEntry already determine a function signature, but the direction is used to specify the semantics, especially of pointer arguments.
Aggregated by	ArgumentDataPrototype.direction, SwServiceArg.direction
Literal	***Description***
in	The argument value is passed to the callee. **Tags:**atp.EnumerationLiteralIndex=0
inout	The argument value is passed to the callee but also passed back from the callee to the caller. **Tags:**atp.EnumerationLiteralIndex=1
out	The argument value is passed from the callee to the caller. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.9: ArgumentDirectionEnum

Each ClientServerOperation provides a name space for its ArgumentDataPrototypes and therefore has a unique identifier which identifies the operation within the corresponding ClientServerInterface.

The ClientServerOperations have no ordering within a ClientServerInterface (there is no such thing as the “first” operation)[3].

[TPS_SWCT_01123] No default values for ArgumentDataPrototypes ⌈It is not possible to define default values for ArgumentDataPrototypes defined in the context of a ClientServerOperation. Default values might lead to complicated mappings to programming languages.⌋*()*

[TPS_SWCT_01124] Definition of ArgumentDataPrototypes within the context of a ClientServerOperation is ordered ⌈In contrast to the unordered relationship of ClientServerInterface to ClientServerOperation, the definition of ArgumentDataPrototypes within the context of a ClientServerOperation is ordered, i.e. a ClientServerOperation may have a *first* argument[4].⌋*(RS_SWCT_-02030)*

Please note that ArgumentDataPrototype inherits from AutosarDataPrototype and therefore has a reference to a concrete AutosarDataType.

The RTE Generator uses the referred AutosarDataTypes to determine the data types of the arguments depending on the value of the attribute ArgumentDataPrototype.serverArgumentImplPolicy.

Enumeration	**ServerArgumentImplPolicyEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	This defines how the argument type of the servers RunnableEntity is implemented.
Aggregated by	ArgumentDataPrototype.serverArgumentImplPolicy
Literal	***Description***
useArgumentType	The argument type of the RunnableEntity is derived from the AutosarDataType of the Argument Prototype. **Tags:**atp.EnumerationLiteralIndex=0
useVoid	The argument type of the RunnableEntity is void. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.10: ServerArgumentImplPolicyEnum

[3]In different parts of the definition of a ClientServerInterface, a “calling-order” of the ClientServerOperations might be prescribed: the client might be required to use the ClientServerOperations in a certain logical ordering.
However, this ordering has nothing to do with the order in which the ClientServerOperations are listed in the definition of a ClientServerInterface

[4]Giving the ArgumentDataPrototypes of a ClientServerOperation both an ordering and a unique identifier might seem redundant.
For example, in the operation “foo(a, b, c)”, we can refer to the “second argument” or to “the argument named b”. In many common programming languages (like C or Java), only the *ordering* is actually used by the client during the invocation of the server (the client invokes the operation as “foo(1,2,3)” not as “foo(a=1,c=3,b=2)”).
In addition, the names of the arguments represent an arbitrary choice made when implementing of the invocation. In *C*, only the data types and ordering of the arguments constitute the signature, *not* the names of the arguments.

[constr_1286] **serverArgumentImplPolicy and ArgumentDataPrototype typed by primitive data types** ⌈The value of the attribute ArgumentDataPrototype.serverArgumentImplPolicy shall **not** be set to useVoid for an ArgumentDataPrototype of direction in that is typed by an AutosarDataType that boils down to a primitive C data type (see [TPS_SWCT_01565]).

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

There is one use case for executing multiple ClientServerOperations using the exact same RunnableEntity. This use case is sketched in Figure 4.5.

In such a case, it may happen that the ClientServerOperations own ArgumentDataPrototypes that are typed by an array data type.

It may also happen that the individual ArgumentDataPrototypes that are typed by array data types refer to a compatible element data type, but define a different number of elements.

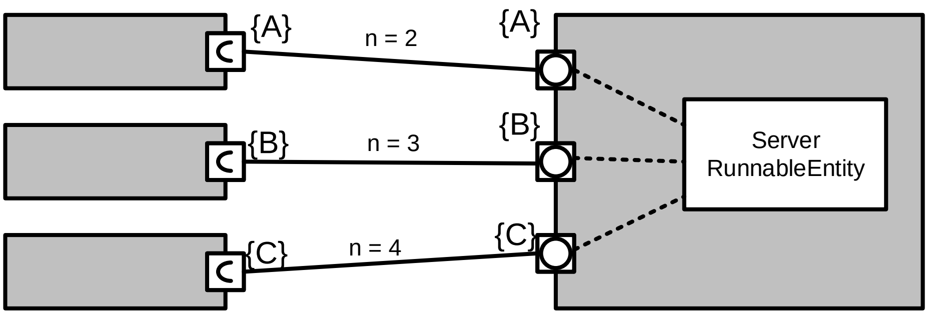


Figure 4.5: Use case for one server RunnableEntity processes calls from different PortPrototypes

Please note that the server RunnableEntity needs information about the currently used array length respectively structure size by usage of additional arguments passed by the Client. This aspect is exemplified by Figure 4.5.

It is only natural that in such a case a Variable-Size Array Data Type would be used because it comes with a built-in capability to indicate the number of elements currently stored in the array without the need to add further arguments to the signature of the RunnableEntity.

Note further that a ClientServerInterface does not define any timing information (how quickly the client expects a response of the server). It does not define how the threading works (if the client for example blocks until the response comes back from the server).

It also does not define explicitly how information is passed between an implementation of the client and the server and the underlying RTE (for example: through “pointers” or “by value”).

[constr_1204] Supported connections by AssemblySwConnector between PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface ⌈

	RPortPrototype	PPortPrototype	PRPortPrototype
RPortPrototype	No	**Yes**	**Yes**
PPortPrototype	**Yes**	No	No
PRPortPrototype	**Yes**	No	No

This rule shall be imposed at any time in the workflow.

⌋()

[constr_1205] Supported connections by DelegationSwConnector between PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface ⌈

innerPort	outerPort		
	RPortPrototype	PPortPrototype	PRPortPrototype
RPortPrototype	**Yes**	No	No
PPortPrototype	No	**Yes**	No
PRPortPrototype	No	**Yes**	No

This rule shall be imposed at any time in the workflow.

⌋()

Please note that a restriction concerning the usage of attribute ClientServerOperation.diagArgIntegrity is described in section 13.8.4.5.

4.2.3.2 Error Handling in Client/Server Communication

This section describes the handling of errors occurring either within an application software-component or during the communication across the VFB [3]. Errors that are created and consumed by basic software modules are not in the scope of this document and therefore will not be discussed.

Therefore, errors in the scope of this document are divided into two simple classes:

- infrastructure errors and
- application errors.

A software-component implementation uses RTE API methods to communicate with other software-components. During this communication certain errors can occur as a result of infrastructure faults, like a bus is not working, or an expected data value was not arriving in time.

These errors are listed in the RTE specification [2], as they are an inherent feature of the infrastructure provided by the VFB. Software-components will therefore typically not raise infrastructure errors on their own.

Instead, the AUTOSAR basic software and the RTE will determine infrastructure faults and communicate the corresponding error codes to the relevant software-components.

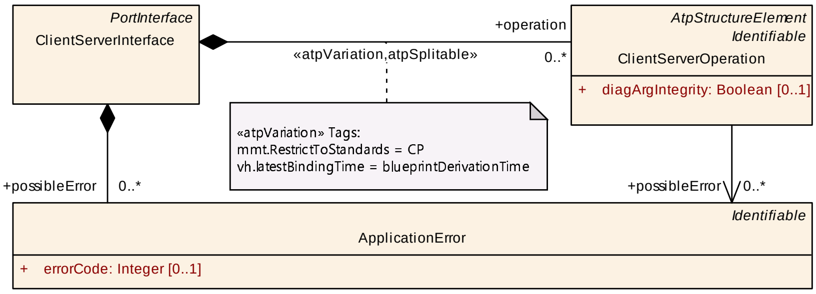


Figure 4.6: Application error meta-model

[TPS_SWCT_01491] AUTOSAR system does not need to explicitly describe infrastructure errors ⌈As the fixed set of infrastructure errors is defined as an implicit part of the VFB, a developer of an AUTOSAR system does not need to explicitly describe these.

It is assumed that these might occur at run-time and application developers should take measures to handle them.⌋*()*

Application errors, on the other hand, are specific to the functionality or information that is described in form of a PortInterface. It is not possible to define such errors up front, instead they are defined at design time of a certain PortInterface.

In principle, such ApplicationErrors could be part of all kinds of PortInterfaces.

[constr_1102] ApplicationError in the scope of one SwComponentType ⌈If a SwComponentType has PortPrototypes typed by different ClientServerInterfaces with equal shortName and ApplicationErrors defined then the following condition applies: ApplicationErrors with the same shortName shall have **identical values** of errorCodes.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1102]: the RTE generator creates symbols for the error codes in which the shortName of the ClientServerInterface and the shortName of the ApplicationError occur.

[constr_1108] Existence and value of attribute ApplicationError.errorCode ⌈Attribute ApplicationError.errorCode shall exist **at the time when the contract phase generation is executed** and its value shall not exceed the closed interval 1 .. 63.

The following exception applies: **only** if attribute possibleError is supposed to represent the error code E_OK, the value 0 shall be allowed.⌋()

By [constr_1108] it is possible to ensure that only the six least significant bits of a return value shall be used for indicating an application error.

Class	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ClientServerInterface.possibleError			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
errorCode	Integer	0..1	attr	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Table 4.11: ApplicationError

Consequently, ClientServerOperations may be associated with a number of ApplicationErrors they possibly raise. These errors are defined as part of the ClientServerInterface.

[constr_1038] Reference to ApplicationError ⌈A possibleError referenced by a ClientServerOperation shall be owned by the ClientServerInterface that also owns the ClientServerOperation **at any time in the workflow**.⌋
()

4.2.4 External Trigger Event Communication

[TPS_SWCT_01196] Semantics of an external trigger event communication ⌈The underlying semantics of an external trigger event communication is that a trigger source may initiate the execution of RunnableEntitys in the connected trigger sinks. Typically, (but not necessarily) these RunnableEntitys are executed in a sequential order.⌋*(RS_SWCT_02030)*

[TPS_SWCT_01197] TriggerInterface ⌈The TriggerInterface defines a set of Trigger to be communicated between software-components. The Trigger represents a special kind of events at which occurrence the trigger sinks shall react in a particular manner.⌋()

Class	**TriggerInterface**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	A trigger interface declares a number of triggers that can be sent by an trigger source. **Tags:**atp.recommendedPackage=PortInterfaces
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*

▽

△

Class	TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
trigger	Trigger	*	aggr	The Trigger of this trigger interface.

Table 4.12: TriggerInterface

Class	Trigger			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	A trigger which is provided (i.e. released) or required (i.e. used to activate something) in the given context.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.releasedTrigger, BswModuleDescription.requiredTrigger, ServiceInterface.trigger, TriggerInterface.trigger			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table 4.13: Trigger

Class	MultidimensionalTime			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::MultidimensionalTime			
Note	This is used to specify a multidimensional time value based on ASAM CSE codes. It is specified by a code which defined the basis of the time and a scaling factor which finally determines the time value. If for example the cseCode is 100 and the cseCodeFactor is 360, it represents 360 angular degrees. If the cseCode is 0 and the cseCodeFactor is 50 it represents 50 microseconds.			
Base	*ARObject*			
Aggregated by	AgeConstraint.maximum, AgeConstraint.minimum, AnalyzedExecutionTime.bestCaseExecutionTime, AnalyzedExecutionTime.worstCaseExecutionTime, ArbitraryEventTriggering.maximumDistance, ArbitraryEventTriggering.minimumDistance, BurstPatternEventTriggering.minimumInterArrivalTime, BurstPatternEventTriggering.patternJitter, BurstPatternEventTriggering.patternLength, BurstPatternEventTriggering.patternPeriod, ConcretePatternEventTriggering.offset, ConcretePatternEventTriggering.patternJitter, ConcretePatternEventTriggering.patternLength, ConcretePatternEventTriggering.patternPeriod, ConfidenceInterval.lowerBound, ConfidenceInterval.upperBound, ExecutionTimeConstraint.maximum, ExecutionTimeConstraint.minimum, IoHwAbstractionServerAnnotation.age, LatencyTimingConstraint.maximum, LatencyTimingConstraint.minimum, LatencyTimingConstraint.nominal, MeasuredExecutionTime.maximumExecutionTime, MeasuredExecutionTime.minimumExecutionTime, MeasuredExecutionTime.nominalExecutionTime, OffsetTimingConstraint.maximum, OffsetTimingConstraint.minimum, PeriodicEventTriggering.jitter, PeriodicEventTriggering.minimumInterArrivalTime, PeriodicEventTriggering.period, ReceiverAnnotation.signalAge, RoughEstimateOfExecutionTime.estimatedExecutionTime, SimulatedExecutionTime.maximumExecutionTime, SimulatedExecutionTime.minimumExecutionTime, SimulatedExecutionTime.nominalExecutionTime, SporadicEventTriggering.jitter, SporadicEventTriggering.maximumInterArrivalTime, SporadicEventTriggering.minimumInterArrivalTime, SporadicEventTriggering.period, SwDataDefProps.swRefreshTiming, SynchronizationTimingConstraint.tolerance, Trigger.triggerPeriod			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
cseCode	CseCodeType	0..1	attr	Specifies the time base by means of CSE codes.
cseCodeFactor	Integer	0..1	attr	The scaling factor for the time value based on the specified CSE code.

Table 4.14: MultidimensionalTime

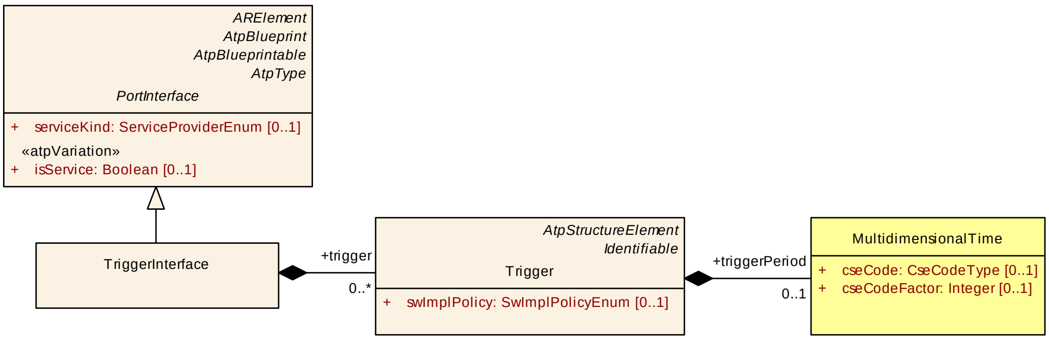


Figure 4.7: `Trigger` of a `TriggerInterface`

As illustrated in Figure 4.7, a `TriggerInterface` is composed of `Trigger`.

[TPS_SWCT_01198] Period for periodic triggering ⌈A `Trigger` can optionally define a period for periodic triggering. It is expressed via the meta-class `MultidimensionalTime` in terms of time or angle. Note that the main use case for this is to specify the properties if the trigger is coming from the Basic Software e.g. from a Complex Driver, it is not used as an input for the RTE generator.⌋*(RS_SWCT_02030)*

Apart from this, a `TriggerInterface` does not define any timing information (e.g. how quickly the source expects a reaction of the sinks). This is property of the timing information in the templates.

[constr_1104] Trigger communication shall not implement an n:1 pattern ⌈An `RPortPrototype` typed by a `TriggerInterface` shall not be connected to `PortPrototype`s typed by `TriggerInterface`s such that a given `Trigger` in the `TriggerInterface` of the `RPortPrototype` is connected to more than one compatible (see [constr_1081], [constr_1082], and [constr_1251]) `Trigger` in the `TriggerInterface`s of the connected `PortPrototype`s **`at the time when the RTE is generated`**.⌋*()*

[constr_1104] boils down to the rule that trigger communication shall not be implemented an n:1 (where n > 1) scenario.

This condition shall be observed (in a similar way as it is observed for individual `dataElement`s in a sender/receiver communication) on the basis of individual `Trigger`s rather than the connection between entire `PortPrototype`s.

Please note that the constraint applies for connections created by `AssemblySwConnector`s and `DelegationSwConnector`s in the same way. This is the reason why [constr_1104] does not get more specific about the nature of the `PortPrototype`s on opposite end of the connection to the `RPortPrototype`.

To be clear, the n:1 (where n > 1) scenario is not supported for trigger communication because there is no active use case for it.

[TPS_SWCT_01199] Queued processing of `Trigger`s ⌈It may happen that at least tentatively a `Trigger` source fires `Trigger`s faster than they can be processed on the side of the `Trigger` sink. To support this use case it is possible to process trigger event communication in a queued manner.

In this case the Triggers are added to a queue from where the foremost trigger is dequeued and processed when the processing of the current Trigger is done. Please note that the queue size is **not** subject to definition in the scope of this document. The actual queue size is defined during the process of RTE configuration.

The specification of whether a Trigger is subject to queued processing is controlled by the attribute Trigger.swImplPolicy.⌋()

[constr_1169] Allowed values for Trigger.swImplPolicy ⌈The **only** allowed values for the attribute Trigger.swImplPolicy are either STANDARD (in which case the Trigger processing does not use a queue) or QUEUED (in which case the processing of Triggers positively uses a queue). **at the time when the contract phase generation is executed.**⌋()

Please note that the value of Trigger.swImplPolicy is not the final word on the implementation of a queue for the specific Trigger. The integrator still has the power to overrule the application software developer's verdict if applicable.

For more information regarding the ability to connect different kinds of PortPrototypes typed by a TriggerInterface to each other please refer to [constr_1204] and [constr_1205].

4.2.5 Communication of Modes

4.2.5.1 Mode Switch Interface

There are two distinctive use cases for the communication of modes via ports:

1. An actual mode transition can be communicated from a mode manager component to its client components to enforce a mode switch.
2. A request for a mode transition can be communicated from any component to a mode manager.

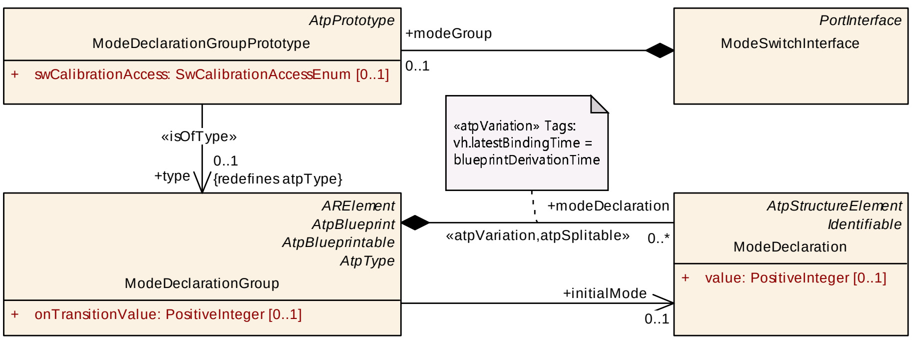


Figure 4.8: Mode Switch Interface

[TPS_SWCT_01087] Propagation of mode information ⌈For communicating a mode switch (i.e. the first use case), the Software-Component Template describes the concept of the communication of ModeDeclarationGroupPrototypes similar to the communication of VariableDataPrototypes but it uses a special type of PortInterface: the collections of ModeDeclarations that are required or provided by a SwComponentType are defined by means of ModeSwitchInterfaces used to type the PortPrototypes owned by the SwComponentType.⌋*(RS_SWCT_02030, RS_SWCT_03203)*

This aspect is depicted in Figure 4.8.

Class	**ModeSwitchInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A mode switch interface declares a ModeDeclarationGroupPrototype to be sent and received. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroup Prototype	0..1	aggr	The ModeDeclarationGroupPrototype of this mode interface.

Table 4.15: ModeSwitchInterface

Class	**ModeDeclarationGroupPrototype**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.providedModeGroup, BswModuleDescription.requiredModeGroup, FirewallStateSwitchInterface.firewallStateMachine, FunctionGroupSet.functionGroup, ModeSwitchInterface.modeGroup, Process.processStateMachine, StateManagementStateNotification.stateMachine			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	This allows for specifying whether or not the enclosing ModeDeclarationGroupPrototype can be measured at run-time.
type	ModeDeclarationGroup	0..1	tref	The "collection of ModeDeclarations" (= ModeDeclarationGroup) supported by a component **Stereotypes:** isOfType

Table 4.16: ModeDeclarationGroupPrototype

[constr_2049] Different ModeDeclarationGroups shall have different shortNames. ⌈A software component is not allowed to type multiple PortPrototypes with ModeSwitchInterfaces where the contained ModeDeclarationGroupPrototypes are referencing ModeDeclarationGroups with identical shortNames but different ModeDeclarations **at the time when the contract phase generation is executed**⌋*()*

Obviously, the rationale for [constr_2049] is to avoid conflicts in generated RTE files.

For instance:

Two ModeDeclarationGroups with identical shortName “Foo” are defined.

ModeDeclarationGroup "Foo"
contains the ModeDeclarations "X", "Y", "Z"

ModeDeclarationGroup "Foo*"
contains the ModeDeclarations "W", "X", "Y", "Z"

In this case a software component is only allowed to use either "Foo" or "Foo*"

[TPS_SWCT_01200] ModeDeclarationGroupPrototype per ModeSwitchInterface ⌈The multiplicity of the aggregation of ModeDeclarationGroupPrototype to ModeSwitchInterface is pragmatically limited to 1.⌋*(RS_SWCT_03203)*

Admittedly, there would be no technical restriction to support a 0..* multiplicity but on the other hand it does not seem as if any reasonable use case for such a scenario exists.

If somehow a SwComponentType would have to consider two or even more ModeDeclarationGroupPrototypes it is very likely that these would be part of different ModeSwitchInterfaces.

The containment of a ModeDeclarationGroupPrototype in a ModeSwitchInterface allows for explicitly defining SwConnectors which communicate between SwComponentPrototypes and to define service interfaces for communication with ServiceSwComponentTypes.

Due to the compatibility rules of PortInterfaces (see chapter 6) each SwComponentType can rely on the availability of required mode activations.

Please note that each SwComponentType can define (via their PortPrototypes and ModeSwitchInterfaces) a list of required and provided ModeDeclarationGroupPrototypes.

[TPS_SWCT_01201] CompositionSwComponentType requires and provides the modes that are required or provided by its contained SwComponentPrototypes ⌈Eventually, a CompositionSwComponentType requires and provides the modes that are required or provided by its contained SwComponentPrototypes.

The delegation of these modes from SwComponentPrototypes to the enclosing CompositionSwComponentType is explicitly described by DelegationSwConnectors.⌋*(RS_SWCT_03202, RS_SWCT_03203)*

The formal description of a software-component does not make any assumptions about the semantics of the required and provided ModeDeclarationGroupPrototypes.

It just requires and provides the ModeDeclarationGroupPrototypes by name. For more information about mode declaration refer to section 9.1.

4.2.5.2 Data Types used for Mode Communication

[TPS_SWCT_01086] Request mode change ⌈The ability to request a mode (i.e. the second use case) is modeled on the VFB via a SenderReceiverInterface and for the RTE it is like a usual communication, that means the connector can also cross ECU boundaries and the communicated dataElements have to be based on AutosarDataTypes.⌋*(RS_SWCT_03202, RS_SWCT_03203)*

However, for semantic consistency with the first use case mentioned at the beginning of section 4.2.5.1, a communicated mode request shall also be mapped to a corresponding ModeDeclarationGroup. This can be defined by the mapping class ModeRequestTypeMap, as shown in figure 4.9.

The ImplementationDataType mapped to a certain ModeDeclarationGroup can then be used in a PortInterface to represent a ModeDeclaration of the associated ModeDeclarationGroup as a numerical value.

[constr_4002] Unambiguous mapping of modes to data types ⌈Within one DataTypeMappingSet, a ModeDeclarationGroup shall not be mapped to different ImplementationDataTypes **at the time when the contract phase generation is executed**.⌋*()*

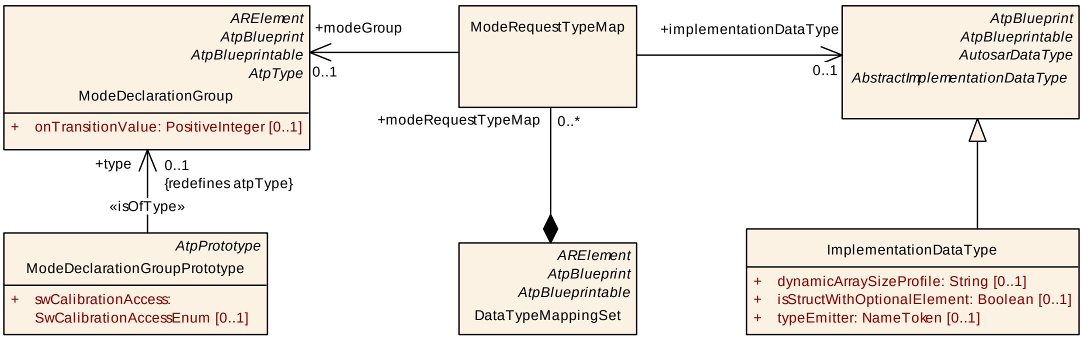


Figure 4.9: Mapping of modes to data types

Class	**ModeRequestTypeMap**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Specifies a mapping between a ModeDeclarationGroup and an ImplementationDataType. This ImplementationDataType shall be used to implement the ModeDeclarationGroup.			
Base	*ARObject*			
Aggregated by	DataTypeMappingSet.modeRequestTypeMap			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
implementation DataType	AbstractImplementation DataType	0..1	ref	This is the corresponding AbstractImplementationData Type. It shall be modeled along the idea of an "unsigned integer-like" data type.
modeGroup	ModeDeclarationGroup	0..1	ref	This is the corresponding ModeDeclarationGroup.

Table 4.17: ModeRequestTypeMap

[constr_1166] Restrictions of ModeRequestTypeMap ⌈For every ModeDeclarationGroup referenced by a ModeDeclarationGroupPrototype used in a PortPrototype typed by a ModeSwitchInterface a ModeRequestTypeMap shall exist that points to the ModeDeclarationGroup and also to an eligible ImplementationDataType.

The ModeRequestTypeMap shall be aggregated by a DataTypeMappingSet which is referenced from the SwcInternalBehavior that is owned by the ApplicationSwComponentType that also owns the PortPrototype.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1871] Existence of attribute ModeRequestTypeMap.implementationDataType ⌈For each ModeRequestTypeMap, attribute implementationDataType shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1872] Existence of attribute ModeRequestTypeMap.modeGroup ⌈For each ModeRequestTypeMap, attribute modeGroup shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1167] ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType ⌈The ImplementationDataType referenced by a ModeRequestTypeMap shall **either** be

- of category VALUE **or**
- of category TYPE_REFERENCE that in turn references an ImplementationDataType of category VALUE.

The baseType referenced by the ImplementationDataType shall have set the value of the attribute BaseTypeDirectDefinition.baseTypeEncoding to NONE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01202] ApplicationDataType defines a subset of the values used in the ModeDeclarationGroup ⌈Please note that the corresponding ApplicationDataType is defining a subset of the values used in the ModeDeclarationGroup and the used labels may differ from the names used for the ModeDeclarations.

It is in the responsibility of a system designer to maintain the data types and ModeDeclarationGroups according to the functional needs.

For example, a ModeRequester may only request a subset of the available Modes (via SenderReceiverInterface or ClientServerInterface). The ModeManager may additionally decide to indicate failure.⌋*(RS_SWCT_03203)*

For more information regarding the ability to connect different kinds of PortPrototypes typed by a ModeSwitchInterface to each other please refer to [constr_1204] and [constr_1205].

4.2.5.3 Relevance for Measurement and Calibration

Please note that by aggregating SwCalibrationAccessEnum in the role swCalibrationAccess, a ModeDeclarationGroupPrototype gains the ability to become measurable. This implies the following constraint:

[constr_1172] Allowed values of SwCalibrationAccessEnum for ModeDeclarationGroupPrototype ⌈The only allowed values of swCalibrationAccess aggregated by ModeDeclarationGroupPrototype are

- notAccessible and
- readOnly.

This rule shall be imposed **at any time in the workflow**.⌋*()*

[TPS_SWCT_01566] Define literals for an MCD system in the context of a FlatInstanceDescriptor ⌈If ModeDeclarationGroupPrototype.swCalibrationAccess is set to readOnly, a referenced FlatInstanceDescriptor.swDataDefProps may in turn refer to a CompuMethod that defines the particular literals used in the MCD system for displaying values of the measured ModeDeclarationGroupPrototypes.⌋*(RS_SWCT_03203)*

The existence of this use case is the reason for putting “AI” at the intersection of compuMethod and FlatInstanceDescriptor in [constr_1015].

Another possible scenario[5] is that a FlatInstanceDescriptor does not exist[6], but still it would be good to have the ability to define particular literals for displaying values in an MCD system.

This case can be supported by the AUTOSAR standard as well by the appearance of “S” at the intersection of compuMethod and McDataInstance in [constr_1015].

4.2.6 Parameter Communication

Of course, the “communication” of ParameterDataPrototypes as part of a ParameterInterface does not establish an actual transmission of data.

The term is used in a conceptual meaning; and the existence of something like a ParameterInterface is justified by the mere idea of unifying the exposure of calibration parameters at the surface of a software-component on the same formal level

[5]that does not necessarily have to be related to ModeDeclarationGroupPrototypes, but to the definition of literals for MCD systems in general

[6]e.g. because the affected piece of data exists in the basic software

as the exposure of other pieces of data, i.e. by means of a PortPrototype typed by a PortInterface.

[constr_1312] PortPrototypes typed by a ParameterInterface ⌈PortPrototypes typed by a ParameterInterface can either be PPortPrototypes or RPortPrototypes. The usage of PRPortPrototypes that are typed by a ParameterInterface is not supported **at any time in the workflow**.⌋*()*

4.3 PortInterface Mapping and Data Scaling

In former versions of this specification, the requirements on PortInterfaces to match each other could lead to situations where PortInterfaces that were "practically" compatible would nevertheless be rejected because of formal reasons (e.g. shortNames of dataElements do not match).

In order to also support scenarios where the developer of a CompositionSwComponentType needs to connect PortPrototypes that would match to each other but don't fulfill formal requirements the concept of "port interface mapping" has been introduced.

[TPS_SWCT_01158] Cases for PortInterfaceMapping ⌈In general, the existence of a PortInterfaceMapping is suitable in the following cases:

1. Two PortPrototypes shall be connected and the PortInterface elements are compatible except the unequal shortNames. This requires a pure logical mapping of the PortInterface elements.
2. PortInterface elements are logically equivalent but the range and resolution is differently. This requires a data conversion respectively a re-scaling of the provided data and arguments to the required data and arguments range and resolution.
3. invalidationPolicy of PortInterface elements is different. This might require the implementation of different invalidation handling strategies for the same dataElement in parallel on the same ECU.
4. Two PortPrototypes shall be connected and the PortInterface elements shall be converted using the AUTOSAR data transformer approach.

⌋*(RS_SWCT_03210)*

More information about the AUTOSAR data transformer approach can be found in section 4.3.3.

Typically, the mapping of such PortInterface is agreed once between the different component vendors and system designer in the early phase of a project.

[TPS_SWCT_01159] Mapping is described separately from the SwConnector as reusable ARElement ⌈The mapping is described separately from the SwConnector as reusable ARElement. A set of PortInterfaceMappings is grouped in a PortInterfaceMappingSet.⌋(RS_SWCT_03210)

[TPS_SWCT_01543] PortInterfaceMapping overrides all other compatibility rules ⌈The existence of a PortInterfaceMapping overrides all other compatibility rules given that the following statements are fulfilled:

- [constr_1071] applies also for the application of a PortInterfaceMapping.
- [constr_1268] applies also for the application of a PortInterfaceMapping.
- [constr_1269] applies also for the application of a PortInterfaceMapping.
- [constr_1270] applies also for the application of a PortInterfaceMapping.
- A structural difference between mapped DataPrototypes can be mitigated by means of a SubElementMapping. This includes the case that a "structure" data type is mapped to an "array" data type and vice versa. [TPS_SWCT_01195] is also applicable.

When using a PortInterfaceMapping, the developer of a software-component needs to properly understand the consequences in terms of model semantics.⌋(RS_SWCT_03210)

Please note that [TPS_SWCT_01543] does not require a tool implementation to ignore and let go unreported deviations of all other compatibility rules in the presence of a PortInterfaceMapping.

If this is considered helpful, the tool **may** still issue warnings with respect to compatibility rules defined in section 6 but this is not mandated by the AUTOSAR standard. The tool, however, **shall not** report errors in this case.

Class	**PortInterfaceMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies a set of (one or more) PortInterfaceMappings. **Tags:**atp.recommendedPackage=PortInterfaceMappingSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
portInterface Mapping	PortInterfaceMapping	*	aggr	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range). **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portInterfaceMapping.shortName, port InterfaceMapping.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime

Table 4.18: PortInterfaceMappingSet

Class	***PortInterfaceMapping*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different Port Interfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ClientServerInterfaceMapping, ModeInterfaceMapping, TriggerInterfaceMapping, VariableAndParameterInterfaceMapping			
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.19: PortInterfaceMapping

4.3.1 PortInterface Mapping

By default, the shortNames of PortInterface elements are used to identify the matching element pairs of connected PortPrototypes. In case of non-matching shortNames (this might be due to distributed development, off-the-shelves development, or reuse of software-components) it is required to explicitly specify which elements of PortInterfaces shall correlate to each other.

This definition is provided with PortInterfaceMappings.

Please note that the PortInterfaceMapping is a very powerful tool for expert users to accommodate for various differences in the PortInterfaces of two PortPrototypes that shall be connected.

In general, it is possible to define a PortInterfaceMapping for all sub-classes of SwConnectors. To make this possible, it is necessary to avoid talking about "provided" or "required" ends because this could lead to confusion.

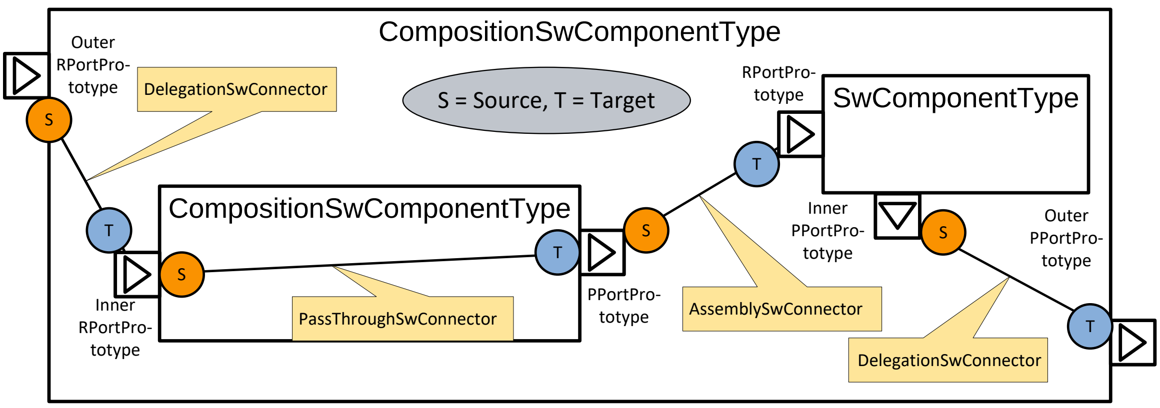


Figure 4.10: "Source" and "Target" end of a SwConnector in a sender/receiver-style scenario

Specifically, it is possible that inside a CompositionSwComponentType a PassThroughSwConnector is connected to a PPortPrototype.

From the perspective of the PassThroughSwConnector, however, the PPortPrototype might represent the "required" end of the communication relation inside the CompositionSwComponentType.

To avoid further confusion, the rules for the specification of the PortInterfaceMapping are specified **from the perspective of the SwConnector rather than the PortPrototype**.

For each SwConnector, a "source" and "target" end is postulated, reflecting the direction of communication implemented by the SwConnector. This understanding is depicted in Figure 4.10.

Please note that the "sender/receiver-style communication" (see section 4.2.2) is conceptually also applicable for other kinds of communication:

- Mode switch communication[7], see section 4.2.5
- External Trigger communication, see section 4.2.4
- NvData communication, see section 11.4.2
- Parameter "communication", see 4.2.6

Note further that the association of the "source" and "target" end of an SwConnector that represents a client/server communication (see section 4.2.3) is aligned along the "primary" interaction, i.e. the call of a ClientServerOperation, see Figure 4.11.

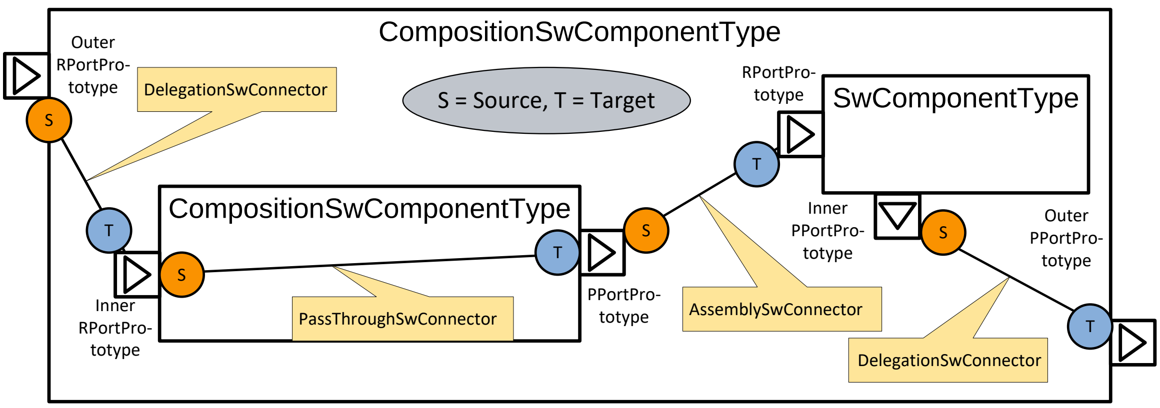


Figure 4.11: "Source" and "Target" end of a SwConnector in a client/server-style scenario

The conclusion of Figure 4.10 regarding "source" and "target" end of an SwConnector for a sender/receiver-style interaction is summarized in Table 4.20.

[7] A mode request is implemented by a sender/receiver communication anyway and does therefore not represent a dedicated communication pattern.

Sender/Receiver	Source	Target
DelegationSwConnector	Outer required PortPrototype	Inner required PortPrototype
DelegationSwConnector	Inner provided PortPrototype	Outer provided PortPrototype
AssemblySwConnector	Provided PortPrototype	Required PortPrototype
PassThroughSwConnector	Required PortPrototype	Provided PortPrototype

Table 4.20: Definition of "source" and "target" of an SwConnector for a sender/receiver-style interaction

The conclusion of Figure 4.11 regarding "source" and "target" end of an SwConnector for a client/server-style interaction is summarized in Table 4.21.

Client/Server	Source	Target
DelegationSwConnector	Inner required PortPrototype	Outer required PortPrototype
DelegationSwConnector	Outer provided PortPrototype	Inner provided PortPrototype
AssemblySwConnector	Required PortPrototype	Provided PortPrototype
PassThroughSwConnector	Provided PortPrototype	Required PortPrototype

Table 4.21: Definition of "source" and "target" of an SwConnector for a client/server-style interaction

There are some documented restrictions for the application of PortInterfaceMappings. But in general, the PortInterfaceMapping could potentially be used in a harmful way by specifying mapping approaches that have no chance to be implementable in the context of an ECU, resulting in undefined semantics in the RTE, i.e. the RTE Generator wouldn't be able to properly process such a model.

[TPS_SWCT_01099] PortInterfaceMapping ⌈Each PortInterfaceMapping describes the mapping of the PortInterface elements of exactly two PortInterfaces.⌋*(RS_SWCT_03155, RS_SWCT_03210)*

To apply the PortInterfaceMapping a SwConnector has to reference a PortInterfaceMapping.

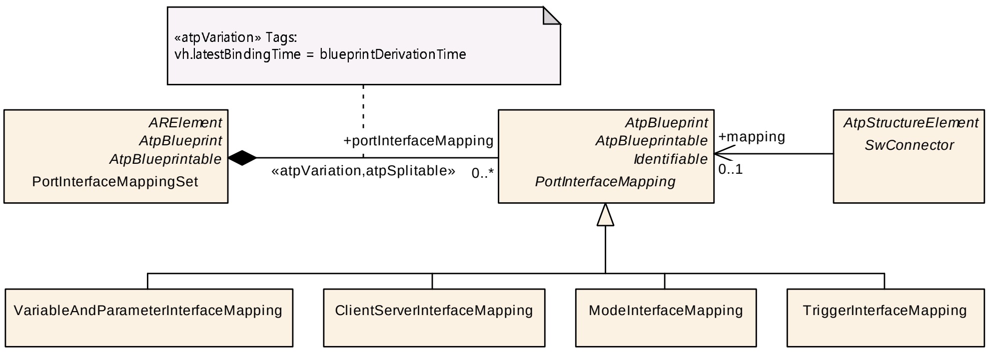


Figure 4.12: Relevant meta-classes for PortInterface element mapping

[constr_1151] Applicability of PortInterfaceMapping ⌈A PortInterfaceMapping is only applicable and valid for a SwConnector if the two PortPrototypes which are referenced by the SwConnector are typed by the same two PortInterfaces which are mapped by the PortInterfaceMapping. This rule shall be imposed **at the time when the RTE is generated**.⌋()

[TPS_SWCT_01100] Precedence of PortInterfaceMapping ⌈The mapping via PortInterfaceMapping has a higher precedence than the mapping by equal shortNames as defined in compatibility rules.

If a connector references an associated PortInterfaceMapping, this mapping shall be strictly binding with respect to the number of mapped data elements.⌋*(RS_SWCT_03155, RS_SWCT_03210)*

Please note that the compatibility rules are described in chapter 6.

[TPS_SWCT_01101] Unmapped elements of PortInterfaces ⌈Unmapped PortInterface elements will not be connected by the referencing SwConnector.⌋*(RS_SWCT_03155, RS_SWCT_03210)*

[constr_1583] PortInterfaceMapping for DataPrototype typed by Compound Primitive Data Type ⌈There is one very limited use case to apply PortInterfaceMapping for a DataPrototype typed by a Compound Primitive Data Type: adjustment of the shortName of the DataPrototype. Everything else is **not supported**. This rule shall be imposed **at any time in the workflow**.⌋()

4.3.1.1 Mapping of Sender Receiver Interface, Parameter Interface and Non Volatile Data Interface Elements

[TPS_SWCT_01102] VariableAndParameterInterfaceMapping ⌈The VariableAndParameterInterfaceMapping defines the correlation of VariableDataPrototypes and ParameterDataPrototypes defined in the context of DataInterfaces, i.e. SenderReceiverInterface, NvDataInterface, or ParameterInterface.⌋*(RS_SWCT_03155, RS_SWCT_03210, RS_SWCT_03170)*

[constr_1159] Consistency of VariableAndParameterInterfaceMapping with respect to the referenced DataInterfaces ⌈Within one VariableAndParameterInterfaceMapping all firstDataPrototypes shall belong to one and only one DataInterface and all secondDataPrototypes shall belong to one other and only one other DataInterface **at any time in the workflow**.⌋()

Please note that the relation of the “source” and “target” end of a SwConnector to the roles firstDataPrototype and secondDataPrototype is not defined.

[TPS_SWCT_01103] Mapping between different kinds of PortInterfaces ⌈Thereby it is possible to describe the mapping between different kinds of PortInterfaces for instance a ParameterInterface and SenderReceiverInterface.⌋*(RS_SWCT_03155, RS_SWCT_03210, RS_SWCT_03170)*

[TPS_SWCT_01104] Possible mappings are restricted by the swImplPolicy ⌈Nevertheless, the possible mappings of VariableDataPrototypes and ParameterDataPrototypes are restricted by the swImplPolicy attribute.⌋*(RS_SWCT_03155, RS_SWCT_03210, RS_SWCT_03170)*

For more explanation of [TPS_SWCT_01104], please refer to [constr_1071].

[constr_1039] Relevance of swImplPolicy ⌈It is not possible to define a mapping between an element where the swImplPolicy is set to queued and another element where the swImplPolicy is set differently. This rule shall be imposed **at any time in the workflow**.⌋*()*

This is required to fulfill the compatibility rules defined in [constr_1071].

[constr_1635] Relevance of attribute isOptional ⌈If a SubElementMapping is defined for the elements of a structured data type then the attribute isOptional[8] shall either not exist for the firstElement and secondElement or it shall have the identical value for the firstElement and secondElement. This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

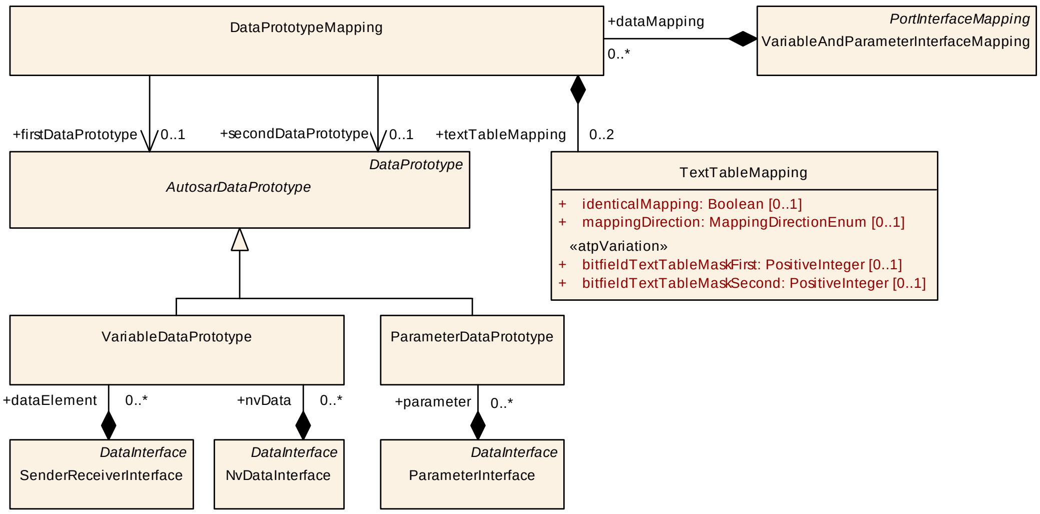


Figure 4.13: Mapping of Sender Receiver Interface, Parameter Interface and Non Volatile Data Interface elements

[constr_1040] Conversion of SenderReceiverInterfaces ⌈The conversion of elements of SenderReceiverInterfaces is possible if one of the following conditions applies **at the time when the RTE is generated**:

- The AutosarDataTypes of the referred DataPrototypes are compatible.
- A conversion of the data is available.
- A DataPrototypeMapping.firstToSecondDataTransformation is defined.

⌋*()*

[8]this is valid for both ApplicationRecordElement and ImplementationDataTypeElement

The compatibility of AutosarDataTypes is described in section 6.2. A description of the conversion of data can be found in section 4.3.2.

Class	**VariableAndParameterInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of VariableDataPrototypes or ParameterDataPrototypes in context of two different SenderReceiverInterfaces, NvDataInterfaces or ParameterInterfaces.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *PortInterfaceMapping*, *Referrable*			
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataMapping	DataPrototypeMapping	*	aggr	Defines the mapping of two particular VariableData Prototypes or ParameterDataPrototypes with unequal names and/or unequal semantic (resolution or range) in context of two different SenderReceiverInterfaces, Nv DataInterfaces or ParameterInterfaces

Table 4.22: VariableAndParameterInterfaceMapping

Class	**DataPrototypeMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two particular VariableDataPrototypes, ParameterDataPrototypes or Argument DataPrototypes with non-equal shortNames, non-equal structure (specific condition is described by [constr_1187]), and/or non-equal semantic (resolution or range) in context of two different Sender ReceiverInterface, NvDataInterface or ParameterInterface or Operations. If the semantic is unequal, the following rules apply: The textTableMapping is only applicable if the referred DataPrototypes are typed by AutosarDataType referring to CompuMethods of category TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE or BITFIELD_TEXTTABLE. In the case that the DataPrototypes are typed by AutosarDataType either referring to CompuMethods of category LINEAR, IDENTICAL or referring to no CompuMethod (which is similar as IDENTICAL) the linear conversion factor is calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys of the referred CompuMethods.			
Base	*ARObject*			
Aggregated by	ClientServerOperationMapping.argumentMapping, VariableAndParameterInterfaceMapping.dataMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
firstData Prototype	AutosarDataPrototype	0..1	ref	First to be mapped DataPrototype in context of a Sender ReceiverInterface, NvDataInterface, ParameterInterface or Operation.
firstToSecond Data Transformation	DataTransformation	0..1	ref	This reference defines the need to execute the Data Transformation <Mip>_<transformerId> functions of the transformation chain when communicating from the Data PrototypeMapping.firstDataPrototype to the Data PrototypeMapping.secondDataPrototype. This reference also specifies the reverse Data Transformation <Mip>_Inv_<transformerId> functions of the transformation chain (i.e. from the DataPrototype Mapping.secondDataPrototype to the DataPrototype Mapping.firstDataPrototype) if the referenced Data Transformation is symmetric, i.e. attribute Data Transformation.dataTransformationKind is set to symmetric.
secondData Prototype	AutosarDataPrototype	0..1	ref	Second to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, Parameter Interface or Operation.

▽

△

Class	DataPrototypeMapping			
secondToFirst Data Transformation	DataTransformation	0..1	ref	This defines the need to execute the reverse Data Transformation <Mip>_Inv_<transformerId> functions of the transformation chain when communicating from the DataPrototypeMapping.secondDataPrototype to the Data PrototypeMapping.firstDataPrototype.
subElement Mapping	SubElementMapping	*	aggr	This represents the owned SubelementMapping.
textTable Mapping	TextTableMapping	0..2	aggr	Applied TextTableMapping(s)

Table 4.23: DataPrototypeMapping

[constr_1873] Existence of `DataPrototypeMapping.firstDataPrototype` ⌈For each `DataPrototypeMapping`, the reference in the role `firstDataPrototype` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1874] Existence of `DataPrototypeMapping.secondDataPrototype` ⌈For each `DataPrototypeMapping`, the reference in the role `secondDataPrototype` shall exist **`at the time when the RTE is generated`**.⌋*()*

4.3.1.2 Mapping of Client Server Interface Elements

[TPS_SWCT_01105] `ClientServerInterfaceMapping` ⌈The `ClientServerInterfaceMapping` defines the correlation of `ClientServerOperation`s defined in the context of two `ClientServerInterface`s.⌋*(RS_SWCT_03155, RS_SWCT_03210)*

[constr_1237] Scope of mapped `ClientServerOperation`s in the context of a `ClientServerOperationMapping` ⌈All `ClientServerOperation`s referenced by a `ClientServerOperationMapping` in the role `firstOperation` shall belong to exactly one `ClientServerInterface`.

All `ClientServerOperation`s referenced by a `ClientServerOperationMapping` in the role `secondOperation` shall belong to exactly one other `ClientServerInterface`.

This rule shall be imposed **`at any time in the workflow`**.⌋*()*

[constr_1238] Scope of mapped `ApplicationError`s in the context of a `ClientServerOperationMapping` ⌈All `ApplicationError`s referenced by a `ClientServerApplicationErrorMapping` in the role `firstApplicationError` shall belong to exactly one `ClientServerInterface`.

All `ApplicationError`s referenced by a `ClientServerApplicationErrorMapping` in the role `secondApplicationError` shall belong to exactly one other `ClientServerInterface`.

This rule shall be imposed **`at any time in the workflow`**.⌋*()*

[constr_1041] Conversion of ClientServerInterfaces ⌈Either the AutosarDataTypes of the referred ArgumentDataPrototypes are compatible or a conversion of the data is available **at the time when the RTE is generated**.⌋()

The compatibility of AutosarDataTypes is described in section 6.2. A description of the conversion of data can be found in section 4.3.2.

[constr_1240] Consistency of ArgumentDataPrototypes within the context of a ClientServerOperationMapping ⌈Unless a ClientServerOperationMapping.firstToSecondDataTransformation exists, for each argument owned by

- a ClientServerOperationMapping.firstOperation and
- ClientServerOperationMapping.secondOperation,

a reference in the role

- ClientServerOperationMapping.argumentMapping.firstDataPrototype or
- ClientServerOperationMapping.argumentMapping.secondDataPrototype

shall exist **at any time in the workflow**, originated by one of the ClientServerOperationMapping.argumentMappings owned by the mentioned ClientServerOperationMapping.⌋()

[constr_1268] ArgumentDataPrototype.direction shall be preserved in a ClientServerOperationMapping ⌈Within the context of a ClientServerOperationMapping, the value of the argument ArgumentDataPrototype.direction of two mapped ArgumentDataPrototype shall be identical **at any time in the workflow**.⌋()

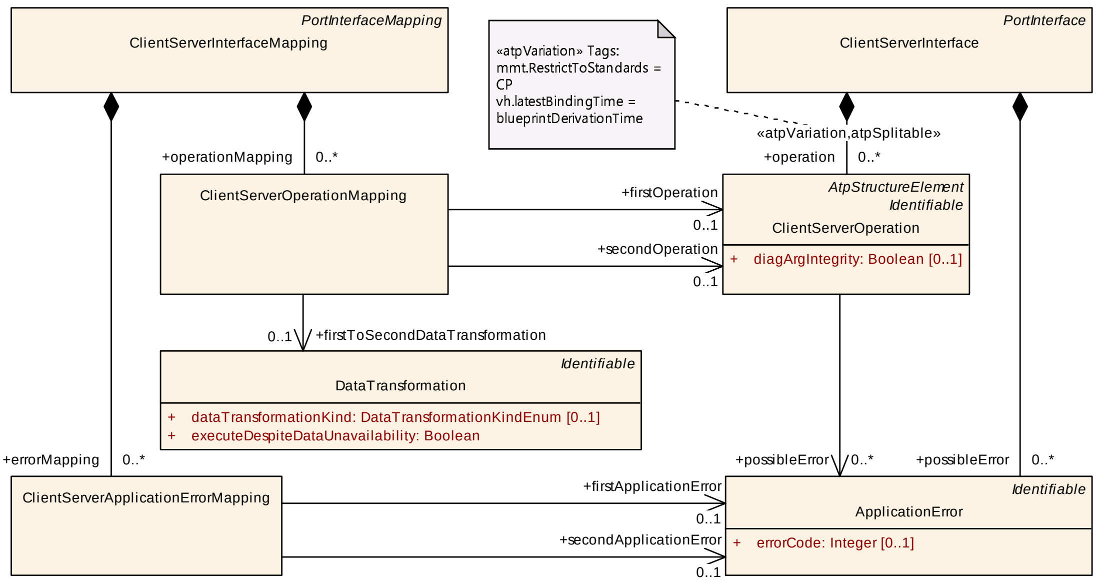


Figure 4.14: Mapping of ClientServerInterface elements and mapping of arguments

[constr_1269] Number of `arguments` shall be preserved in a `ClientServerOperationMapping` ⌈Within the context of a `ClientServerOperationMapping`, the number of `arguments` of `firstOperation` and `secondOperation` shall be identical **at any time in the workflow**.⌋()

[constr_1270] `ArgumentDataPrototype` shall be mapped only once in a `ClientServerOperationMapping` ⌈Within the context of a `ClientServerOperationMapping`, each `argument` shall only be referenced **once** in the role `firstDataPrototype` or `secondDataPrototype` **at any time in the workflow**.⌋()

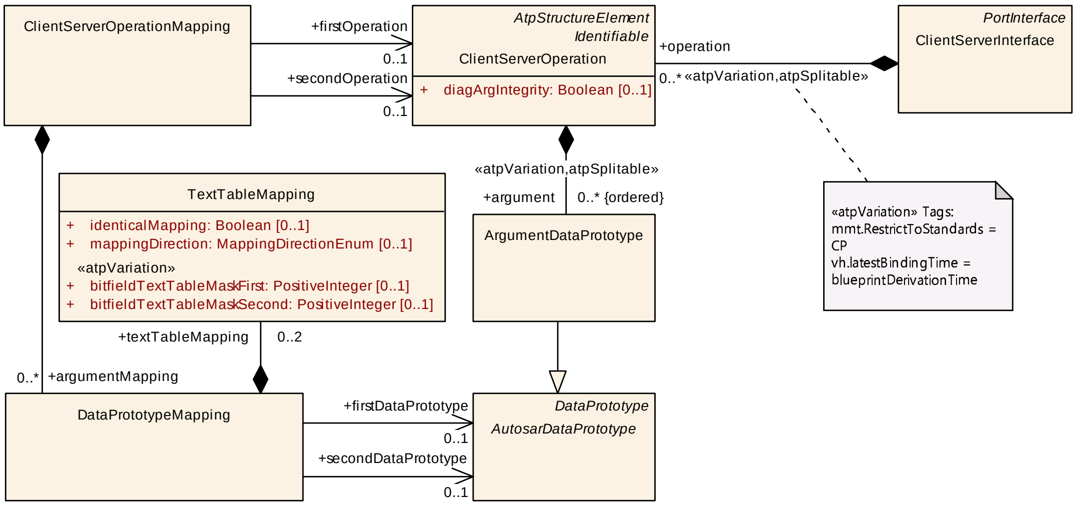


Figure 4.15: Mapping of `ArgumentDataPrototypes`

[constr_1469] Applicability of constraints depending on the existence of a data transformation ⌈[constr_1269], [constr_1270], [constr_1268], and [constr_1240] shall – **at any time in the workflow** – **not** apply under the following conditions:

- A reference from the respective `ClientServerOperationMapping` to a `DataTransformation` in the role `firstToSecondDataTransformation` exists.
- The value of the attribute `dataTransformationKind` of the referenced `DataTransformation` is set to `DataTransformationKindEnum.asymmetricFromByteArray` or `DataTransformationKindEnum.asymmetricToByteArray`.

⌋()

Class	**ClientServerInterfaceMapping**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	Defines the mapping of ClientServerOperations in context of two different ClientServerInterfaces.
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *PortInterfaceMapping*, *Referrable*
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping

▽

△

Class	ClientServerInterfaceMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
errorMapping	ClientServerApplicationErrorMapping	*	aggr	Map two different ApplicationErrors defined in the context of two different ClientServerInterfaces.
operationMapping	ClientServerOperationMapping	*	aggr	Mapping of two ClientServerOperations in two different ClientServerInterfaces

Table 4.24: ClientServerInterfaceMapping

Class	ClientServerOperationMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two particular ClientServerOperations in context of two different ClientServer Interfaces.			
Base	*ARObject*			
Aggregated by	ClientServerInterfaceMapping.operationMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
argumentMapping	DataPrototypeMapping	*	aggr	Defines the mapping of two particular ArgumentData Prototypes with unequal names or unequal semantic (resolution or range) in context of Operations.
firstOperation	ClientServerOperation	0..1	ref	First to-be-mapped ClientServerOperation of a Client ServerInterface.
firstToSecondDataTransformation	DataTransformation	0..1	ref	This reference indicates that a DataTransformation is intended in the context of the ClientServerOperation Mapping.
secondOperation	ClientServerOperation	0..1	ref	Second to-be-mapped ClientServerOperation of a Client ServerInterface.

Table 4.25: ClientServerOperationMapping

Class	ClientServerApplicationErrorMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the ability to map ApplicationErrors onto each other.			
Base	*ARObject*			
Aggregated by	ClientServerInterfaceMapping.errorMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
firstApplicationError	ApplicationError	0..1	ref	This represents the first ApplicationError in the context of the ClientServerApplicationErrorMapping.
secondApplicationError	ApplicationError	0..1	ref	This represents the second ApplicationError in the context of the ClientServerApplicationErrorMapping.

Table 4.26: ClientServerApplicationErrorMapping

[constr_1875] Existence of reference `ClientServerOperationMapping.firstOperation` ⌈For each `ClientServerOperationMapping`, the reference in the role `firstOperation` shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1876] Existence of reference `ClientServerOperationMapping.secondOperation` ⌈For each `ClientServerOperationMapping`, the reference in the role `secondOperation` shall exist **at the time when the RTE is generated**.⌋*()*

4.3.1.3 Mapping of Mode Interface Elements

[TPS_SWCT_01160] **ModeInterfaceMapping** ⌈The ModeInterfaceMapping defines the correlation of ModeDeclarationGroupPrototypes defined in the context of ModeSwitchInterfaces.⌋*(RS_SWCT_03210)*

[TPS_SWCT_01167] Validity of ModeInterfaceMapping ⌈The mapping of ModeDeclarationGroupPrototypes is only valid if these are typed by (read "refer to") compatible ModeDeclarationGroups.⌋*(RS_SWCT_03210)*

The compatibility of ModeDeclarationGroups is described in chapter 6.7.

Class	**ModeInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of ModeDeclarationGroupPrototypes in context of two different ModeInterfaces.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *PortInterfaceMapping*, *Referrable*			
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
modeMapping	ModeDeclarationGroupPrototypeMapping	0..1	aggr	Mapping of two ModeDeclarationGroupPrototypes in two different ModeInterfaces

Table 4.27: ModeInterfaceMapping

Class	**ModeDeclarationGroupPrototypeMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Defines the mapping of two particular ModeDeclarationGroupPrototypes (in the given context) that are unequally named and/or require a reference to a ModeDeclarationMappingSet in order to become compatible by definition of ModeDeclarationMappings.			
Base	*ARObject*			
Aggregated by	ModeInterfaceMapping.modeMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
firstModeGroup	ModeDeclarationGroupPrototype	0..1	ref	ModeDeclarationGroupPrototype to be mapped.
modeDeclarationMappingSet	ModeDeclarationMappingSet	0..1	ref	This represents the available mappings of Mode Declarations in the context ot this ModeDeclarationGroupPrototype.
secondModeGroup	ModeDeclarationGroupPrototype	0..1	ref	ModeDeclarationGroupPrototype to be mapped.

Table 4.28: ModeDeclarationGroupPrototypeMapping

[constr_1877] Existence of reference ModeDeclarationGroupPrototypeMapping.firstModeGroup ⌈For each ModeDeclarationGroupPrototypeMapping, the reference in the role firstModeGroup shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1878] Existence of reference ModeDeclarationGroupPrototypeMapping.secondModeGroup ⌈For each ModeDeclarationGroupPrototypeMapping, the reference in the role secondModeGroup shall exist **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01449] Semantics of a ModeDeclarationGroupPrototypeMapping ⌈A ModeDeclarationGroupPrototypeMapping shall be used to identify two ModeDeclarationGroups that afterwards shall be considered compatible. This also applies if the two ModeDeclarationGroups deviate with respect to the contained modeTransitions.⌋*(RS_SWCT_03210)*

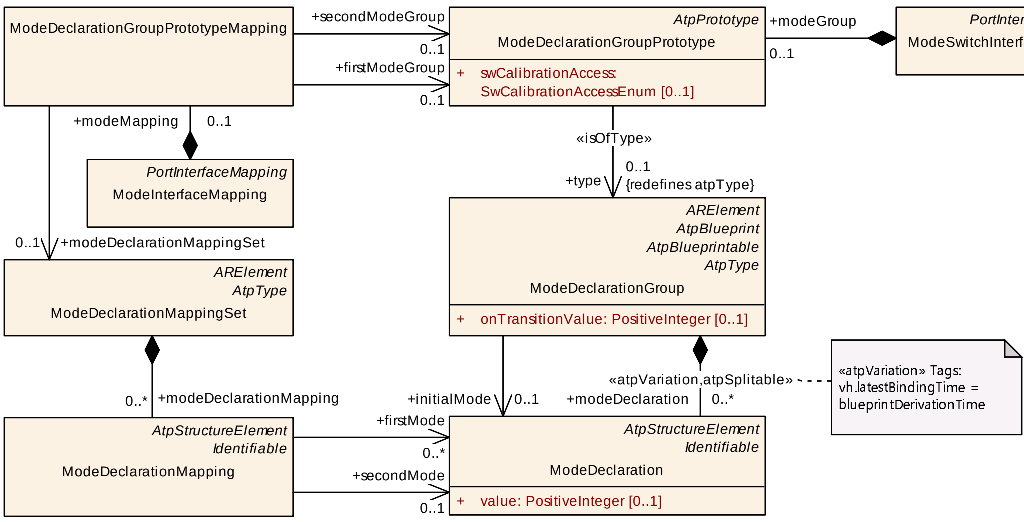


Figure 4.16: Mapping of ModeSwitchInterface elements

[constr_1246] Consistency of firstMode and secondMode in the scope of one ModeDeclarationMappingSet ⌈Within the scope of one ModeDeclarationMappingSet,

- all firstModes shall belong to one and only one ModeDeclarationGroup and
- all secondModes shall belong to one and only one **other** ModeDeclarationGroup

at any time in the workflow.⌋*()*

[constr_1247] Consistency of ModeDeclarationMappingSet with respect to the referenced firstModeGroup and secondModeGroup ⌈If a ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet exists, then

- the ModeDeclarationGroup owning the modeDeclarations referenced in the role firstMode shall be the type of the ModeDeclarationGroupPrototypeMapping.firstModeGroup and
- the ModeDeclarationGroup owning the modeDeclarations referenced in the role secondMode shall be the type of the ModeDeclarationGroupPrototypeMapping.secondModeGroup.

This rule shall be imposed **at any time in the workflow**.⌋*()*

[TPS_SWCT_01462] ModeDeclarationMapping defines the explicit correlation of ModeDeclarations ⌈The meta-class ModeDeclarationMapping defines the explicit correlation of ModeDeclarations defined in the context of two ModeDeclarationGroups.⌋()

[TPS_SWCT_01463] ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet defines the applicable set of ModeDeclarationMappings ⌈The attribute ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet defines the applicable set of ModeDeclarationMappings for the connection of ModeDeclarationGroupPrototypes typed by ModeDeclarationGroups with differently named ModeDeclarations and/or with a different number of ModeDeclarations.⌋()

Class	**ModeDeclarationMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class implements a container for ModeDeclarationGroupMappings **Tags:**atp.recommendedPackage=PortInterfaceMappingSets			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
modeDeclarationMapping	ModeDeclarationMapping	*	aggr	This represents the collection of ModeDeclarationMappings owned by the enclosing ModeDeclarationMappingSet.

Table 4.29: ModeDeclarationMappingSet

Class	**ModeDeclarationMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class implements a concrete mapping of two ModeDeclarations.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeDeclarationMappingSet.modeDeclarationMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
firstMode	ModeDeclaration	*	ref	This represents the first ModeDeclaration of the ModeDeclarationMapping. This reference has the multiplicity 1 .. * to support use cases where e.g. one mode of the mode user is mapped to several modes of the mode manager.
secondMode	ModeDeclaration	0..1	ref	This represents the second ModeDeclaration of the ModeDeclarationMapping.

Table 4.30: ModeDeclarationMapping

[constr_1879] Existence of reference ModeDeclarationMapping.firstMode ⌈For each ModeDeclarationMapping, at least one reference firstMode shall exist **at the time when the RTE is generated**.⌋()

[constr_1880] Existence of reference ModeDeclarationMapping.secondMode ⌈For each ModeDeclarationMapping, the reference secondMode shall exist **at the time when the RTE is generated**.⌋()

[TPS_SWCT_01464] `ModeDeclaration` of a mode user is mapped to exactly one `ModeDeclaration` of a mode manager ⌈The mode that corresponds to the ModeDeclaration of the Mode User is entered or exited when the mode of the mode manager that corresponds to the mapped (i.e. referenced by the same ModeDeclarationMapping) ModeDeclaration of the mode manager is entered or exited.⌋ *(RS_SWCT_03115)*

[TPS_SWCT_01465] `ModeDeclaration` of a mode user is mapped to several `ModeDeclaration`s of a mode manager ⌈The mode that corresponds to the mapped ModeDeclaration of the mode user is entered when any of the modes of the Mode Manager that correspond to ModeDeclarations referenced by the applicable ModeDeclarationMapping is entered.

The mode that corresponds to the mapped ModeDeclaration of the mode user is exited when any of the modes of the Mode Manager that correspond to ModeDeclarations referenced by the applicable ModeDeclarationMapping is exited if the new mode is not mapped to related mode of the mode user.⌋ *(RS_SWCT_03115)*

Please note if one ModeDeclaration of a mode user is mapped to **several** ModeDeclarations of a mode manager by means of several ModeDeclarationMappings the intended semantics is defined in a way that the individual mode transitions of the mode manager are representing “exit” and “enter” events for the Mode User. In other words, the individual transitions are recognizable by the mode user.

If one ModeDeclaration of a mode user is (by utilizing the multiplicity of the role firstMode) mapped to several ModeDeclarations of a mode manager in the context of a single ModeDeclarationMapping, the semantics is defined in a way that the individual mode transitions of the Mode Manager are not recognizable to the Mode User.

[constr_1209] Mapping of `ModeDeclaration`s of mode user to `ModeDeclaration` of mode manager ⌈A configuration that maps **several** ModeDeclarations representing modes of a mode user to **one** ModeDeclaration representing a mode of a mode manager shall be rejected **at any time in the workflow**.⌋ *()*

[constr_1210] Mapping of `ModeDeclaration`s of mode user to all `ModeDeclaration`s of mode manager ⌈If a ModeDeclarationMapping exists that references a ModeDeclaration representing a mode of the mode manager, then ModeDeclarationMappings shall exist that map all modes of the mode manager to modes of the mode user **at the time when the RTE is generated**.⌋ *()*

Please note that [constr_1210] prevents the existence of configurations where the mode user is not in a defined mode when no transition is ongoing.

[TPS_SWCT_01545] `ModeDeclaration` of a *mode user* that is not mapped to a `ModeDeclaration` of a *mode manager* ⌈A ModeDeclaration of a *mode user* that is not mapped to a ModeDeclaration of a *mode manager* represents a valid model. In this case the related mode is never entered nor exit during runtime of the ECU.⌋ *(RS_SWCT_03115)*

4.3.1.4 Mapping of Trigger Interface Elements

[TPS_SWCT_01161] **`TriggerInterfaceMapping`** ⌈The `TriggerInterfaceMapping` defines the correlation of `Trigger`s defined in the context `TriggerInterface`s.⌋*(RS_SWCT_03210)*

Class	**TriggerInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of unequal named Triggers in context of two different TriggerInterfaces.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *PortInterfaceMapping*, *Referrable*			
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
triggerMapping	TriggerMapping	*	aggr	Mapping of two Trigger in two different TriggerInterface

Table 4.31: TriggerInterfaceMapping

Class	**TriggerMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	Defines the mapping of two particular unequally named Triggers in the given context.			
Base	*ARObject*			
Aggregated by	TriggerInterfaceMapping.triggerMapping			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
firstTrigger	Trigger	0..1	ref	A Trigger to be mapped.
secondTrigger	Trigger	0..1	ref	A Trigger to be mapped.

Table 4.32: TriggerMapping

[constr_1881] Existence of reference `TriggerMapping.firstTrigger` ⌈For each `TriggerMapping`, the reference `firstTrigger` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1882] Existence of reference `TriggerMapping.secondTrigger` ⌈For each `TriggerMapping`, the reference `secondTrigger` shall exist **`at the time when the RTE is generated`**.⌋*()*

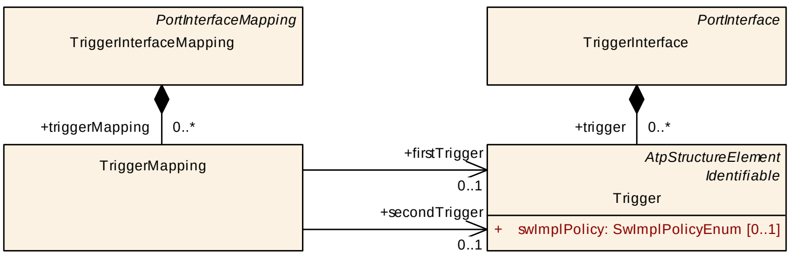


Figure 4.17: Mapping of `TriggerInterface` elements

4.3.1.5 Mapping of Elements of a composite Data Type

The mapping of elements of PortInterfaces is not limited to mapping entire DataPrototypes onto each other.

[TPS_SWCT_01023] Mapping of elements of composite data types ⌈For applications of DataInterfaces it is also possible to formally describe the mapping of elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY onto each other.⌋*(RS_SWCT_03210, RS_SWCT_03135)*

This ability (for which [constr_10087] applies) can be used if e.g. dataElements on the "source" and "target" end side are typed by different ApplicationRecordDataTypes.

In this case the mapping of elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY onto each other allows for the definition of specific pairs of elements that fulfill the compatibility rules.

[constr_10087] Restriction for the existence of a SubElementMapping ⌈The existence of a DataPrototypeMapping.subElementMapping is only supported if the PortPrototypes that are referenced by the respective SwConnector are typed by a DataInterface **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01551] Mapping of elements on the "source" end to elements on the "target" end ⌈Unless the attribute swImplPolicy is set to queued, it is not required that all elements on the "source" end need to be mapped to elements on the "target" end to achieve compatibility.⌋*(RS_SWCT_03210, RS_SWCT_03135)*

The details regarding the compatibility rules are explained in chapter 6.3.

[constr_1279] Unmapped elements of ApplicationCompositeDataTypes or ImplementationDataTypes and the attribute swImplPolicy ⌈If the attribute swImplPolicy is set to queued, then it is not allowed to have unmapped elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY on the "target" end **at the time when the RTE is generated**.⌋*()*

[constr_1280] Unmapped dataElement on the "target" end shall have an initValue ⌈If elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY are not considered in a SubElementMapping **and** the NonqueuedReceiverComSpec is aggregated by an AbstractRequiredPortPrototype referenced by the "target" end, then the enclosing dataElement shall have an initValue.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01024] Combination of ApplicationCompositeDataType and nested ImplementationDataType ⌈The mapping of elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY works for both ApplicationCompositeDataType and nested ImplementationDataTypes and even for combinations of them, i.e. one PortInterface may use an ApplicationCompositeDataType while the other PortInterface uses a nested ImplementationDataType.⌋*(RS_SWCT_03210, RS_SWCT_03135)*

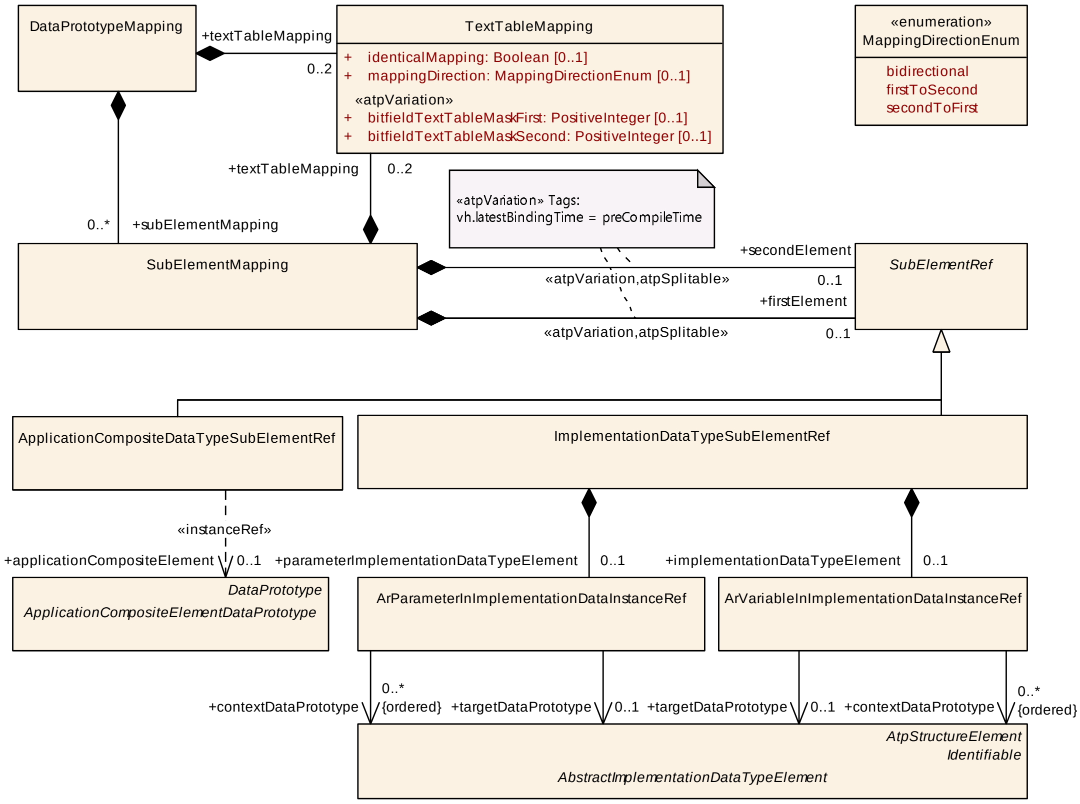


Figure 4.18: Mapping of elements of composite data types

[TPS_SWCT_01195] Mapping of composite element to primitive DataPrototype
⌈It is also possible to map an element of a composite data type on the "source" end to a primitive DataPrototype on the "target" end.

For this purpose the multiplicity of the dataElement/nvData contained in the PortPrototype on the "source" end shall be set to 1 and the multiplicity of the dataElement/nvData contained in the PortPrototype on the "target" end shall be set to 0.⌋*(RS_SWCT_03136)*

In general, the multiplicity of the respective dataElement/nvData contained in the PortPrototype on the "source" end can technically also be set to 0 but this case is reserved for future use.

[constr_1190] Only one mapping for composite to primitive use case ⌈In the case described by [TPS_SWCT_01195] only one subElementMapping shall exist at the enclosing DataPrototypeMapping **at the time when the RTE is generated**.⌋*()*

[constr_1300] Primitive DataPrototype on the "source" end shall not be mapped to element of a composite data type on the "target" end of the SwConnector ⌈The usage of DataPrototypeMapping or SubElementMapping does not support the following configuration:

- The AutosarDataPrototype contained in the PortPrototype on the "source" end of an SwConnector is typed by an ApplicationPrimitiveDataType of category VALUE or ImplementationDataType of category VALUE or category TYPE_REFERENCE that eventually resolves to category VALUE.
- The DataPrototypeMapping aggregates a subElementMapping that refers to a ImplementationDataTypeElement or ApplicationCompositeElementDataPrototype contained in the PortPrototype on the "target" end.

This rule shall be imposed **at any time in the workflow**.⌋*()*

[constr_1611] Existence of ImplementationDataTypeSubElementRef.implementationDataTypeElement as opposed to ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement ⌈For any given ImplementationDataTypeSubElementRef, either the aggregation

- ImplementationDataTypeSubElementRef.implementationDataTypeElement or
- ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement

shall exist **at any time in the workflow**.⌋*()*

In other words, the ImplementationDataTypeSubElementRef shall **either** refer to the nested hierarchy inside a VariableDataPrototype **or** a ParameterDataPrototype.

Class	**SubElementMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class allows for the definition of mappings of elements of a composite data type.			
Base	*ARObject*			
Aggregated by	DataPrototypeMapping.subElementMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
firstElement	SubElementRef	0..1	aggr	This represents the first element referenced in the scope of the mapping. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=firstElement, firstElement.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SubElementMapping**			
secondElement	SubElementRef	0..1	aggr	This represents the second element referenced in the scope of the mapping. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=secondElement, secondElement.variation Point.shortLabel vh.latestBindingTime=preCompileTime
textTable Mapping	TextTableMapping	0..2	aggr	This allows for the text-table translation of individual elements of a composite data type.

Table 4.33: SubElementMapping

Class	***SubElementRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class provides the ability to reference elements of composite data type.			
Base	*ARObject*			
Subclasses	ApplicationCompositeDataTypeSubElementRef, ImplementationDataTypeSubElementRef			
Aggregated by	SubElementMapping.firstElement, SubElementMapping.secondElement			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 4.34: SubElementRef

Class	**ImplementationDataTypeSubElementRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the specialization of SubElementMapping with respect to Implementation DataTypes.			
Base	*ARObject*, *SubElementRef*			
Aggregated by	SubElementMapping.firstElement, SubElementMapping.secondElement			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
implementation DataType Element	ArVariableIn ImplementationData InstanceRef	0..1	aggr	This represents the referenced implementationDataType Element.
parameter Implementation DataType Element	ArParameterIn ImplementationData InstanceRef	0..1	aggr	This represents the referenced ImplementationDataType Element.

Table 4.35: ImplementationDataTypeSubElementRef

Class	**ApplicationCompositeDataTypeSubElementRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the specialization of SubElementMapping with respect to Application CompositeDataTypes.			
Base	*ARObject*, *SubElementRef*			
Aggregated by	SubElementMapping.firstElement, SubElementMapping.secondElement			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	ApplicationCompositeDataTypeSubElementRef			
application Composite Element	ApplicationComposite ElementDataPrototype	0..1	iref	This represents the referenced ApplicationComposite DataPrototype. **InstanceRef implemented by:** ApplicationComposite ElementInPortInterfaceInstanceRef

Table 4.36: ApplicationCompositeDataTypeSubElementRef

[constr_1883] Existence of `ApplicationCompositeDataTypeSubElementRef.applicationCompositeElement` ⌈For each `ApplicationCompositeDataTypeSubElementRef`, the reference `applicationCompositeElement` shall exist **`at the time when the RTE is generated`**.⌋()

[constr_1184] Consistency of `rootDataPrototype` and `base` in the context of `ApplicationCompositeElementInPortInterfaceInstanceRef` ⌈The `rootDataPrototype` referenced by `ApplicationCompositeElementInPortInterfaceInstanceRef` shall be owned by the applicable subclass of `DataInterface` referenced in the role `base`.

This implies that the `rootDataPrototype` shall be a `ParameterDataPrototype` if the `base` is a `ParameterInterface`. Otherwise, the `rootDataPrototype` shall be a `VariableDataPrototype`.

This rule shall be applied at **`at any time in the workflow`**.⌋()

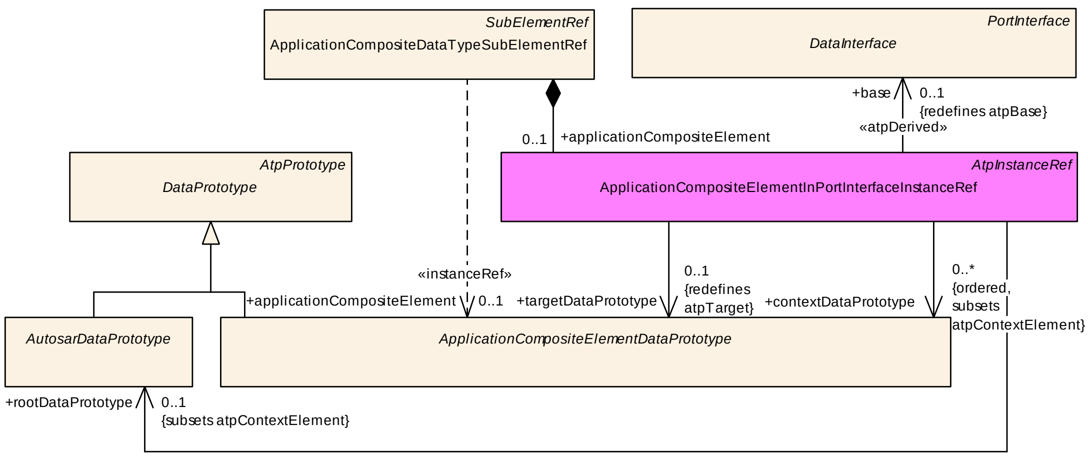


Figure 4.19: Implementation of the InstanceRef for the mapping of elements of composite application data types

[constr_1185] Consistency of data types in the context of `ApplicationCompositeElementInPortInterfaceInstanceRef` ⌈The definition of attributes `contextDataPrototype` and `targetDataPrototype` shall (via the type-prototype pattern) be enclosed in the context of the definition of the data type used to type `rootDataPrototype` **`at any time in the workflow`**.⌋()

In other words, it shall be possible to reach `contextDataPrototype` and `targetDataPrototype` by means of the type-prototype chain created by the definition of the data type used to type `rootDataPrototype`. And, as implied by the definition of the

InstanceRef, the contextDataPrototypes shall enclose each other and, eventually, the targetDataPrototype.

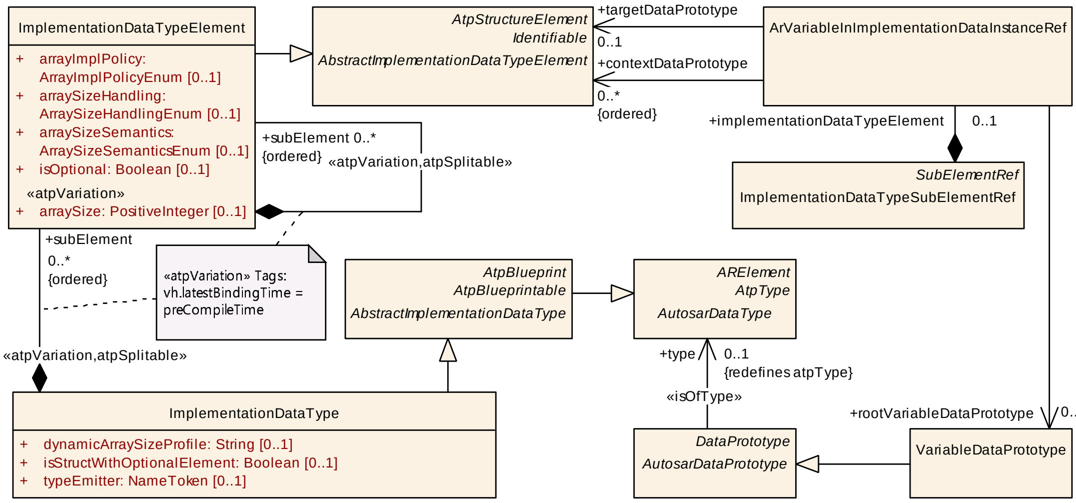


Figure 4.20: Implementation of the InstanceRef for the mapping of elements of a VariableDataPrototype typed by a composite implementation data type

[constr_1186] Consistency of data types in the context of ArVariableInImplementationDataInstanceRef ⌈The definition of attributes contextDataPrototype and targetDataPrototype shall be enclosed in the context of the definition of the data type used to type rootVariableDataPrototype **at any time in the workflow.**⌋()

[constr_1518] Consistency of data types in the context of ArParameterInImplementationDataInstanceRef ⌈The definition of attributes contextDataPrototype and targetDataPrototype shall be enclosed in the context of the definition of the data type used to type rootParameterDataPrototype **at any time in the workflow.**⌋()

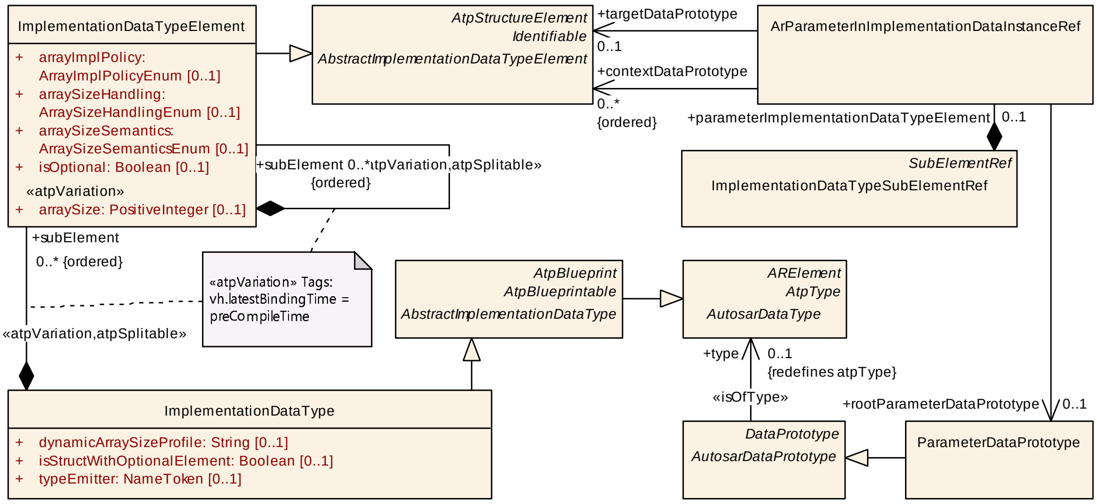


Figure 4.21: Implementation of the InstanceRef for the mapping of elements of a ParameterDataPrototype typed by a composite implementation data type

4.3.2 Data Conversion

[TPS_SWCT_01560] Supported categorys of CompuMethods for data conversion ⌈Data conversion shall be supported for AutosarDataTypes that refer to CompuMethods of category

- LINEAR,
- IDENTICAL,
- SCALE_LINEAR_AND_TEXTTABLE, as long as there is **only one linear scale**,
- TEXTTABLE,
- BITFIELD_TEXTTABLE, and
- RAT_FUNC - as long as the semantics of the latter comes down to a reciprocal linear data scaling.

⌋*(RS_SWCT_03210)*

[TPS_SWCT_01561] Application of data conversion to composite AutosarDataTypes ⌈Data conversion is also applicable for composite AutosarDataTypes. The actual conversion, however, shall be individually applied to each leaf element of a given composite AutosarDataType.⌋*(RS_SWCT_03210)*

4.3.2.1 Linear Data Scaling

A *Linear Data Scaling* can be defined under following preconditions:

[TPS_SWCT_01549] Definition of linear data scaling ⌈The term Linear Scaling is defined as follows:

1. Regarding the existence of CompuMethods one of the following cases shall apply:
   (a) The involved AutosarDataTypes refer to CompuMethods of category IDENTICAL, LINEAR, or RAT_FUNC.
   (b) If one side (sender or receiver) does not refer to a CompuMethod then a “default” CompuMethod of category IDENTICAL shall be assumed.
2. The CompuMethods refer either to compatible Units or to Units that in turn refer to compatible definitions of PhysicalDimension.
3. Both CompuMethods fulfill the following condition:

$$Int = \frac{N_0 * phys^0 + N_1 * phys^1 + N_2 * phys^2 + ... + N_i * phys^i}{D_0 * phys^0 + D_1 * phys^1 + D_2 * phys^2 + ... + D_i * phys^i}$$

   with

   - N_2=N_3=...=N_i=0

- D_1=D_2=...=D_i=0
- N_1 ≠0
- D_0 ≠0

The coefficient N_0 represents the offset and can take any value.

⌋*(RS_SWCT_03210)*

[TPS_SWCT_01550] Definition of reciprocal linear data scaling ⌈The term Reciprocal Linear Scaling is defined as follows:

1. The involved AutosarDataTypes refer to CompuMethods of category RAT_FUNC.
2. The CompuMethods refer either to compatible Units or to Units that in turn refer to compatible definitions of PhysicalDimension.
3. Both CompuMethods fulfill the following condition:

$$Int = \frac{N_0 * phys^0 + N_1 * phys^1 + N_2 * phys^2 + ... + N_i * phys^i}{D_0 * phys^0 + D_1 * phys^1 + D_2 * phys^2 + ... + D_i * phys^i}$$

with

- N_1=N_2=...=N_i=0
- D_2=D_3=...=D_i=0
- N_0 ≠0
- D_1 ≠0

The coefficient D_0 represents the (reciprocal) offset and can take any value.

⌋*(RS_SWCT_03210)*

[TPS_SWCT_01168] Linear conversion factor can be calculated ⌈In such cases a linear conversion factor can be calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys/compuPhysToInternal of the referred CompuMethods.⌋*(RS_SWCT_03210)*

4.3.2.2 Table Conversion

[TPS_SWCT_01162] Existence of TextTableMapping ⌈A TextTableMapping can be defined if the AutosarDataTypes refer to CompuMethods of category TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE, and BITFIELD_TEXTTABLE.⌋*(RS_SWCT_03210)*

Please note that the use case behind the appearance of BITFIELD_TEXTTABLE in [TPS_SWCT_01162] is the fact that BSW modules such as the Dem need to put data into the NVRAM that has the nature of single bits embedded into a composite data type.

The TextTableMapping is defined as a table based conversion.

[TPS_SWCT_01163] Conversion from firstValue to secondValue ⌈A firstValue of a valuePair is converted into the secondValue in case of a data flow from the firstDataPrototype to the secondDataPrototype.⌋*(RS_SWCT_03210)*

[TPS_SWCT_01164] Conversion from secondValue to firstValue ⌈In case of a data flow from the secondDataPrototype to firstDataPrototype the secondValue is substituted by the firstValue.⌋*(RS_SWCT_03210)*

[TPS_SWCT_01165] Invertible mapping ⌈If the mappingDirection attribute is set to bidirectional then the TextTableMapping has to be invertible. This requires that the list of all firstValues and the list of all secondValues do not contain identical values inside a list.⌋*(RS_SWCT_03210)*

[TPS_SWCT_01166] Non-invertible mapping ⌈For non-invertible TextTableMapping, a dedicated TextTableMapping for each direction can be defined.⌋*(RS_SWCT_03210)*

[constr_1303] Applicability of TextTableMapping depending on the value of CompuMethod.category ⌈If – **at the time when the RTE is generated** – a DataPrototypeMapping aggregates a TextTableMapping then only certain combinations of the value of the applicable CompuMethod.category are supported:

- category of firstDataPrototype: TEXTTABLE,
  category of secondDataPrototype: TEXTTABLE
- category of firstDataPrototype: SCALE_LINEAR_AND_TEXTTABLE,
  category of secondDataPrototype: TEXTTABLE
- category of firstDataPrototype: TEXTTABLE,
  category of secondDataPrototype: SCALE_LINEAR_AND_TEXTTABLE
- category of firstDataPrototype: BITFIELD_TEXTTABLE,
  category of secondDataPrototype: TEXTTABLE
- category of firstDataPrototype: TEXTTABLE,
  category of secondDataPrototype: BITFIELD_TEXTTABLE
- category of firstDataPrototype: BITFIELD_TEXTTABLE,
  category of secondDataPrototype: BITFIELD_TEXTTABLE

⌋*()*

To some extent, *bitfields* can be regarded as a hybrid between a primitive and a structured data type:

- On the one hand, a *bitfield* is defined in the context of a primitive ImplementationDataType.

- On the other hand, by means of the definition of a mask, it is possible to define **isolated parts** within the primitive ImplementationDataType that potentially can be totally independent of each other with respect to the semantics of the data that match the mask.

In other words, the existence of semantically independent and potentially isolated parts within the primitive ImplementationDataType creates a **similar characteristic** as if the definitions of the isolated parts were created by means of defining primitive ImplementationDataTypeElements within the context of a composite ImplementationDataType.

And because it is possible to regard the "mission statement" of a DataPrototype that refers to a CompuMethod of category BITFIELD_TEXTTABLE as to mimic the semantics of a structured data type it is also possible to apply some rules that are already in place for structured data types in this specific case as well.

This conclusion, in combination with the existence of [TPS_SWCT_01551], sets the stage for [TPS_SWCT_01583].

[TPS_SWCT_01583] Completeness of TextTableMapping is not a requirement ⌈If a DataPrototypeMapping contains one or more TextTableMapping(s) where the DataPrototype on the **sender side** refers to a CompuMethod of category BITFIELD_TEXTTABLE it is **not** required that for each possible value and each possible bit mask on the sender side corresponding values on the receiver side are specified.⌋*(RS_SWCT_03210)*

With respect to [TPS_SWCT_01583] it is still important to observe that within a single mask **all values on the sender side shall have a mapping** to the receiver side.

Otherwise, the RTE generator would not be able to create mapping code that unambiguously takes care of mapping the correct values onto each other.

[constr_1313] Completeness of TextTableMapping for the values of a given bit mask on the sender side ⌈If a DataPrototypeMapping contains one or more TextTableMapping(s) where the DataPrototype on the **sender side** refers to a CompuMethod of category BITFIELD_TEXTTABLE then all DataPrototypeMapping.textTableMapping shall aggregate a collection of TextTableMapping.valuePair where each possible value of the **sender bit mask**[9] is represented by exactly one TextTableValuePair.firstValue ([TPS_SWCT_01163]) or TextTableValuePair.secondValue ([TPS_SWCT_01164]).

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[constr_1304] Existence of attribute bitfieldTextTableMaskFirst ⌈The attribute bitfieldTextTableMaskFirst shall be defined **only if** the firstDataPrototype of a DataPrototypeMapping refers to a CompuMethod that has the value of category set to BITFIELD_TEXTTABLE.

[9] Depending on the applicable case this means either bitfieldTextTableMaskFirst (applies if [TPS_SWCT_01163] is in place) or bitfieldTextTableMaskSecond for the case of [TPS_SWCT_01164].

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1305] Existence of attribute bitfieldTextTableMaskSecond ⌈The attribute bitfieldTextTableMaskSecond shall be defined **only if** the secondDataPrototype of a DataPrototypeMapping refers to a CompuMethod that has the value of category set to BITFIELD_TEXTTABLE.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1306] Limitation of TextTableMapping for CompuMethods that have the value of category set to BITFIELD_TEXTTABLE ⌈For any TextTableMapping where both firstDataPrototype and secondDataPrototype refer to CompuMethods that have the value of category set to BITFIELD_TEXTTABLE **and** where the attribute TextTableMapping.valuePair exists the value of attribute TextTableMapping.identicalMapping shall be set to false.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1307] Consistency of values and masks in TextTableMapping ⌈If a TextTableMapping element defines bit masks as bitfieldTextTableMaskFirst or bitfieldTextTableMaskSecond then all contained TextTableMapping.valuePair.firstValues as well as all TextTableMapping.valuePair.secondValues shall **not** specify a value that would be ruled out when - depending on the given value of TextTableMapping.mappingDirection - the relevant bit mask is applied.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

Example for [constr_1307]: For a bit mask 0b00001000 only the corresponding values 8 and 0 are allowed.

Class	**TextTableMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two DataPrototypes typed by AutosarDataTypes that refer to CompuMethods of category TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE or BITFIELD_TEXTTABLE.			
Base	*ARObject*			
Aggregated by	DataPrototypeMapping.textTableMapping, SenderRecArrayTypeMapping.senderToSignalTextTableMapping, SenderRecArrayTypeMapping.signalToReceiverTextTableMapping, SenderReceiverToSignalMapping.senderToSignalTextTableMapping, SenderReceiverToSignalMapping.signalToReceiverTextTableMapping, SenderRecRecordElementMapping.senderToSignalTextTableMapping, SenderRecRecordElementMapping.signalToReceiverTextTableMapping, SubElementMapping.textTableMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bitfieldTextTableMaskFirst	PositiveInteger	0..1	attr	This attribute can be used to support the mapping of bit field to bit field, boolean values to bit fields, and vice versa. The attribute defines the bit mask for the first element of the TextTableMapping. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

▽

△

Class	TextTableMapping			
bitfieldTextTableMaskSecond	PositiveInteger	0..1	attr	This attribute can be used to support the mapping of bit field to bit field, boolean values to bit fields, and vice versa. The attribute defines the bit mask for the second element of the TextTableMapping. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
identicalMapping	Boolean	0..1	attr	If identicalMapping is set == true the values of the two referenced DataPrototypes do not need any conversion of the values.
mappingDirection	MappingDirectionEnum	0..1	attr	Specifies the conversion direction for which the TextTable Mapping is applicable.
valuePair	TextTableValuePair	*	aggr	Defines a pair of values which are translated into each other.

Table 4.37: TextTableMapping

[constr_1884] Existence of attribute `TextTableMapping.identicalMapping` ⌈For each `TextTableMapping`, the attribute `identicalMapping` shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1885] Existence of attribute `TextTableMapping.mappingDirection` ⌈For each `TextTableMapping`, the attribute `mappingDirection` shall exist **at the time when the RTE is generated**.⌋*()*

Enumeration	**MappingDirectionEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	Specifies the conversion direction for which the mapping is applicable.
Aggregated by	TextTableMapping.mappingDirection
Literal	*Description*
bidirectional	The TextTableMapping is applicable in both directions. **Tags:**atp.EnumerationLiteralIndex=0
firstToSecond	The TextTableMapping is applicable in the direction from firstDataPrototype / firstOperationArgument referring into the PortInterface of the PPortPrototype to secondDataPrototype / secondOperation Argument referring into the PortInterface of the RPortPrototype. **Tags:**atp.EnumerationLiteralIndex=1
secondToFirst	The TextTableMapping is applicable in the direction from secondDataPrototype / secondOperation Argument referring into the PortInterface of the PPortPrototype to firstDataPrototype / firstOperation Argument referring into the PortInterface of the RPortPrototype. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.38: MappingDirectionEnum

Class	**TextTableValuePair**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines a pair of text values which are translated into each other.			
Base	*ARObject*			
Aggregated by	TextTableMapping.valuePair			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	TextTableValuePair			
firstValue	Numerical	0..1	attr	Value of first DataPrototype provided similar to a numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
secondValue	Numerical	0..1	attr	Value of second DataPrototype provided similar to a numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 4.39: TextTableValuePair

[constr_1886] Existence of attribute TextTableValuePair.firstValue ⌈For each TextTableValuePair, the attribute firstValue shall exist **at the time when the RTE is generated.**⌋*()*

[constr_1887] Existence of attribute TextTableValuePair.secondValue ⌈For each TextTableValuePair, the attribute secondValue shall exist **at the time when the RTE is generated.**⌋*()*

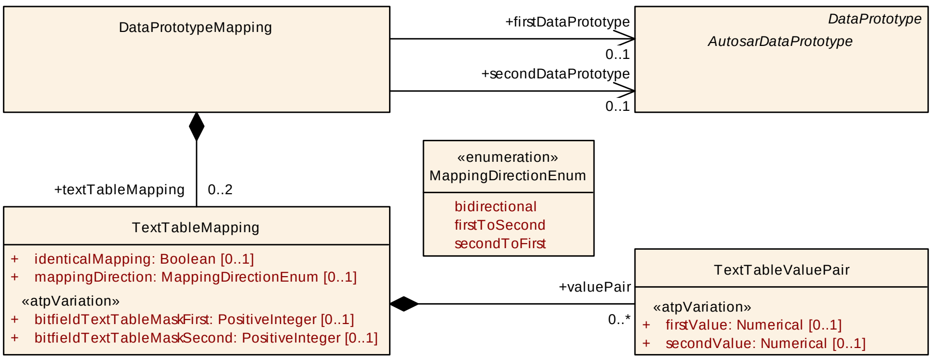


Figure 4.22: Mapping of DataPrototypes that eventually refer to CompuMethods of category TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE, and BITFIELD_TEXTTABLE

4.3.3 Relevance for Data Transformation

One (prominent) use-case for item 4 in [TPS_SWCT_01158] is the interaction between the NvBlockSwComponentType and the AUTOSAR Dcm.

Specifically, the RTE will call a data transformer to convert the *uint8*-array representation of the diagnostic data available from a PortPrototype owned by the Dcm ServiceSwComponentType to a VariableDataPrototype owned by a PortPrototype of NvBlockSwComponentType.

For the configuration of this purpose, the applicable DataPrototypeMapping refers to a DataTransformation in the role firstToSecondDataTransformation and - for the case of two connected PortPrototypes that use asymmetric data transformation - secondToFirstDataTransformation (see Figure 4.23).

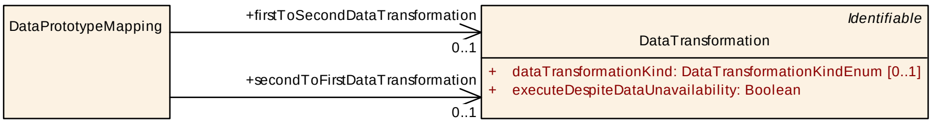


Figure 4.23: Configuration of Ecu-internal data transformation

Note that for this specific interaction between an ApplicationSwComponentType and a ServiceSwComponentType [TPS_SWCT_01579]/[TPS_SWCT_01831] applies which defines that attribute isService shall be set to false for the dataElements in PortPrototypes typed by a SenderReceiverInterface.

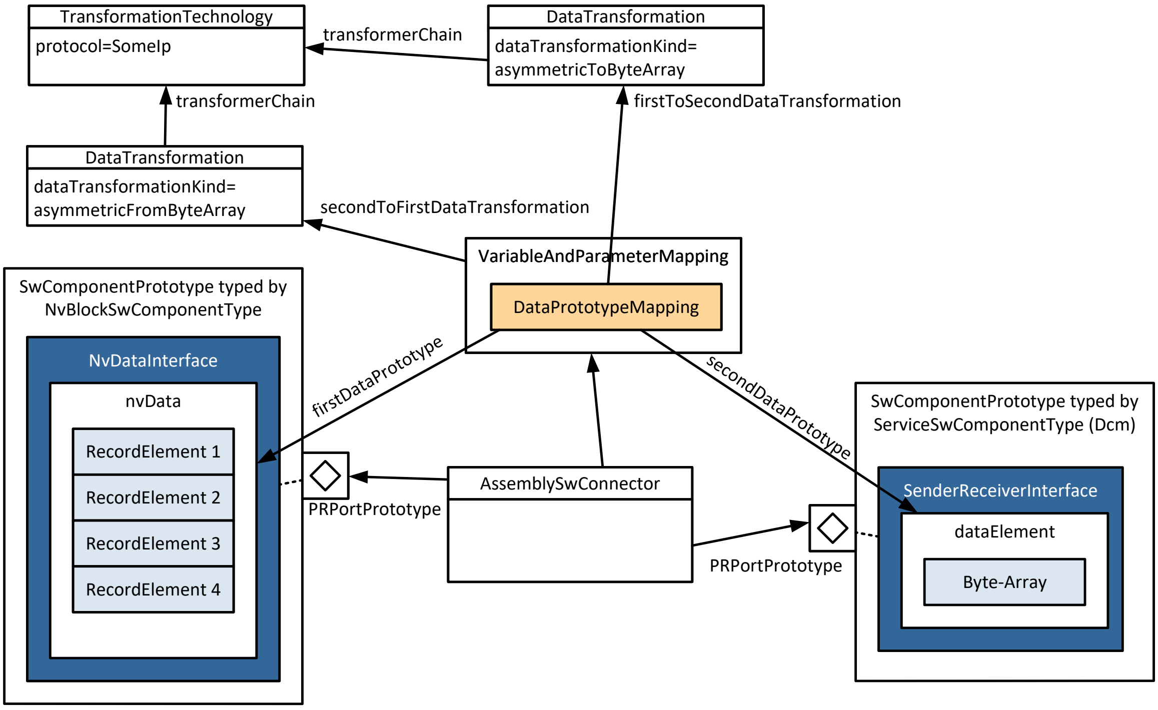


Figure 4.24: Use case for the existence of asymmetric data transformation in both directions

[TPS_SWCT_01768] Semantics of DataPrototypeMapping.secondToFirstDataTransformation ⌈For symmetric data transformations (i.e. the value of attribute DataTransformation.dataTransformationKind is set to DataTransformationKindEnum.symmetric) it is sufficient to specify the reference firstToSecondDataTransformation.

There are, however, use cases for asymmetric data transformations between two connected PRPortPrototypes and in this case it is necessary to specify each direction separately.

For this purpose, the reference secondToFirstDataTransformation exists in addition to firstToSecondDataTransformation.⌋*(RS_SWCT_03210)*

Figure 4.24 describes the most prominent use case for the necessity to specify both firstToSecondDataTransformation and secondToFirstDataTransformation.

An SwComponentPrototype typed by NvBlockSwComponentType exposes a PRPortPrototype that is connected to another PRPortPrototype attached to an SwComponentPrototype that represents the Dcm service software-component.

The PRPortPrototype on the side of the NvBlockSwComponentType in typed by an NvDataInterface that in turn aggregates a single nvData. The data type used to define the nvData is a structured data type.

The service software-component representing the Dcm, however, is not capable of dealing with structured data types. It can only handle primitive types and arrays of primitive types, e.g. bytes.

Therefore, the existence of (asymmetric) data transformers is conveniently utilized to serialize the content of the structured data type into a linear array and vice versa.

To expressly define this intended semantics, the DataPrototypeMapping defines two references:

- firstToSecondDataTransformation that refers to a DataTransformation where attribute dataTransformationKind is set to the value asymmetricToByteArray. This reference represents the direction from the NvBlockSwComponentType to the Dcm.
- secondToFirstDataTransformation that refers to a DataTransformation where attribute dataTransformationKind is set to the value asymmetricFromByteArray. This reference represents the direction from the Dcm to the NvBlockSwComponentType.

This approach to modeling is formalized in [constr_1631] and [constr_1632].

[constr_1631] Applicability of DataPrototypeMapping.secondToFirstDataTransformation ⌈The reference to DataTransformation in the role DataPrototypeMapping.secondToFirstDataTransformation shall only exist if reference DataPrototypeMapping.firstToSecondDataTransformation exists and refers to a DataTransformation where attribute dataTransformationKind exists and is **not** set to the value symmetric.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[constr_1632] Restriction for firstToSecondDataTransformation and secondToFirstDataTransformation ⌈If – **at the time when the RTE is generated** – both the reference firstToSecondDataTransformation and the reference secondToFirstDataTransformation exist in the context of the same DataPrototypeMapping then

- the firstToSecondDataTransformation shall refer to a DataTransformation with attribute dataTransformationKind set to asymmetricToByteArray and
- the secondToFirstDataTransformation shall refer to a DataTransformation with attribute dataTransformationKind set to asymmetricFromByteArray.

⌋()

Class	**DataTransformation**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	A DataTransformation represents a transformer chain. It is an ordered list of transformers.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Aggregated by	DataTransformationSet.dataTransformation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataTransformationKind	DataTransformationKindEnum	0..1	attr	This attribute controls the kind of DataTransformation to be applied.
executeDespiteDataUnavailability	Boolean	1	attr	Specifies whether the transformer chain is executed even if no input data are available.
transformerChain (ordered)	TransformationTechnology	1..*	ref	This attribute represents the definition of a chain of transformers that are supposed to be executed according to the order of being referenced from DataTransformation.

Table 4.40: DataTransformation

[constr_1888] Existence of attribute DataTransformation.executeDespiteDataUnavailability ⌈For each DataTransformation, the attribute executeDespiteDataUnavailability shall exist **at the time when the RTE is generated**.⌋()

Enumeration	**DataTransformationKindEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer
Note	This enumeration contributes to the definition of the scope of the DataTransformation.
Aggregated by	DataTransformation.dataTransformationKind
Literal	***Description***
asymmetricFromByteArray	The DataTransformation shall only be applied to the receiving end only, i.e. transform from byte array to data type. **Tags:**atp.EnumerationLiteralIndex=0
asymmetricToByteArray	The DataTransformation shall be applied to the sending end only, i.e. from data type to byte array. **Tags:**atp.EnumerationLiteralIndex=1
symmetric	The DataTransformation shall be applied at both the sending and the receiving end of the communication. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.41: DataTransformationKindEnum

4.4 Port Annotation

4.4.1 Introduction

[TPS_SWCT_01203] **PortPrototype may own port annotations** ⌈In addition to the formal specification required to implement the communication via ports, a PortPrototype may own so-called port annotations.

They do not directly influence the signature of calls via this PortPrototype, but contain further information that may be useful for the application developers of the components on both sides of the connection.⌋*(RS_SWCT_02110)*

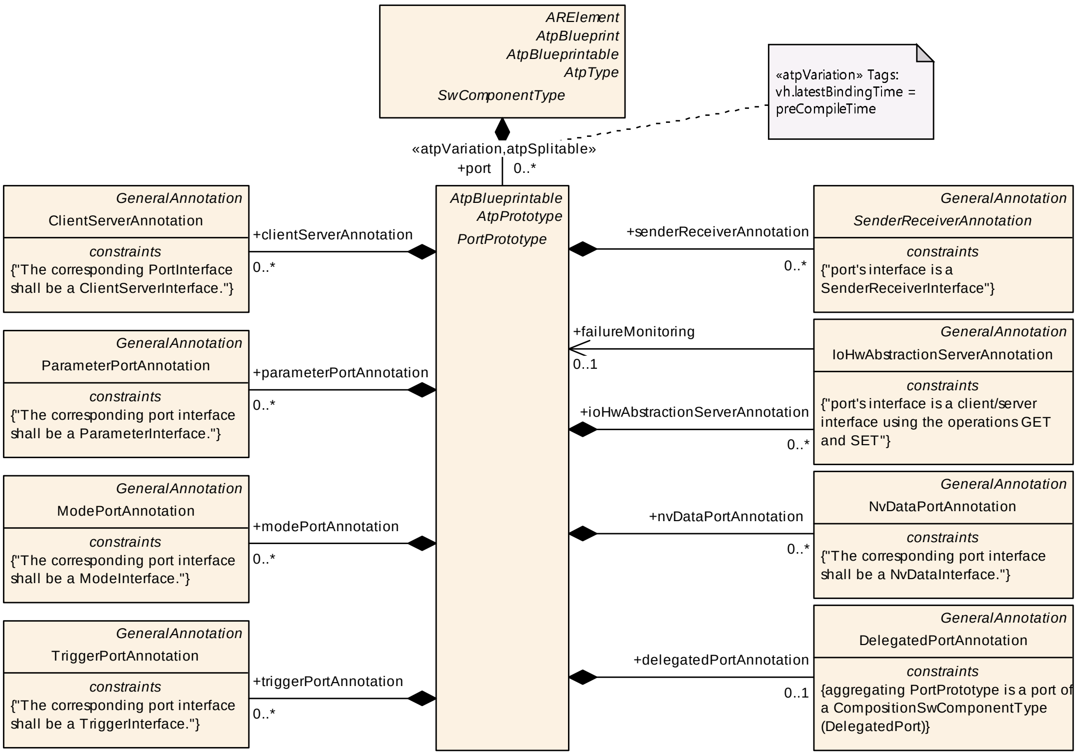


Figure 4.25: Application Level Port Annotations Overview

A summary of port-level annotations can be found in Figure 4.25.

[TPS_SWCT_01204] **GeneralAnnotation** ⌈Beside formally specified attributes it is also possible to place textual information as provided in GeneralAnnotation.⌋*(RS_SWCT_02110)*

4.4.2 SenderReceiverAnnotation

Embedded automotive software is used to implement open-loop and closed-loop control-algorithms. Therefore, a software-component description has to accommodate typical control engineering description means which have only indirect influence of the embedded software itself.

These annotations provide the (function-) developer with a direct indication whether a certain software-component is appropriate for the control-algorithm to be designed. A typical annotation is the signal quality which is characterized by several properties. Each of the property is an annotation in its own.

[TPS_SWCT_01205] Typical annotations for sender/receiver communication ⌈Typical annotations for sender/receiver communication are:

- **Signal Age**: this attribute expresses that the associated software-component will only work correctly given that the propagation of the signal from a sensor to a consumer can be finished within a particular time-limit. Of course, this cannot be identified on component or role level, but has to take into account the instance view as well as the actual ECU- and bus-scheduling.
- **Raw**: a raw signal is typically taken directly from the basic software modules of the ECU abstraction layer. In particular, no sensor software-component has filtered its original value. A `dataElement` in an `RPortPrototype` of a `SwComponentType` using this annotation indicates to the control engineer (who develops a control-algorithm for this component) that the signal has to be filtered (This relationship applies for `SenderReceiverInterface`s).
- **Filtered**: this attribute indicates that a raw signal has been manipulated by some application software-components by using a certain filter.
- **Computed**: this attribute indicates that this signal is not measured directly but calculated from tentatively several other measured or calculated signals. In a vehicle, there might be alternative signals to be used from other components having a better quality, e.g. a raw signal.
- **Min**: this annotation indicates that the signal carries a minimum value. If, for example, a reference value computed in the software-component is below that value some dedicated actions (e.g. failure-mode) might have to be taken.
- **Max**: this annotation indicates that the signal carries a maximum value. If, for example, a reference value computed in the software-component is above that value some dedicated actions (e.g. failure-mode) might have to be taken.

In the meta-model this aspect is implemented by the abstract meta-class `SenderReceiverAnnotation` which represents the base class of both `SenderAnnotation` and `ReceiverAnnotation`.⌋*(RS_SWCT_02110)*

The relationship of abstract meta-class `SenderReceiverAnnotation` to `SenderAnnotation` and `ReceiverAnnotation`is depicted in Figure 4.26.

Class	***SenderReceiverAnnotation*** (abstract)
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Annotation of the data elements in a port that realizes a sender/receiver interface.
Base	*ARObject*, *GeneralAnnotation*
Subclasses	ReceiverAnnotation, SenderAnnotation

▽

△

Class	*SenderReceiverAnnotation* (abstract)			
Aggregated by	*PortPrototype*.senderReceiverAnnotation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
computed	Boolean	0..1	attr	Flag whether this data element was not measured directly but instead was calculated from possibly several other measured or calculated values.
dataElement	VariableDataPrototype	0..1	ref	The instance of VariableDataPrototype annotated.
limitKind	DataLimitKindEnum	0..1	attr	This min or max has not to be mismatched with the min- and max for data-value in a compu-method. For example, this annotation shows when the result of the calculation performed in a RunnableEntity owned by one AtomicSw ComponentType is transmitted to another AtomicSw ComponentType whose RunnableEntity will use this value as a limit, e.g. the max.power which can be used by that software-component, or the current min. slip.
processingKind	ProcessingKindEnum	0..1	attr	This attribute controls how data is processed according to the possible values of ProcessingKindEnum.

Table 4.42: SenderReceiverAnnotation

Class	SenderAnnotation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation of a sender port, specifying properties of data elements that don't affect communication or generation of the RTE.			
Base	*ARObject*, *GeneralAnnotation*, *SenderReceiverAnnotation*			
Aggregated by	*PortPrototype*.senderReceiverAnnotation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.43: SenderAnnotation

Class	ReceiverAnnotation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation of a receiver port, specifying properties of data elements that don't affect communication or generation of the RTE. The given attributes are requirements on the required data.			
Base	*ARObject*, *GeneralAnnotation*, *SenderReceiverAnnotation*			
Aggregated by	*PortPrototype*.senderReceiverAnnotation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
signalAge	MultidimensionalTime	0..1	aggr	The maximum allowed age of the signal since it was originally read by a sensor. This is a requirement specified on the receiver side.

Table 4.44: ReceiverAnnotation

Enumeration	ProcessingKindEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Kind of processing which has been applied to a data element.
Aggregated by	*SenderReceiverAnnotation*.processingKind
Literal	**Description**

▽

△

Enumeration	**ProcessingKindEnum**
filtered	Indicates that a raw signal has been manipulated by some application software components by using filters. **Tags:**atp.EnumerationLiteralIndex=0
none	Indicates that none of the other option apply. **Tags:**atp.EnumerationLiteralIndex=1
raw	Specifies that a signal is taken directly from the basic software modules, i.e. from the ECU abstraction layer. It indicates to a developer that the control algorithm in the software has to provide filters. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.45: ProcessingKindEnum

Enumeration	**DataLimitKindEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Indicates whether the data element carries a minimum or maximum value, thereby limiting the current range of another value.
Aggregated by	*SenderReceiverAnnotation*.limitKind
Literal	***Description***
max	Limitation to maximum value **Tags:**atp.EnumerationLiteralIndex=0
min	Limitation to minimum value **Tags:**atp.EnumerationLiteralIndex=1
none	No limitation applicable **Tags:**atp.EnumerationLiteralIndex=2

Table 4.46: DataLimitKindEnum

[TPS_SWCT_01206] Min and Max annotations are valid for a certain amount of time ⌈The Min and Max annotations are valid for a certain amount of time. The value is likely to change to another valid value while the ECU is running. e.g. the maximal torque which can be requested from an engine is a typical use-case.⌋*(RS_SWCT_02110)*

This value might vary depending on e.g. the status of the climate control system. Therefore, these annotations shall not be mismatched with the min and max attributes of `CompuMethod`s.

The application level port annotations for sender/receiver communication have to be associated to each `dataElement` in a `PortPrototype`, e.g. there might be a “raw” `dataElement` and a “filtered” `dataElement` in the same `PortPrototype`!

[TPS_SWCT_01207] `VariableDataPrototype`s use the same application-level `SenderReceiverAnnotation` ⌈Furthermore, if two `VariableDataPrototype`s use the same application-level `SenderReceiverAnnotation`, a reference from the annotation to the `VariableDataPrototype`s will be established by an appropriate tool.⌋*(RS_SWCT_02110)*

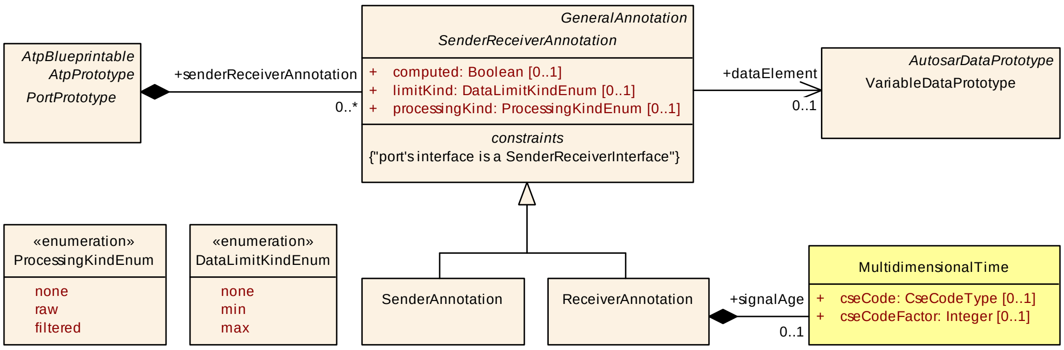


Figure 4.26: SenderReceiverAnnotation

[TPS_SWCT_01208] Grouping for SenderReceiverAnnotation ⌈The SenderReceiverAnnotation for sender/receiver communication are grouped into

- processing type, indicating to some extent the direct quality of the signal,
- computed, which is just a flag or,
- limit type, showing the component expects an actual limit.

In the case of an RPortPrototype, the signal age of the value, carried by the associated SwConnector, can be specified. Each of these groups can be interpreted as a property of the signal-quality.⌋*(RS_SWCT_02110)*

For more information about meta-class SenderReceiverAnnotation please refer to Figure 4.26.

[constr_4004] Context of SenderReceiverAnnotation ⌈A SenderReceiverAnnotation shall only be aggregated by a PortPrototype typed by a SenderReceiverInterface **at any time in the workflow**.⌋*()*

4.4.3 ClientServerAnnotation

[TPS_SWCT_01209] ClientServerAnnotation ⌈The ClientServerAnnotation can be used to provide more information with respect to the ClientServerOperation of the PortPrototype.⌋*(RS_SWCT_02110)*

Class	**ClientServerAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port regarding a certain Operation.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.clientServerAnnotation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operation	ClientServerOperation	0..1	ref	This represents the ClientServerOperation that the ClientServerAnnotation corresponds to.

Table 4.47: ClientServerAnnotation

The main use-case is to define additional information related to the ClientServer-Operation.

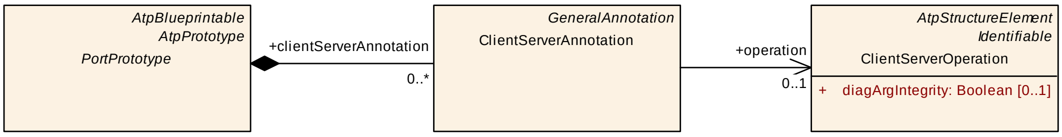


Figure 4.27: ClientServerAnnotation

[constr_4005] Context of ClientServerAnnotation ⌈A ClientServerAnnotation shall only be aggregated by a PortPrototype typed by a ClientServerInterface **at any time in the workflow**.⌋*()*

4.4.4 Annotation for the I/O Hardware Abstraction Layer

Within the ECU-Abstraction Layer there are ECU-signals defined. These signals represent the electrical signals as they arrive in the micro-controller peripheral and are fetched from the registers via the MCAL.

Access to the I/O Hardware Abstraction Layer is done via service interfaces, i.e. the I/O Hardware Abstraction Layer provides GET- and SET-operations at the specified service ports of a SensorActuatorSwComponentType.

[TPS_SWCT_01524] Usage of IoHwAbstractionServerAnnotation ⌈IoHwAbstractionServerAnnotation can be used for all kinds of PortInterfaces **except** NvDataInterface.⌋*(RS_SWCT_02110)*

Class	**IoHwAbstractionServerAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	The IoHwAbstractionServerAnnotation will only be used from a sensor- or an actuator component while interacting with the IoHwAbstraction layer. Note that the "server" in the name of this meta-class is not meant to restrict the usage to ClientServer Interfaces.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.ioHwAbstractionServerAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
age	MultidimensionalTime	0..1	aggr	In case of a SET operation, the age will be interpreted as Delay while in a GET operation (input) it specifies the Lifetime of the signal within the IoHwAbstraction Layer **Tags:**xml.sequenceOffset=10
argument	ArgumentDataPrototype	0..1	ref	Reference to the corresponding ArgumentDataPrototype. **Tags:**xml.sequenceOffset=20
bswResolution	Float	0..1	attr	This value is determined by an appropriate combination of the range, the unit as well as the data-elements type, i.e. (ecuSignalRange.upperLimit-ecuSignalRange.lower Limit) / (2ˆdatatypelength - 1) **Tags:**xml.sequenceOffset=30

▽

△

Class	**IoHwAbstractionServerAnnotation**			
dataElement	VariableDataPrototype	0..1	ref	Reference to the corresponding VariableDataPrototype. **Tags:**xml.sequenceOffset=40
failure Monitoring	PortPrototype	0..1	ref	This is only applicable in SET operations. If it is enabled, the IoHwAbstraction layer will monitor the result of the operation and issue an diagnostic signal. This means especially, that an additional client-server port has to be created. Tools can use this information to cross-check whether for each data-element in a SET operation with FailureMonitoring enabled an additional port is created The referenced port monitors a failure in the to be monitored VariableDataPrototype of the IoHwAbstraction layer. The referenced port has to be another port of the same Actuator or Sensor Component. **Tags:**xml.sequenceOffset=50
filtering Debouncing	FilterDebouncingEnum	0..1	attr	This attribute is used to indicate what kind of filtering/ debouncing has been put to the signal in the IoHw Abstraction layer. rawData means that no modification of the signal has been applied. This is the default value debounceData means that the signal is a mean value waitTimeData means that the signal is delivered by a GET operation after a certain amount of time **Tags:**xml.sequenceOffset=60
pulseTest	PulseTestEnum	0..1	attr	This attribute indicates to the connected SensorActuator SwComponentType whether the VariableDataPrototype can be used to generate pulse test sequences using the IoHwAbstraction layer **Tags:**xml.sequenceOffset=70
trigger	Trigger	0..1	ref	Reference to the corresponding Trigger. **Tags:**xml.sequenceOffset=80

Table 4.48: IoHwAbstractionServerAnnotation

Enumeration	**FilterDebouncingEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	This enumeration defines possible values for the filter debouncing strategy.
Aggregated by	IoHwAbstractionServerAnnotation.filteringDebouncing
Literal	*Description*
debounceData	The signal is a mean value **Tags:**atp.EnumerationLiteralIndex=0
rawData	Means that no modification of the signal has been applied. This is the default value **Tags:**atp.EnumerationLiteralIndex=1
waitTimeDate	The signal is delivered by a GET operation after a certain amount of time **Tags:**atp.EnumerationLiteralIndex=2

Table 4.49: FilterDebouncingEnum

Enumeration	PulseTestEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	This element indicates to the connected Actuator Software component whether the data-element can be used to generate pulse test sequences using the IoHwAbstraction layer
Aggregated by	IoHwAbstractionServerAnnotation.pulseTest
Literal	Description
disable	Disables the pulse test **Tags:**atp.EnumerationLiteralIndex=0
enable	Enables the pulse test **Tags:**atp.EnumerationLiteralIndex=1

Table 4.50: PulseTestEnum

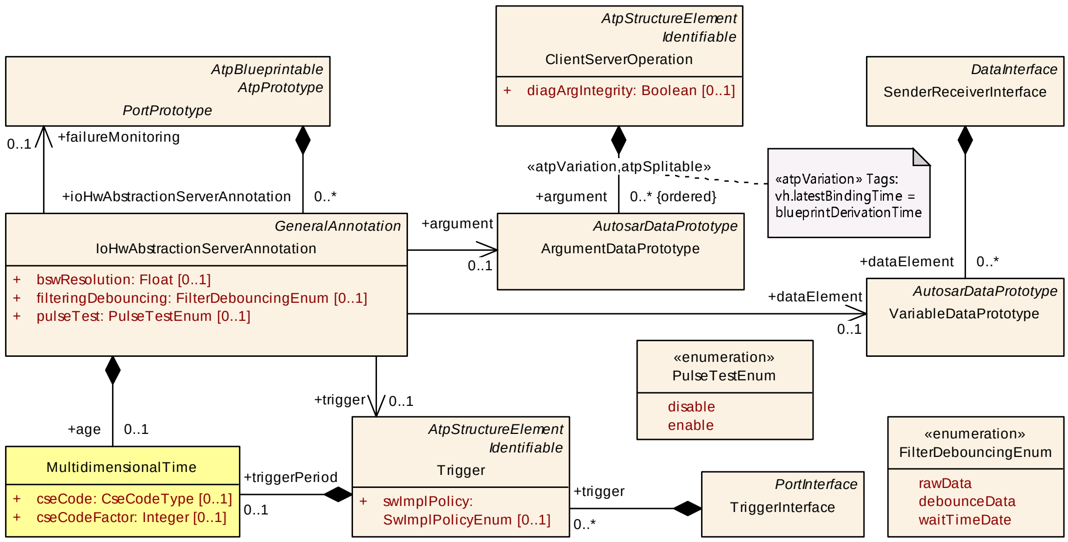


Figure 4.28: IoHwAbstractionServerAnnotation

[TPS_SWCT_01211] Assign several annotations to ArgumentDataPrototype
⌈The ClientServerOperations provide an ArgumentDataPrototype where several annotations can be assigned to.⌋*(RS_SWCT_02110)*

They are depicted in the IoHwAbstractionServerAnnotation meta-class in Figure 4.28.

A detailed description of the attributes can be found in the IoHwAbstraction Layer software specification document [16].

For example, the signal age has a very dedicated meaning in this particular interface with respect to a register whereas the signal age in the SenderReceiverAnnotation is more generic. Especially, there is no relationship with the micro-controller peripherals.

4.4.5 Parameter Port Annotation

[TPS_SWCT_01212] **ParameterPortAnnotation** ⌈The ParameterPortAnnotation can be used to provide more information with respect to calibration parameter prototypes of the PortPrototype.

The data provided at the PortPrototype is calibration parameters. The ParameterPortAnnotation provides a reference to a particular ParameterDataPrototype.⌋*(RS_SWCT_02110)*

Class	**ParameterPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain ParameterDataPrototype.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.parameterPortAnnotation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
parameter	ParameterDataPrototype	0..1	ref	The instance of annotated ParameterDataPrototype.

Table 4.51: ParameterPortAnnotation

The main use-case is to allow easy access to the information which calibration parameters influence the data on the PortPrototype.

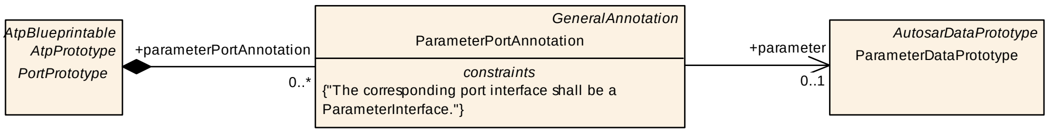


Figure 4.29: ParameterPortAnnotation

[constr_4006] Context of ParameterPortAnnotation ⌈A ParameterPortAnnotation shall only be aggregated by a PPortPrototype owned by a ParameterSwComponentType **at any time in the workflow**.⌋*()*

4.4.6 Mode Port Annotation

[TPS_SWCT_01213] **ModePortAnnotation** ⌈The ModePortAnnotation can be used to provide more information with respect to the mode declaration group prototype of the PortPrototype.⌋*()*

The main use-case is to allow for the definition of additional information related to the mode declaration group prototype.

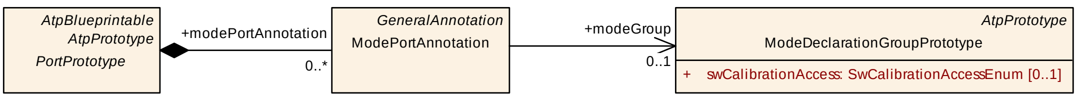


Figure 4.30: ModePortAnnotation

Class	**ModePortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain ModeDeclarationGroupPrototype.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.modePortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	0..1	ref	The instance of annotated ModeDeclarationGroupPrototype.

Table 4.52: ModePortAnnotation

[constr_4007] Context of ModePortAnnotation ⌈A ModePortAnnotation shall only be aggregated by a PortPrototype typed by a ModeSwitchInterface **at any time in the workflow**.⌋*()*

4.4.7 Trigger Port Annotation

[TPS_SWCT_01214] TriggerPortAnnotation ⌈The TriggerPortAnnotation can be used to provide more information with respect to the Trigger of the PortPrototype.⌋*(RS_SWCT_02110)*

Class	**TriggerPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain Trigger.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.triggerPortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
trigger	Trigger	0..1	ref	The instance of annotated trigger.

Table 4.53: TriggerPortAnnotation

The main use-case is to define additional information related to the trigger.

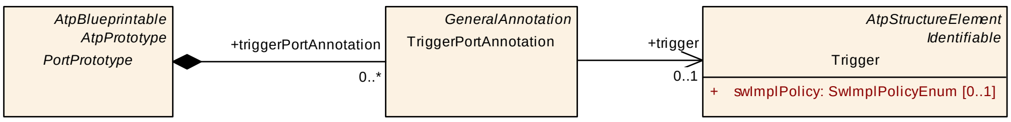


Figure 4.31: TriggerPortAnnotation

[constr_4008] Context of TriggerPortAnnotation ⌈A TriggerPortAnnotation shall only be aggregated by a PortPrototype typed by a TriggerInterface **at any time in the workflow**.⌋*()*

4.4.8 Non Volatile Data Port Annotation

[TPS_SWCT_01215] **NvDataPortAnnotation** ⌈The NvDataPortAnnotation can be used to provide more information with respect to the non-volatile data of the PortPrototype.⌋*(RS_SWCT_02110)*

Class	NvDataPortAnnotation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port regarding a certain VariableDataPrototype.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.nvDataPortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
variable	VariableDataPrototype	0..1	ref	The instance of nv data annotated.

Table 4.54: NvDataPortAnnotation

The main use-case is to define additional information related to the non-volatile data elements.

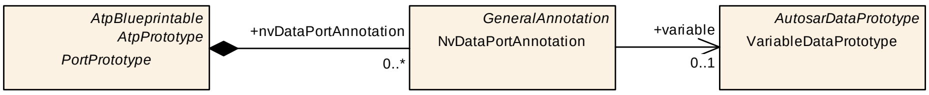


Figure 4.32: NvDataPortAnnotation

[constr_4009] Context of NvDataPortAnnotation ⌈An NvDataPortAnnotation shall only be aggregated by a PortPrototype typed by an NvDataInterface **at any time in the workflow**.⌋*()*

4.4.9 Delegated Port Annotations

[TPS_SWCT_01216] **DelegatedPortAnnotation** ⌈The DelegatedPortAnnotation is used to define the Signal Fan In or Signal Fan Out inside the CompositionSwComponentType.

This information is used to pre-define and pre-check resulting communication patterns in the VFB (1:n, n:1, 1:1) if empty CompositionSwComponentTypes are used as interface definition for sub-systems.

The DelegatedPortAnnotation guides either the system designer in connecting the empty CompositionSwComponentType or the sub-system designer in applying communication pattern (1:n, n:1, 1:1) inside the CompositionSwComponentType.⌋*(RS_SWCT_02110)*

[TPS_SWCT_01217] **Semantics of DelegatedPortAnnotation.signalFan** ⌈The attribute values have following definition:

- **single**: the internal connections in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that each dataElement present in the SenderReceiverInterfaces or operation in the ClientServerInterfaces of the outer PortPrototype is involved in a 1:1 communication pattern only.

- **nfold**: The internal connections in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that at least one dataElement present in the SenderReceiverInterfaces or one operation in the ClientServerInterfaces of the outer PortPrototype is involved in a 1:n or n:1 communication pattern.

⌋*(RS_SWCT_02110)*

[constr_4010] Context of DelegatedPortAnnotation ⌈A DelegatedPortAnnotation shall only be aggregated by a PortPrototype aggregated by a CompositionSwComponentType.⌋*()*

Class	**DelegatedPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a "delegated port" to specify the Signal Fan In or Signal Fan Out inside the CompositionSw ComponentType.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*PortPrototype*.delegatedPortAnnotation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
signalFan	SignalFanEnum	0..1	attr	Specifies the Signal Fan In or Signal Fan Out inside the Composition Type.

Table 4.55: DelegatedPortAnnotation

Enumeration	**SignalFanEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Signal Fan inside the Composition Component Type.
Aggregated by	DelegatedPortAnnotation.signalFan
Literal	***Description***
nfold	The connections internally in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that at least one data element present in the S/R interface or one ClientServerOperation in the C/S interface of the outer PortPrototype is involved in a 1:n or n:1 communication pattern. **Tags:**atp.EnumerationLiteralIndex=0
single	The connections internally in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that each VariableDataPrototype present in the S/R interface or ClientServerOperation in the C/S interface of the outer PortPrototype is involved in a 1:1 communication pattern only. **Tags:**atp.EnumerationLiteralIndex=1

Table 4.56: SignalFanEnum

4.4.10 General Annotation

Besides, formally specified attributes it is also possible to place textual information as provided in the abstract GeneralAnnotation (see Figure 4.33 for an overview).

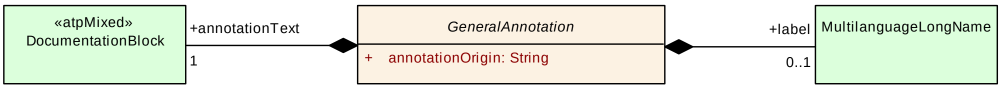


Figure 4.33: textual information in annotations

Class	***GeneralAnnotation*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::GeneralAnnotation			
Note	This class represents textual comments (called annotations) which relate to the object in which it is aggregated. These annotations are intended for use during the development process for transferring information from one step of the development process to the next one. The approach is similar to the "yellow pads" ... This abstract class can be specialized in order to add some further formal properties.			
Base	*ARObject*			
Subclasses	Annotation, ClientServerAnnotation, DelegatedPortAnnotation, IoHwAbstractionServerAnnotation, ModePortAnnotation, NvDataPortAnnotation, ParameterPortAnnotation, *SenderReceiverAnnotation*, TriggerPortAnnotation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
annotationOrigin	String	1	attr	This attribute identifies the origin of the annotation. It is an arbitrary string since it can be an individual's name as well as the name of a tool or even the name of a process step. **Tags:**xml.sequenceOffset=30
annotationText	DocumentationBlock	1	aggr	This is the text of the annotation. **Tags:**xml.sequenceOffset=40
label	MultilanguageLongName	0..1	aggr	This is the headline for the annotation. **Tags:**xml.sequenceOffset=20

Table 4.57: GeneralAnnotation

4.5 Communication Specification

[TPS_SWCT_01218] Big picture of ComSpec ⌈The highest level of description of information exchanged between components in an AUTOSAR system is the PortInterfaces, as shown in earlier sections.

Such PortInterface however, only describes structure and does not include information about whether communication needs to be done reliably, or whether an initial value exists in case the real data is not yet available.

This information is role-specific, i.e. it shall be applied on the level of PortPrototypes rather than PortInterfaces. Therefore, most communication-relevant attributes are related to the PortPrototypes of an SwComponentType.

The communication attributes are organized in a so-called **communication specification** (in terms of the meta-model: ComSpec) classes.⌋*(RS_SWCT_02030)*

Note that the communication specification is optional, i.e. its existence is not required in any case. Figures 4.34 and 4.35 provide an overview of communication specifications. The derived meta-classes are explained in the following sub-chapters.

As explained before, ComSpec meta-classes which are required on the level of a SwComponentType are attached to the PortPrototype declarations which in turn are part of the definition of a SwComponentType.

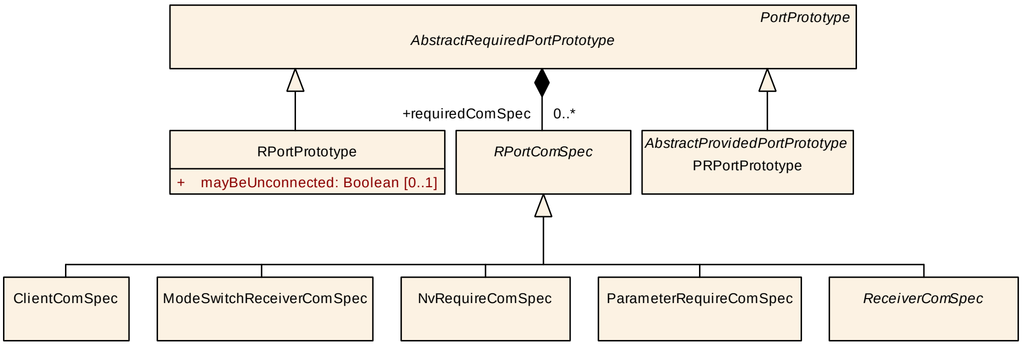


Figure 4.34: Overview of communication attributes of RPortPrototype

Nevertheless, the usage of ComSpecs is **not** restricted to the PortPrototypes of AtomicSwComponentTypes (for more details please refer to section 2.5).

Sections 7.5.1 and 7.5.2 then explain the sender-receiver and client-server communication patterns with respect to the RTE, the RTE events and the corresponding communication attributes.

Several ComSpecs allow defining initValues in relation to the associated DataPrototype. For further details about the representation of initValues please refer to section 5.7.2.

Furthermore, [constr_1043] applies such that only specific subclasses of ComSpec can be owned by PortPrototypes typed by the corresponding kind of PortInterface.

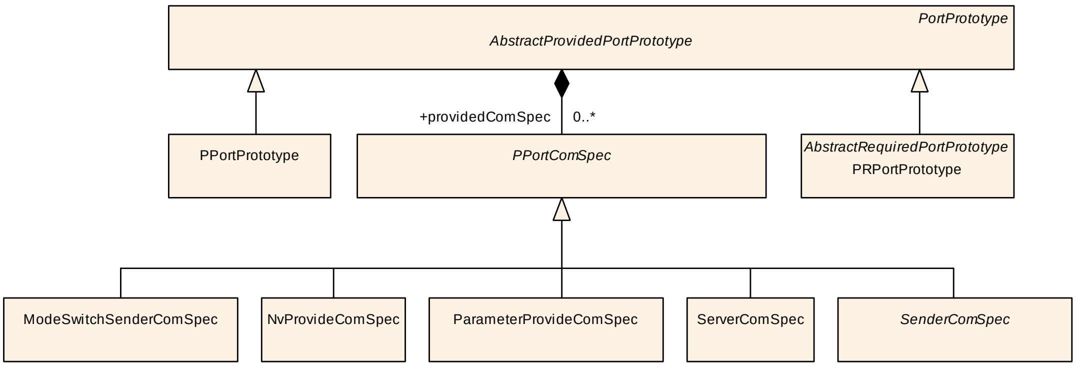


Figure 4.35: Overview of communication attributes of PPortPrototype

[constr_1290] Limitation on the number of PPortComSpecs in the context of one PPortPrototype ⌈Within the context of one PPortPrototype, there can only be **one** (sub-class of) PPortComSpec that references a given

- dataElement (in the case of SenderComSpec),
- operation (in the case of ServerComSpec),
- modeGroup (in the case of ModeSwitchSenderComSpec),

- variable (in the case of NvProvideComSpec), or
- parameter (in the case of ParameterProvideComSpec)

at the time when the contract phase generation is executed.⌋()

In other words, it is not allowed that two or more PPortComSpec exist in the context of a one PPortPrototype that refer to the same dataElement or operation.

[constr_1291] Limitation on the number of RPortComSpecs in the context of one PPortPrototype ⌈Within the context of one RPortPrototype, there can only be **one** RPortComSpec that references a given

- dataElement (in the case of ReceiverComSpec),
- operation (in the case of ClientComSpec),
- modeGroup (in the case of ModeSwitchReceiverComSpec),
- variable (in the case of NvRequireComSpec), or
- parameter (in the case of ParameterRequireComSpec)

at the time when the contract phase generation is executed.⌋()

In other words, it is not allowed that two or more RPortComSpec exist in the context of a one RPortPrototype that refer to the same dataElement or operation.

[TPS_SWCT_01454] PRPortPrototype can own both RPortComSpecs and PPortComSpecs ⌈In contrast to PPortPrototype and RPortPrototype, PRPortPrototype can own both RPortComSpecs and PPortComSpecs at the same time.⌋ *(RS_SWCT_02030, RS_SWCT_03250)*

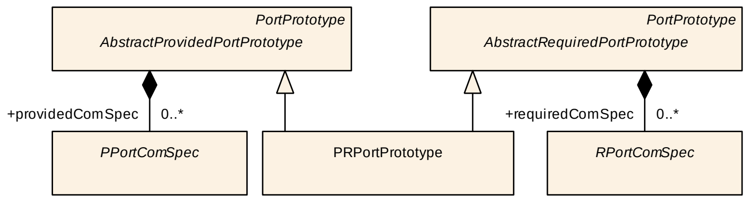


Figure 4.36: Modeling of ComSpecs for PRPortPrototype

Nevertheless, the following restriction applies:

[constr_1292] Limitation on the number of RPortComSpecs/PPortComSpecs in the context of one PRPortPrototype ⌈Within the context of one PRPortPrototype, there can only be **one** RPortComSpec and **one** PPortComSpec that references a given

- dataElement (in the case of ReceiverComSpec/SenderComSpec),
- operation (in the case of ClientComSpec/ServerComSpec),
- modeGroup (in the case of ModeSwitchReceiverComSpec/ModeSwitchSenderComSpec), or

- variable (in the case of NvRequireComSpec/NvProvideComSpec)

at the time when the contract phase generation is executed.⌋()

In other words, it is not allowed that two or more PPortComSpec exist in the context of a one PRPortPrototype that refer to the same dataElement or operation.

In the same manner, it is not allowed that two or more RPortComSpec exist in the context of one PRPortPrototype that refer to the same dataElement or operation.

The rationale for the existence of [constr_1290], [constr_1291], and [constr_1292] is that the AUTOSAR communication layer needs an unambiguous specification of the communication behavior.

The existence of redundant RPortComSpecs/PPortComSpecs may easily be contradicting each other and this would inhibit the creation of a valid configuration for the AUTOSAR Com.

As explained in section 2.5, there are cases where PortPrototypes owned by a CompositionSwComponentType could have initValues.

Therefore, it is possible that PortPrototypes owned by CompositionSwComponentTypes can have ComSpecs. It is *not* required that the ComSpecs defined on the composition level match the ComSpecs defined inside the CompositionSwComponentType.

If consistency would be required this constraint might be a major obstacle for integrating existing AtomicSwComponentTypes into a CompositionSwComponentType that has PortPrototypes with ComSpecs.

Class	***PPortComSpec*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of a provided PortPrototype. This class will contain attributes that are valid for all kinds of provide ports, independent of client-server or sender-receiver communication patterns.			
Base	*ARObject*			
Subclasses	ModeSwitchSenderComSpec, NvProvideComSpec, ParameterProvideComSpec, *SenderComSpec*, ServerComSpec			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.58: PPortComSpec

Class	***RPortComSpec*** (abstract)
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Communication attributes of a required PortPrototype. This class will contain attributes that are valid for all kinds of require-ports, independent of client-server or sender-receiver communication patterns.
Base	*ARObject*
Subclasses	ClientComSpec, ModeSwitchReceiverComSpec, NvRequireComSpec, ParameterRequireComSpec, *ReceiverComSpec*

▽

△

Class	***RPortComSpec*** (abstract)			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 4.59: RPortComSpec

[constr_1043] Allowed combinations of a specific *Type of* `PortInterface`, a specific *Type of* `PortPrototype`, and a specific *Type of* `ComSpec` ⌈

Type of PortPrototype	Type of ComSpec	Role of Element	Type of PortInterface	Role of Type-Ref
PPortPrototype	NonqueuedSender-ComSpec	dataElement	SenderReceiverIn-terface	providedInterface
PPortPrototype	QueuedSenderCom-Spec	dataElement	SenderReceiverIn-terface	providedInterface
RPortPrototype	Nonqueue-dReceiverComSpec	dataElement	SenderReceiverIn-terface	requiredInterface
RPortPrototype	QueuedReceiver-ComSpec	dataElement	SenderReceiverIn-terface	requiredInterface
PRPortPrototype	NonqueuedSender-ComSpec	dataElement	SenderReceiverIn-terface	providedRequired-Interface
PRPortPrototype	Nonqueue-dReceiverComSpec	dataElement	SenderReceiverIn-terface	providedRequired-Interface
PRPortPrototype	QueuedReceiver-ComSpec	dataElement	SenderReceiverIn-terface	providedRequired-Interface
PRPortPrototype	QueuedSenderCom-Spec	dataElement	SenderReceiverIn-terface	providedRequired-Interface
PPortPrototype	NvProvideComSpec	nvData	NvDataInterface	providedInterface
RPortPrototype	NvRequireComSpec	nvData	NvDataInterface	requiredInterface
PRPortPrototype	NvProvideComSpec	nvData	NvDataInterface	providedRequired-Interface
PRPortPrototype	NvRequireComSpec	nvData	NvDataInterface	providedRequired-Interface
PPortPrototype	ModeSwitchSender-ComSpec	modeGroup	ModeSwitchInter-face	providedInterface
RPortPrototype	ModeSwitchRe-ceiverComSpec	modeGroup	ModeSwitchInter-face	requiredInterface
PRPortPrototype	ModeSwitchSender-ComSpec	modeGroup	ModeSwitchInter-face	providedRequired-Interface
PRPortPrototype	ModeSwitchRe-ceiverComSpec	modeGroup	ModeSwitchInter-face	providedRequired-Interface
PPortPrototype	ParameterProvide-ComSpec	parameter	ParameterInter-face	providedInterface
RPortPrototype	ParameterRequire-ComSpec	parameter	ParameterInter-face	requiredInterface
PPortPrototype	ServerComSpec	operation	ClientServerIn-terface	providedInterface
RPortPrototype	ClientComSpec	operation	ClientServerIn-terface	requiredInterface

▽

△

PRPortPrototype	ServerComSpec	operation	ClientServerInterface	providedRequiredInterface
PRPortPrototype	ClientComSpec	operation	ClientServerInterface	providedRequiredInterface

This rule shall be imposed **at any time in the workflow**

⌋*()*

[constr_10372] Relation between *Type of* `PortPrototype`, *Type of* `ComSpec`, and *Type of* `PortInterface` ⌈With respect to [constr_1043], if a *Type of* *PortPrototype* aggregates a *Type of* *ComSpec*, then the *Type of* *PortPrototype* shall

- reference a *Type of* *PortInterface* in the role *Role of Type-Ref* and
- the *Role of Element* that is referenced from the *Type of* *ComSpec* shall be aggregated by the exact same *Type of* *PortInterface* that is also referenced by the enclosing *Type of* *PortPrototype* in the role *Role of Type-Ref*.

⌋*()*

4.5.1 Communication Specification for Sender-Receiver Communication

Communication specification applies in different ways to specific kinds of communication.

[TPS_SWCT_01219] `ComSpec` for queued and non-queued sender-receiver communication ⌈Sender-receiver communication might be queued or non-queued. This aspect is primarily reflected in the value of dataElement.swDataDefProps.swImplPolicy.

If the value of this attribute is set to queued then QueuedSenderComSpec and/or QueuedReceiverComSpec shall be defined. In all other applicable cases NonqueuedSenderComSpec or NonqueuedReceiverComSpec shall be used.

Thus, the constraints [constr_1129], [constr_1130], [constr_1131], and [constr_1132] shall apply.

While in the case of queued communication the queueLength attribute remains the only information item the non-queued case foresees several attributes for controlling communication behavior.⌋*()*

4.5.1.1 Receiver ComSpec

Figure 4.37 shows the meta-model of the communication attributes relevant sender-receiver communication at an RPortPrototype.

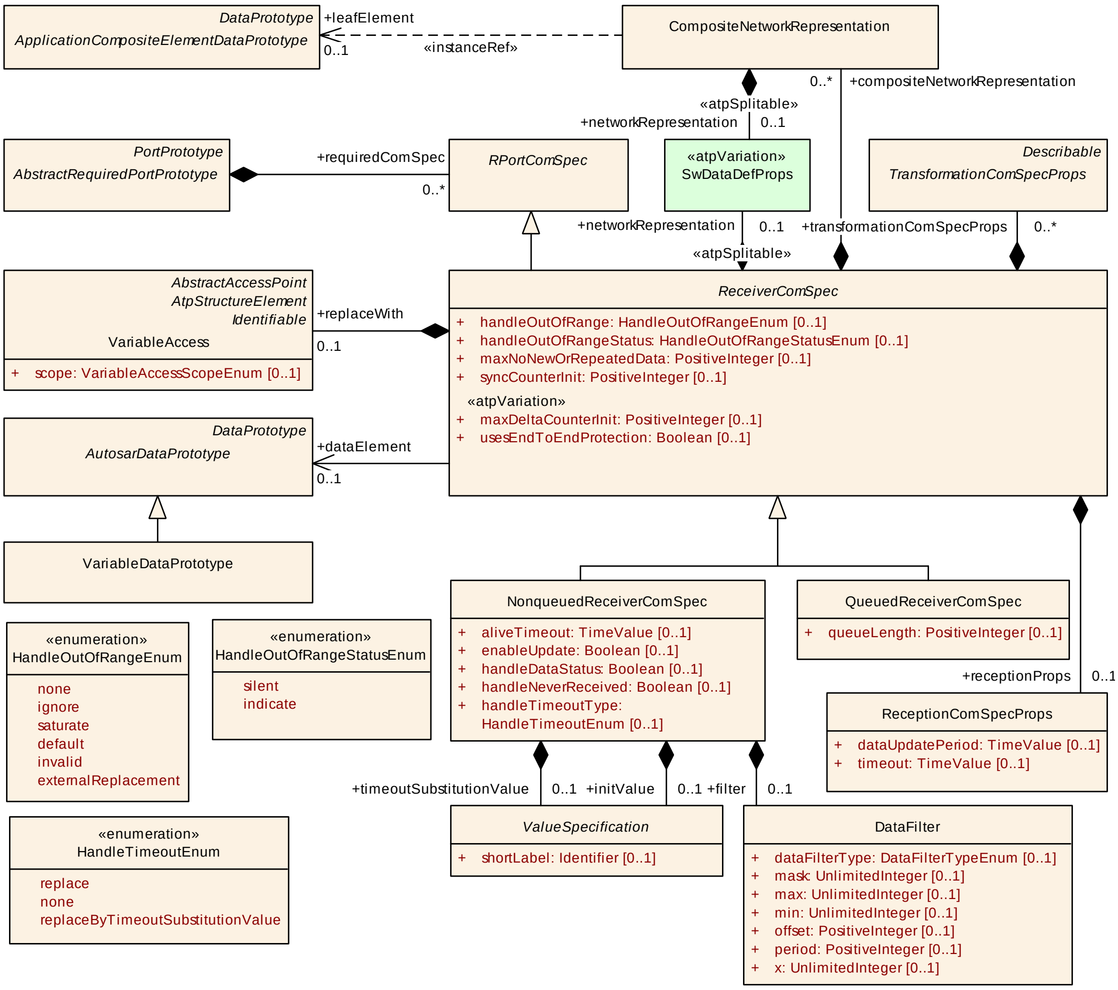


Figure 4.37: Communication attributes of RPortPrototype with respect to sender-receiver communication.

[constr_1538] Restriction for ReceiverComSpec.dataElement ⌈The reference ReceiverComSpec.dataElement **shall not** refer to an ArgumentDataPrototype or ParameterDataPrototype **at any time in the workflow**.⌋*()*

[constr_1103] NonqueuedReceiverComSpec and enableUpdate ⌈A NonqueuedReceiverComSpec that has the value of attribute enableUpdate set to true **at the time when the contract phase generation is executed** may not reference a dataElement that in turn is referenced by a VariableAccess in the role dataReadAccess.⌋*()*

In general, it is considered beneficial for software-components to define initValues for all the dataElements received by RPortPrototypes.

These initValues are required by the RTE for several functionalities, e.g. :

- Providing a default value for not yet received dataElements (see [TPS_SWCT_01220]).
- Providing default values in case of unconnected RPortPrototypes (see [constr_1100]).

- Partial mapping of composite data (see [constr_1280])

Therefore, the availability of initValue increases the flexibility of the usage of the software-component in different scenarios.

On the other hand, there are also use cases where initValues are not mandatory, i.e. the DataPrototype remains intentionally uninitialized. This is expressed by applying a SwAddrMethod where the sectionInitializationPolicy is set to NO-INIT, or when the software component is intentionally only prepared for intra-partition communication.

In response to these conflicting objectives [TPS_SWCT_01688] is written as a recommendation as opposed to a binding constraint.

[TPS_SWCT_01688] initValue should exist in an RPortPrototype ⌈The optional attribute initValue should exist if the enclosing NonqueuedReceiverComSpec is owned by an RPortPrototype.⌋*()*

[TPS_SWCT_01455] Duplicate existence of initValue in the context of a PRPortPrototype ⌈If an initValue is defined in a NonqueuedReceiverComSpec owned by a PRPortPrototype, its value shall be ignored.⌋*(RS_SWCT_03250)*

[TPS_SWCT_01220] initValue defines an initial value that shall be taken if the corresponding dataElement has not yet been received ⌈The aggregation of ValueSpecification in the role initValue defines an initial value that shall be taken if the corresponding dataElement has not yet been received but the application software is attempting to access its value.⌋*()*

[constr_1891] Existence of attribute NonqueuedReceiverComSpec.initValue ⌈For each NonqueuedReceiverComSpec, attribute initValue shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	***ReceiverComSpec*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Receiver-specific communication attributes (RPortPrototype typed by SenderReceiverInterface).			
Base	*ARObject*, *RPortComSpec*			
Subclasses	NonqueuedReceiverComSpec, QueuedReceiverComSpec			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compositeNetworkRepresentation	CompositeNetworkRepresentation	*	aggr	This represents a CompositeNetworkRepresentation defined in the context of a ReceiverComSpec. The purpose of this aggregation is to be able to specify the network representation of leaf elements of Application CompositeDataTypes.
dataElement	AutosarDataPrototype	0..1	ref	Data element these attributes belong to.
handleOutOfRange	HandleOutOfRangeEnum	0..1	attr	This attribute controls how values that are out of the specified range are handled according to the values of HandleOutOfRangeEnum.
handleOutOfRangeStatus	HandleOutOfRangeStatusEnum	0..1	attr	Control the way how return values are created in case of an out-of-range situation.

▽

△

Class	***ReceiverComSpec*** (abstract)			
maxDelta CounterInit	PositiveInteger	0..1	attr	Initial maximum allowed gap between two counter values of two consecutively received valid Data, i.e. how many subsequent lost data is accepted. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter Init is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
maxNoNewOr RepeatedData	PositiveInteger	0..1	attr	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.
network Representation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the data Element is mapped to a communication bus. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=networkRepresentation
receptionProps	ReceptionComSpec Props	0..1	aggr	"This aggregation represents the definition transmission props in the context of the enclosing ReceiverComSpec.
replaceWith	VariableAccess	0..1	aggr	This aggregation is used to identify the AutosarData Prototype to be taken for sourcing an external replacement in the out-of-range and invalidValue handling.
syncCounterInit	PositiveInteger	0..1	attr	Number of Data required for validating the consistency of the counter that shall be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.
transformation ComSpecProps	TransformationCom SpecProps	*	aggr	This references the TransformationComSpecProps which define port-specific configuration for data transformation.

▽

△

Class	*ReceiverComSpec* (abstract)			
usesEndToEnd Protection	Boolean	0..1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 4.60: ReceiverComSpec

Enumeration	HandleOutOfRangeStatusEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	This enumeration defines how the RTE handles values that are out of range.
Aggregated by	*ReceiverComSpec*.handleOutOfRangeStatus
Literal	***Description***
indicate	The RTE sets the return status to RTE_E_OUT_OF_RANGE if the received value is out of range and the attribute handleOutOfRange is not set to "none" or "invalid". **Tags:**atp.EnumerationLiteralIndex=0
silent	The RTE sets the return status to RTE_E_OK **Tags:**atp.EnumerationLiteralIndex=1

Table 4.61: HandleOutOfRangeStatusEnum

Class	NonqueuedReceiverComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to non-queued receiving.			
Base	*ARObject*, *RPortComSpec*, *ReceiverComSpec*			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
aliveTimeout	TimeValue	0..1	attr	Specify the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been received according to the specified timing description. If the aliveTimeout attribute is 0 no timeout monitoring shall be performed.
enableUpdate	Boolean	0..1	attr	This attribute controls whether application code is entitled to check whether the value of the corresponding Variable DataPrototype has been updated.
filter	DataFilter	0..1	aggr	The applicable filter algorithm for filtering the value of the corresponding dataElement.
handleData Status	Boolean	0..1	attr	If this attribute is set to true, then the Rte_IStatus API shall exist. If the attribute does not exist or is set to false, then the Rte_IStatus API may still exist in response to the existence of further conditions.

▽

△

Class	NonqueuedReceiverComSpec			
handleNever Received	Boolean	0..1	attr	This attribute specifies whether for the corresponding VariableDataPrototype the "never received" flag is available. If yes, the RTE is supposed to assume that initially the VariableDataPrototype has not been received before. After the first reception of the corresponding VariableDataPrototype the flag is cleared. • If the value of this attribute is set to "true" the flag is required. • If set to "false", the RTE shall not support the "never received" functionality for the corresponding VariableDataPrototype.
handleTimeout Type	HandleTimeoutEnum	0..1	attr	This attribute controls the behavior with respect to the handling of timeouts.
initValue	ValueSpecification	0..1	aggr	Initial value to be used in case the sending component is not yet initialized. If the sender also specifies an initial value, then the receiver's value will be used.
timeout Substitution Value	ValueSpecification	0..1	aggr	This attribute represents the substitution value applicable in the case of a timeout.

Table 4.62: NonqueuedReceiverComSpec

Class	**QueuedReceiverComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to queued receiving.			
Base	*ARObject*, *RPortComSpec*, *ReceiverComSpec*			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
queueLength	PositiveInteger	0..1	attr	Length of queue for received events.

Table 4.63: QueuedReceiverComSpec

[constr_1889] Existence of attribute QueuedReceiverComSpec.queueLength
⌈For each QueuedReceiverComSpec, attribute queueLength shall exist **at the time when the contract phase generation is executed.**⌋*()*

Class	**ReceptionComSpecProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	This meta-class defines a set of reception attributes which the application software is assumed to implement.			
Base	*ARObject*			
Aggregated by	*ReceiverComSpec*.receptionProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataUpdate Period	TimeValue	0..1	attr	This attribute defines the period in which the application shall check for updated data. This attribute is used for the configuration of the E2E protection, but may also indicate a general data reception period.

▽

△

Class	**ReceptionComSpecProps**			
timeout	TimeValue	0..1	attr	This attribute defines the time interval after which the application shall assume that the to be received data reception has timed out, i.e. the respective data has not been received for that amount of time.

Table 4.64: ReceptionComSpecProps

Enumeration	**HandleTimeoutEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Strategies of handling a reception timeout violation.
Aggregated by	NonqueuedReceiverComSpec.handleTimeoutType
Literal	***Description***
none	If set to none no replacement shall take place. **Tags:**atp.EnumerationLiteralIndex=0
replace	If set to replace, the replacement value shall be the ComInitValue. **Tags:**atp.EnumerationLiteralIndex=1
replaceByTimeout SubstitutionValue	If set to replace, the replacement value shall be the timeout substitution value. **Tags:**atp.EnumerationLiteralIndex=2

Table 4.65: HandleTimeoutEnum

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE xml.xsd.type=double

Table 4.66: TimeValue

[constr_1129] **swImplPolicy and NonqueuedReceiverComSpec** ⌈The attribute swImplPolicy of a dataElement referenced by a NonqueuedReceiverComSpec **shall not** be set to the value queued **at the time when the contract phase generation is executed**.⌋*()*

[constr_1130] **swImplPolicy and QueuedReceiverComSpec** ⌈The attribute swImplPolicy of a dataElement referenced by a QueuedReceiverComSpec **shall** be set to the value queued **at the time when the contract phase generation is executed**.⌋*()*

[constr_1188] Existence of ReceiverComSpec.replaceWith ⌈The aggregation of VariableAccess in the role ReceiverComSpec.replaceWith shall exist **if and only if at least one of the following conditions is fulfilled**:

- Attribute ReceiverComSpec.handleOutOfRange is set to the value externalReplacement.

- Attribute SenderReceiverInterface.invalidationPolicy.handleInvalid is set to the value externalReplacement.

This rule shall be applied **at the time when the contract phase generation is executed.**⌋()

[TPS_SWCT_01753] Application of compatibility rules for ReceiverComSpec.replaceWith ⌈Compatibility rules as formulated by [constr_1068] and [constr_1187] shall be applicable for the reference ReceiverComSpec.replaceWith.⌋()

[TPS_SWCT_01223] networkRepresentation defines how a specific dataElement is represented on a communication bus ⌈For sender-receiver communication, it is possible to specify how dataElements are represented given that the communication requires the usage of a dedicated communication bus.

That is, by means of the networkRepresentation it is possible to define how a specific dataElement is represented on a communication bus. For this purpose the networkRepresentation is implemented as an aggregation of SwDataDefProps.⌋()

[TPS_SWCT_01224] CompuMethods of dataElement and the networkRepresentation are used for conversion purposes ⌈The attached CompuMethods of both the dataElement and the networkRepresentation can be used to identify the conversion between the two.

The advantage of this approach is that this can also be used without any modifications in combination with a general remapping and rescaling of dataElements between different SwComponentTypes, regardless whether they are located on the same or on different ECUs.⌋()

Please note that the decision whether to take the networkRepresentation for data mapping is done in the context of the AUTOSAR System Template [10]. Please find more detailed information about this aspect in the applicable specification.

[TPS_SWCT_01452] Applicability of networkRepresentation for ApplicationCompositeDataType ⌈The aggregation of networkRepresentation at the ReceiverComSpec or SenderComSpec only applies for dataElements typed by ApplicationPrimitiveDataTypes.

For the case of using an ApplicationCompositeDataType an additional mechanism shall be used.

In particular, compositeNetworkRepresentation shall be used to define the networkRepresentation of **leaf elements** of ApplicationCompositeDataTypes.⌋()

[constr_1196] Existence of networkRepresentation vs. compositeNetworkRepresentation ⌈If a ReceiverComSpec or SenderComSpec aggregates networkRepresentation it shall **not** aggregate compositeNetworkRepresentation (and vice versa) **at the time when the contract phase generation is executed.**⌋()

[constr_1197] Existence of compositeNetworkRepresentation shall be comprehensive ⌈If at least one compositeNetworkRepresentation exists then for each leaf ApplicationCompositeElementDataPrototype of the affected ApplicationCompositeDataType exactly one compositeNetworkRepresentation shall be defined **at the time when the contract phase generation is executed**.

For each such compositeNetworkRepresentation, attributes leafElement and networkRepresentation shall exist **at the time when the contract phase generation is executed**.⌋*()*

Granted, the definition of [constr_1197] to some extent has a recursive character. The meaning is that if it is actually intended to define a compositeNetworkRepresentation then the definition shall be completely covering the entire set of leaf elements of the corresponding ApplicationCompositeDataType. In other words, it's all or nothing.

[TPS_SWCT_01593] Semantics of attribute ReceiverComSpec.transformationComSpecProps ⌈The ReceiverComSpec.transformationComSpecProps is used to configure PortPrototype-specific properties for data transformation in case of receiving inter-ECU communication.⌋*()*

[TPS_SWCT_01682] The meaning of E2E-related attributes in a ReceiverComSpec if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined. ⌈The attributes usesEndToEndProtection, syncCounterInit, maxDeltaCounterInit, and maxNoNewOrRepeatedData in ReceiverComSpec have no meaning if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined in the same ReceiverComSpec.⌋*()*

4.5.1.2 Sender ComSpec

The communication attributes on the sender side are sketched in Figure 4.38.

[constr_1131] swImplPolicy and NonqueuedSenderComSpec ⌈The attribute swImplPolicy of a dataElement referenced by a NonqueuedSenderComSpec **shall not** be set to the value queued **at the time when the contract phase generation is executed**.⌋*()*

[constr_1132] swImplPolicy and QueuedSenderComSpec ⌈The attribute swImplPolicy of a dataElement referenced by a QueuedSenderComSpec **shall** be set to the value queued **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01751] The meaning of E2E-related attributes in a SenderComSpec if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined ⌈The attribute usesEndToEndProtection has no meaning if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined in the same SenderComSpec.⌋*()*

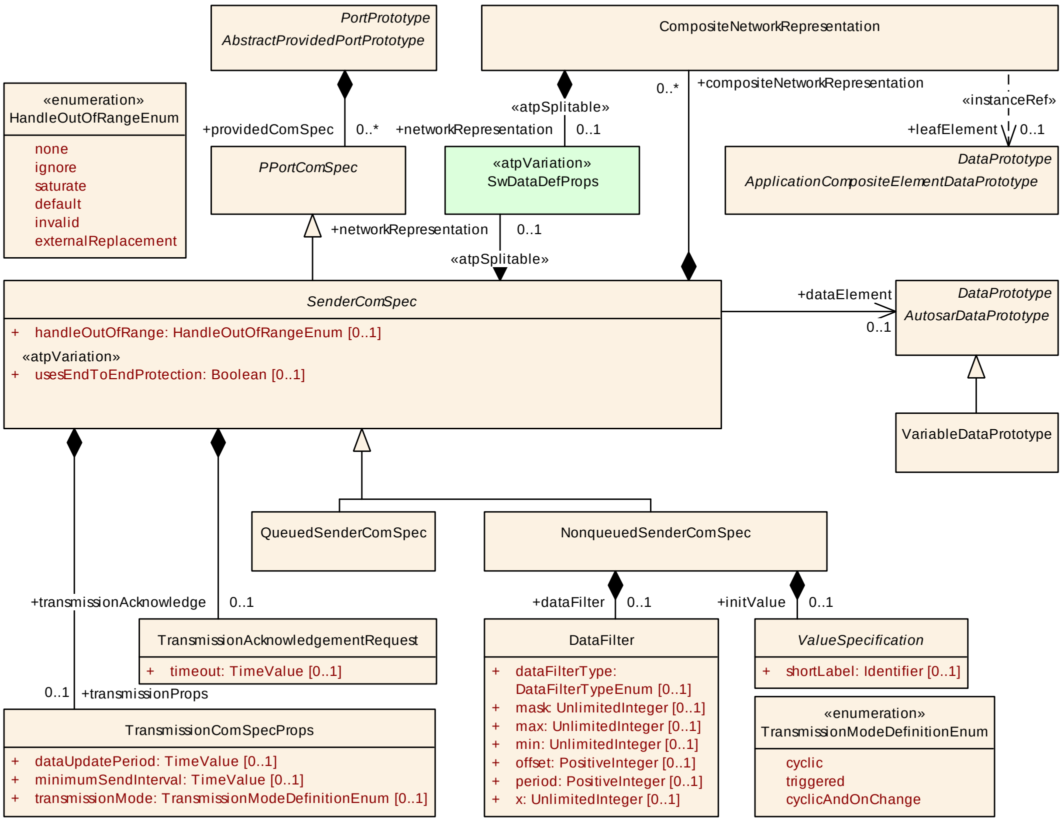


Figure 4.38: Communication attributes of PPortPrototype with respect to sender-receiver communication.

Please note:

- SenderComSpec.usesEndToEndProtection does not have any influence on code generation.

  It could be used, for example, by a validation framework to make sure that, if set to true the dataElement meets a transformer configuration for all respective SwConnectors connecting to the PortPrototype that owns the SenderComSpec.

- SenderComSpec.usesEndToEndProtection could be used as a statement from the application developer that the given dataElement shall be end-to-end protected.

  However, it seems far-fetched for an application developer to expressly state that a dataElement shall **not** be end-to-end protected. This goes beyond the responsibility of an application developer.

Therefore, two relevant states for SenderComSpec.usesEndToEndProtection can be expected:

  - attribute exists and is set to true (application developer asserts the necessity to end-to-end protect the dataElement)
  - attribute does not exist (application developer doesn't care)

- The application developer may not have enough oversight to envision how the dataElement is communicated, i.e. local vs. network communication. Setting usesEndToEndProtection to true and then deploy the enclosing software-component such that it communicates only locally on the respective PortPrototype also seems unusual for the current situation regarding transformer-based communication.

[constr_1539] Restriction for SenderComSpec.dataElement ⌈The reference SenderComSpec.dataElement **shall not** refer to an ArgumentDataPrototype or ParameterDataPrototype **at the time when the contract phase generation is executed.**⌋*()*

Class	***SenderComSpec*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a sender port (PPortPrototype typed by SenderReceiverInterface).			
Base	*ARObject*, *PPortComSpec*			
Subclasses	NonqueuedSenderComSpec, QueuedSenderComSpec			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
compositeNetworkRepresentation	CompositeNetworkRepresentation	*	aggr	This represents a CompositeNetworkRepresentation defined in the context of a SenderComSpec.
dataElement	AutosarDataPrototype	0..1	ref	Data element these quality of service attributes apply to.
handleOutOfRange	HandleOutOfRangeEnum	0..1	attr	This attribute controls how out-of-range values shall be dealt with.
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=networkRepresentation
transmissionAcknowledge	TransmissionAcknowledgementRequest	0..1	aggr	Requested transmission acknowledgement for data element.
transmissionProps	TransmissionComSpecProps	0..1	aggr	This aggregation represents the definition transmission props in the context of the enclosing SenderComSpec.
usesEndToEndProtection	Boolean	0..1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 4.67: SenderComSpec

[TPS_SWCT_01820] Existence of attribute SenderComSpec.handleOutOfRange ⌈If attribute SenderComSpec.handleOutOfRange does not exist at the time when the RTE is generated then value none shall be assumed.⌋*()*

Class	**QueuedSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to distribution of events (PPortPrototype, SenderReceiverInterface and dataElement carries an "event").			
Base	*ARObject*, *PPortComSpec*, *SenderComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.68: QueuedSenderComSpec

Class	**NonqueuedSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for non-queued sender/receiver communication (sender side)			
Base	*ARObject*, *PPortComSpec*, *SenderComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataFilter	DataFilter	0..1	aggr	The applicable filter algorithm for filtering the value of the corresponding dataElement.
initValue	ValueSpecification	0..1	aggr	Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.

Table 4.69: NonqueuedSenderComSpec

Class	**TransmissionComSpecProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	This meta-class defines a set of transmission attributes which the application software is assumed to implement.			
Base	*ARObject*			
Aggregated by	*SenderComSpec*.transmissionProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataUpdate Period	TimeValue	0..1	attr	This attribute defines the period in which the application is assumed to transmit the respective data.
minimumSend Interval	TimeValue	0..1	attr	This attribute defines the minimum interval between two consecutive transmissions of the respective data the application is assumed to ensure.
transmission Mode	TransmissionMode DefinitionEnum	0..1	attr	The attribute defines the mode in which the application is assumed to transmit the respective data.

Table 4.70: TransmissionComSpecProps

Class	**TransmissionAcknowledgementRequest**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests transmission acknowledgement that data has been sent successfully. Success/failure is reported via a SendPoint of a RunnableEntity.			
Base	*ARObject*			
Aggregated by	*SenderComSpec*.transmissionAcknowledge			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
timeout	TimeValue	0..1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table 4.71: TransmissionAcknowledgementRequest

[constr_1892] Existence of attribute TransmissionAcknowledgementRequest.timeout ⌈For each TransmissionAcknowledgementRequest, attribute timeout shall exist **at the time when the contract phase generation is executed.**⌋()

Enumeration	**HandleOutOfRangeEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	A value of this type is taken for controlling the range checking behavior of the AUTOSAR RTE.
Aggregated by	ISignalProps.handleOutOfRange, *ReceiverComSpec*.handleOutOfRange, *SenderComSpec*.handleOutOfRange
Literal	***Description***
default	The RTE will use the initValue if the actual value is out of the specified bounds. **Tags:**atp.EnumerationLiteralIndex=0
externalReplacement	This indicates that the value replacement is sourced from the attribute replaceWith. **Tags:**atp.EnumerationLiteralIndex=1
ignore	The RTE will ignore any attempt to send or receive the corresponding dataElement if the value is out of the specified range. **Tags:**atp.EnumerationLiteralIndex=2
invalid	The RTE will use the invalidValue if the value is out of the specified bounds. **Tags:**atp.EnumerationLiteralIndex=3
none	A range check is not required. **Tags:**atp.EnumerationLiteralIndex=4
saturate	The RTE will saturate the value of the dataElement such that it is limited to the applicable upper bound if it is greater than the upper bound. Consequently, it is limited to the applicable lower bound if the value is less than the lower bound. **Tags:**atp.EnumerationLiteralIndex=5

Table 4.72: HandleOutOfRangeEnum

Enumeration	**TransmissionModeDefinitionEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	This meta-class defines possible settings for the transmission mode.
Aggregated by	TransmissionComSpecProps.transmissionMode
Literal	***Description***
cyclic	The data is assumed to be transmitted in a cyclic manner. The cycle is defined by dataUpdatePeriod. **Tags:**atp.EnumerationLiteralIndex=0
cyclicAndOnChange	The data is assumed to be transmitted in a cyclic manner (with cycle time dataUpdatePeriod) and additionally there may be arbitrary transmission if the data value changes (minimumSendInterval to be respected, if defined). **Tags:**atp.EnumerationLiteralIndex=2
triggered	The data is assumed to be transmitted in an arbitrary manner (minimumSendInterval to be respected, if defined). **Tags:**atp.EnumerationLiteralIndex=1

Table 4.73: TransmissionModeDefinitionEnum

Class	**CompositeNetworkRepresentation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	This meta-class is used to define the network representation of leaf elements of composite application data types.			
Base	*ARObject*			
Aggregated by	*ReceiverComSpec*.compositeNetworkRepresentation, *SenderComSpec*.compositeNetworkRepresentation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
leafElement	ApplicationCompositeElementDataPrototype	0..1	iref	This represents that leaf element of an application composite data type. **InstanceRef implemented by:**ApplicationCompositeElementInPortInterfaceInstanceRef
networkRepresentation	SwDataDefProps	0..1	aggr	The SwDataDefProps owned by the CompositeNetworkRepresentation are used to define the network representation of the leaf element of an ApplicationCompositeDataType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=networkRepresentation

Table 4.74: CompositeNetworkRepresentation

4.5.1.3 Data Filter

Figure 4.39 shows the model of the communication attributes relevant for defining data filters.

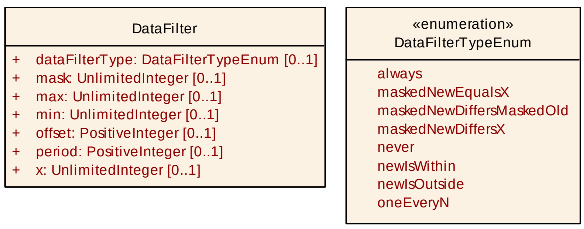


Figure 4.39: `DataFilter` and its communication attributes.

[TPS_SWCT_01221] `DataFilter` ⌈For every `RPortPrototype` or `PPortPrototype` typed by a `SenderReceiverInterface`, a `DataFilter` can be defined given that non-queued communication is foreseen.⌋*()*

Several filter algorithms formally described by the enumeration type `DataFilterTypeEnum` in the meta-model are taken from the ISO 17356-4 specification [17] that is referenced by the RTE specification [2].

[TPS_SWCT_01222] Applicability of `DataFilter` ⌈This ISO 17356-4 specification states that "filtering is only used for messages that can be interpreted as C language unsigned integer types (characters, unsigned integers and enumerations)."⌋*(RS_SWCT_03221)*

[constr_1044] Applicability of `DataFilter` ⌈According to the origin of `DataFilter`, i.e. ISO 17356-4 specification [17], `DataFilter`s can only be applied to values with an integer base type **`at any time in the workflow`**.⌋*()*

Class	**DataFilter**			
Package	M2::AUTOSARTemplates::CommonStructure::Filter			
Note	Base class for data filters. The type of the filter is specified in attribute dataFilterType. Some of the filter types require additional arguments which are specified as attributes of this class.			
Base	*ARObject*			
Aggregated by	ISignalPort.dataFilter, NonqueuedReceiverComSpec.filter, NonqueuedSenderComSpec.dataFilter, SignalBasedEventElementToISignalTriggeringMapping.filter, SignalBasedFieldToISignalTriggering Mapping.filter, SignalServiceTranslationElementProps.filter, TransmissionModeCondition.dataFilter			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataFilterType	DataFilterTypeEnum	0..1	attr	This attribute specifies the type of the filter.
mask	UnlimitedInteger	0..1	attr	Mask for old and new value.
max	UnlimitedInteger	0..1	attr	Value to specify the upper boundary
min	UnlimitedInteger	0..1	attr	Value to specify the lower boundary
offset	PositiveInteger	0..1	attr	Specifies the initial number of messages to occur before the first message is passed
period	PositiveInteger	0..1	attr	Specifies number of messages to occur before the message is passed again
x	UnlimitedInteger	0..1	attr	Value to compare with

Table 4.75: DataFilter

[constr_1890] Existence of attribute `DataFilter.dataFilterType` ⌈For each `DataFilter`, attribute `dataFilterType` shall exist **`at the time when the RTE is generated`**.⌋*()*

Enumeration	**DataFilterTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::Filter
Note	This enum specifies the supported DataFilterTypes.
Aggregated by	DataFilter.dataFilterType
Literal	***Description***
always	No filtering is performed so that the message always passes. **Tags:**atp.EnumerationLiteralIndex=0
maskedNewDiffers MaskedOld	Pass messages where the masked value has changed. (new_value&mask) !=(old_value&mask) new_value: current value of the message old_value: last value of the message (initialized with the initial value of the message, updated with new_value if the new message value is not filtered out) **Tags:**atp.EnumerationLiteralIndex=1
maskedNewDiffers X	Pass messages whose masked value is not equal to a specific value x (new_value&mask) != x new_value: current value of the message **Tags:**atp.EnumerationLiteralIndex=2
maskedNewEquals X	Pass messages whose masked value is equal to a specific value x (new_value&mask) == x new_value: current value of the message **Tags:**atp.EnumerationLiteralIndex=3

▽

△

Enumeration	**DataFilterTypeEnum**
never	The filter removes all messages. **Tags:**atp.EnumerationLiteralIndex=4
newIsOutside	Pass a message if its value is outside a predefined boundary. (min > new_value) OR (new_value > max) **Tags:**atp.EnumerationLiteralIndex=5
newIsWithin	Pass a message if its value is within a predefined boundary. min <= new_value <= max **Tags:**atp.EnumerationLiteralIndex=6
oneEveryN	Pass a message once every N message occurrences. Algorithm: occurrence % period == offset Start: occurrence = 0. Each time the message is received or transmitted, occurrence is incremented by 1 after filtering. Length of occurrence is 8 bit (minimum). **Tags:**atp.EnumerationLiteralIndex=7

Table 4.76: DataFilterTypeEnum

4.5.1.4 Communication between Application and NV Block Software Components

In many cases, communication between an NvBlockSwComponentType (see section 11.4.2) and an ApplicationSwComponentType is implemented based on mixed communication approaches, i.e. the ApplicationSwComponentType most likely uses a PortPrototype typed by a SenderReceiverInterface while the NvBlockSwComponentType uses a PortPrototype typed by an NvDataInterface.

[constr_10071] Allowed multiplicities of SenderComSpec attributes for communication between ApplicationSwComponentType and NvBlockSwComponentType ⌈

Sender	ApplicationSwComponentType	
Receiver	NvBlockSwComponentType	
Queuing Configuration	**non-queued**	**queued**
SenderComSpec.transmissionAcknowledge	d/c	n/a
SenderComSpec.dataElement	1	n/a
SenderComSpec.handleOutOfRange	d/c	n/a
SenderComSpec.usesEndToEndProtection	d/c	n/a
SenderComSpec.transmissionProps.dataUpdatePeriod	0..1	n/a
SenderComSpec.transmissionProps.minimumSendInterval	0..1	n/a
SenderComSpec.transmissionProps.transmissionMode	0..1	n/a
SenderComSpec.networkRepresentation	d/c	n/a
SenderComSpec.compositeNetworkRepresentation	d/c	n/a

▽

△

Sender	ApplicationSwComponentType	
Receiver	NvBlockSwComponentType	
Queuing Configuration	non-queued	queued
NonqueuedSenderComSpec.dataFilter	d/c	n/a
NonqueuedSenderComSpec.initValue	0..1	n/a

This rule shall be imposed at the time when the RTE is generated.

⌋()

In such a case it is necessary to clarify the usage of attributes of the respective ComSpec on both the sender and receiver side. For this purpose, the constraints [constr_10071] and [constr_10072] are defined[10].

Please note that the abbreviation “d/c”, as mentioned in [constr_10071], stands for “don’t care”.

[constr_10072] Allowed multiplicities of SenderComSpec attributes for communication between NvBlockSwComponentType and ApplicationSwComponentType ⌈

Sender	NvBlockSwComponentType	
Receiver	ApplicationSwComponentType	
Queuing Configuration	non-queued	queued
ReceiverComSpec.replaceWith	0	n/a
ReceiverComSpec.dataElement	1	n/a
ReceiverComSpec.receptionProps.dataUpdatePeriod	0	n/a
ReceiverComSpec.receptionProps.timeout	0	n/a
ReceiverComSpec.usesEndToEndProtection	0	n/a
ReceiverComSpec.maxDeltaCounterInit	0	n/a
ReceiverComSpec.handleOutOfRange	0	n/a
ReceiverComSpec.handleOutOfRangeStatus	0	n/a
ReceiverComSpec.maxNoNewOrRepeatedData	0	n/a
ReceiverComSpec.syncCounterInit	0	n/a
ReceiverComSpec.transformationComSpecProps	0	n/a
ReceiverComSpec.networkRepresentation	0	n/a
ReceiverComSpec.compositeNetworkRepresentation	0	n/a
QueuedReceiverComSpec.queueLength	n/a	n/a
NonqueuedReceiverComSpec.filter	0	n/a
NonqueuedReceiverComSpec.timeoutSubstitutionValue	0	n/a
NonqueuedReceiverComSpec.initValue	0..1	n/a
NonqueuedReceiverComSpec.aliveTimeout	0	n/a
NonqueuedReceiverComSpec.enableUpdate	0	n/a

▽

[10] As a general rule, queued communication between ApplicationSwComponentType and NvBlockSwComponentType is not supported by AUTOSAR. The [constr_10071] and [constr_10072] nevertheless contain a column that represents the case of queued communication for the sake of completeness. But consequently, this column is consistently filled with “n/a” entries in the respective tables.

△

Sender	NvBlockSwComponentType	
Receiver	ApplicationSwComponentType	
Queuing Configuration	non-queued	queued
NonqueuedReceiverComSpec.handleDataStatus	0	n/a
NonqueuedReceiverComSpec.handleNeverReceived	0..1	n/a
NonqueuedReceiverComSpec.handleTimeoutType	0	n/a

This rule shall be imposed **at the time when the RTE is generated**.

⌋*()*

4.5.1.5 Communication behavior to be implemented by the Software Component

AUTOSAR supports the handling of periodic data transmission and reception timeout checking in the Classic AUTOSAR Basic Software layer. The definition of the transmission modes and time values is defined using the TPS_SystemTemplate [10].

The only value which is also available at the NonqueuedReceiverComSpec is the aliveTimeout - as this value configures both communication stack timeout handling (in case of remote communication) and RTE timeout handling (in case of local communication).

With the introduction of the transformer technology the aspects of periodic data transmission and checking for timeout on reception may have to be implemented by the application code. The main reason is that the E2E-Transformer needs to be called periodically in order to keep its state machine up to date.

So, the application code needs to call the transmission / reception APIs periodically in order to fulfill these timing requirements. But there may also be other reasons why the application shall take care of the periodicity, for example in case the LdCom module (which doesn't have any timing features) is used.

The meta-classes TransmissionComSpecProps and ReceptionComSpecProps have been introduced to define the expected communication behavior to be implemented by the application code.

As the TransmissionComSpecProps and ReceptionComSpecProps define what the expected communication behavior is, the values can also be utilized by communication (network) measurement tools to verify whether the application code actually implements the attributes properly.

The attribute ReceptionComSpecProps.dataUpdatePeriod defines the time period in which the receiving application shall call the reception API to check for new data.

The attribute ReceptionComSpecProps.timeout defines the time after which the application shall assume that the to-be-received data reception has timed out.

The attribute NonqueuedReceiverComSpec.aliveTimeout might be defined as well, resulting in duplicate timeout checks.

The communication stack / RTE would check for the value of attribute aliveTimeout and the application code would check for the ReceptionComSpecProps.timeout. Although it does not seem sensible to perform a timeout check twice, it is not forbidden.

The attribute TransmissionComSpecProps.dataUpdatePeriod defines the time period in which the sending application shall call the send API.

The attributes TransmissionComSpecProps.minimumSendInterval and TransmissionComSpecProps.transmissionMode define values which influence the transmission behavior, implemented by the application code.

Note that the communication stack might also have a periodicity, minimum sending time and transmission mode defined. And such doubled implementation might lead to undesired effects. However AUTOSAR does not regulate the usage of the mechanisms to be exclusive.

4.5.2 Communication Specification for Client-Server Communication

4.5.2.1 Client ComSpec

The communication aspects relevant for client communication are sketched in Figure 4.40.

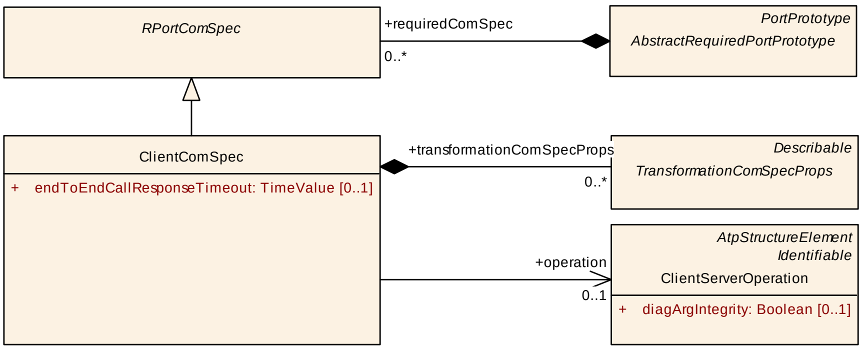


Figure 4.40: Communication attributes of RPortPrototype with respect to client-server communication.

[constr_1540] Existence of ClientComSpec.operation ⌈The reference ClientComSpec.operation **shall exist** if the AbstractRequiredPortPrototype that owns the ClientComSpec is typed by a ClientServerInterface. This rule shall be imposed **at any time in the workflow**.⌋()

Class	ClientComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Client-specific communication attributes (RPortPrototype typed by ClientServerInterface).			
Base	*ARObject*, *RPortComSpec*			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
endToEndCallResponseTimeout	TimeValue	0..1	attr	This attribute defines the maximum time interval in which the application shall expect the servers's response (time between the sending of the call invocation until the arrival of the server's response).
operation	ClientServerOperation	0..1	ref	This represents the corresponding ClientServerOperation.
transformationComSpecProps	TransformationComSpecProps	*	aggr	This references the TransformationComSpecProps which define port-specific configuration for data transformation.

Table 4.77: ClientComSpec

Note: on the *AUTOSAR adaptive platform* the ClientComSpec can also be used in the context of RPortPrototypes typed by PortInterfaces that are not available on the *AUTOSAR classic platform*. This is the motivation for the existence of [constr_1540].

[TPS_SWCT_01595] Semantics of attribute ClientComSpec.transformationComSpecProps ⌈The attribute ClientComSpec.transformationComSpecProps shall be used to configure PortPrototype-specific properties for data transformation in case of Client/Server inter-ECU communication for the reception of the server's response.⌋*(RS_SWCT_03221)*

4.5.2.2 Server ComSpec

The server side looks very similar but provides an attribute for specifying the queue length.

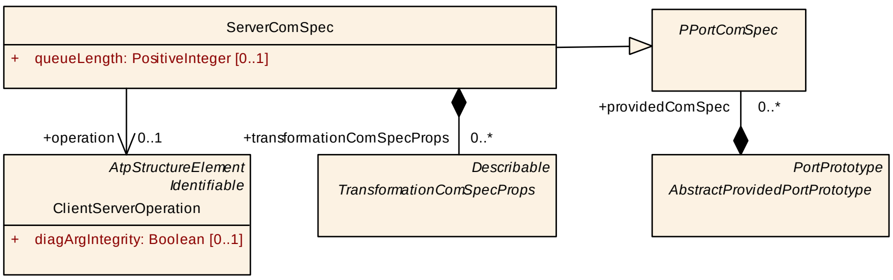


Figure 4.41: Communication attributes of PPortPrototype with respect to client-server communication.

[constr_1541] Existence of ServerComSpec.operation ⌈The reference ServerComSpec.operation **shall exist** if the AbstractProvidedPortPrototype that owns the ServerComSpec is typed by a ClientServerInterface.

This rule shall be imposed **at any time in the workflow**.⌋()

Note: on the *AUTOSAR adaptive platform* the ServerComSpec can also be used in the context of RPortPrototypes typed by PortInterfaces that are not available on the *AUTOSAR classic platform*. This is the motivation for the existence of [constr_1541].

Class	**ServerComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a server port (PPortPrototype and ClientServerInterface).			
Base	*ARObject*, *PPortComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operation	ClientServerOperation	0..1	ref	Operation these communication attributes apply to.
queueLength	PositiveInteger	0..1	attr	Length of call queue on the server side. The queue is implemented by the RTE. The value shall be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.
transformationComSpecProps	TransformationComSpecProps	*	aggr	This references the TransformationComSpecProps which define port-specific configuration for data transformation.

Table 4.78: ServerComSpec

[constr_1893] Existence of attribute ServerComSpec.queueLength ⌈For each ServerComSpec, attribute queueLength shall exist **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01225] RunnableEntity implements the functionality of more than one ClientServerOperations ⌈A single RunnableEntity can implement the functionality of more than one ClientServerOperations.

For this purpose, one OperationInvokedEvent for each affected ClientServerOperation shall reference the respective RunnableEntity.

The attribute ServerComSpec.queueLength shall be taken for the determination of the resulting queue length, [constr_1128] applies.⌋*()*

For more information about requirements towards ClientServerOperations triggering the same RunnableEntity can be found in section 7.2.4.3, specifically [constr_2000].

[constr_1128] Queue length of ClientServerOperations associated with the same RunnableEntity ⌈If two or more OperationInvokedEvents reference a single RunnableEntity the value of the ServerComSpec attribute queueLength shall be **identical** for all ServerComSpecs owned by PPortPrototypes of the enclosing SwComponentType that reference one of the ClientServerOperations that are also referenced by the OperationInvokedEvents.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01596] Semantics of attribute ServerComSpec.transformationComSpecProps ⌈The attribute ServerComSpec.transformationComSpecProps shall be used to configure PortPrototype-specific properties for data transformation in case of Client/Server inter-ECU communication for the reception of the client's request.⌋*(RS_SWCT_03221)*

See chapter 4.5.6 for details.

4.5.3 Communication Specification for Mode Switch Communication

4.5.3.1 Mode Switch Sender ComSpec

In analogy to the previous section, Figure 4.42 shows the meta-model elements relevant for a mode switch communication.

On the sender side it is possible to specify that an acknowledgement is supposed to be returned that indicates the successful processing of the mode switch request.

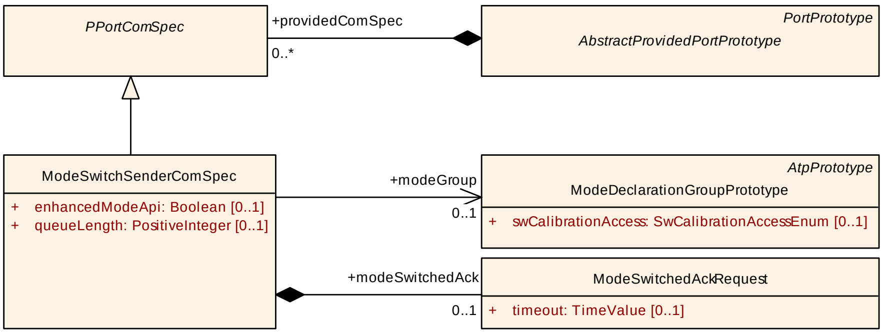


Figure 4.42: Communication attributes of PPortPrototype with respect to mode switch communication.

Class	**ModeSwitchSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of PPortPrototypes with respect to mode communication			
Base	*ARObject*, *PPortComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
enhancedModeApi	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to "true" the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
modeGroup	ModeDeclarationGroupPrototype	0..1	ref	ModeDeclarationGroupPrototype (of the same Port Interface) to which these communication attributes apply.
modeSwitchedAck	ModeSwitchedAckRequest	0..1	aggr	If this aggregation exists an acknowledgement for the successful processing of the mode switch request is required.

▽

△

Class	ModeSwitchSenderComSpec			
queueLength	PositiveInteger	0..1	attr	Length of call queue on the mode user side. The queue is implemented by the RTE. The value shall be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.

Table 4.79: ModeSwitchSenderComSpec

Class	**ModeSwitchedAckRequest**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests acknowledgements that a mode switch has been proceeded successfully			
Base	*ARObject*			
Aggregated by	ModeSwitchSenderComSpec.modeSwitchedAck			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
timeout	TimeValue	0..1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table 4.80: ModeSwitchedAckRequest

[constr_1894] Existence of attribute `ModeSwitchSenderComSpec.queueLength` ⌈For each `ModeSwitchSenderComSpec`, attribute `queueLength` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1895] Existence of attribute `ModeSwitchSenderComSpec.modeGroup` ⌈For each `ModeSwitchSenderComSpec`, attribute `modeGroup` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

4.5.3.2 Mode Switch Receiver ComSpec

[TPS_SWCT_01514] Duplicate existence of `enhancedModeApi` in the context of a `PRPortPrototype` ⌈If the attribute `enhancedModeApi` is defined in a `ModeSwitchReceiverComSpec` owned by a `PRPortPrototype`, its value shall be ignored.⌋*(RS_SWCT_03250)*

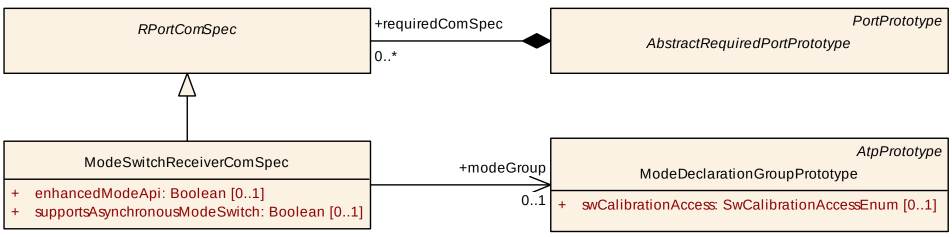


Figure 4.43: Communication attributes of `PPortPrototype` with respect to mode switch communication.

Class	**ModeSwitchReceiverComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to mode communication			
Base	*ARObject*, *RPortComSpec*			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
enhancedMode Api	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to "true" the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
modeGroup	ModeDeclarationGroup Prototype	0..1	ref	ModeDeclarationGroupPrototype (of the same Port Interface) to which these communication attributes apply.
supports Asynchronous ModeSwitch	Boolean	0..1	attr	This attribute controls the behavior of the corresponding RPortPrototype with respect to the question whether it can deal with asynchronous mode switch requests, i.e. if set to true, the RPortPrototype is able to deal with an asynchronous mode switch request.

Table 4.81: ModeSwitchReceiverComSpec

[constr_1896] Existence of attribute `ModeSwitchReceiverComSpec.modeGroup` ⌈For each `ModeSwitchReceiverComSpec`, attribute `modeGroup` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

4.5.4 Communication Specification for Parameters

Granted, the definition of a `ComSpec` for `ParameterDataPrototype`s looks strange on first sight. A `ParameterDataPrototype` owned by a `PPortPrototype` typed by a `ParameterInterface` is not actually transmitted over any communication medium. Therefore, the term *communication* should in this case be taken with a grain of salt.

However, it is generally necessary to be able to define role-specific initial values for `ParameterDataPrototype`s aggregated in a `ParameterInterface`. In other words, the actual problem closely resembles the definition of initial values in the case of sender-receiver communication.

[TPS_SWCT_01226] `initValue` on the level of a `ComSpec` is relevant for connections to the corresponding `PortPrototype` ⌈Please note that (along the example of sender-receiver communication) only the `initValue` defined in the context of a `ParameterProvideComSpec` or `ParameterRequireComSpec` is relevant for connections to the corresponding `PortPrototype`. An `initValue` defined in the scope of a `ParameterDataPrototype` is ignored.⌋*()*

Therefore, it is only reasonable to apply the existing and well-known pattern to the definition of initial values for `ParameterDataPrototype`s aggregated in a `ParameterInterface`. The actual modeling is sketched in Figure 4.44 for provided `ParameterDataPrototype`s and in Figure 4.45 for required `ParameterDataPrototype`s.

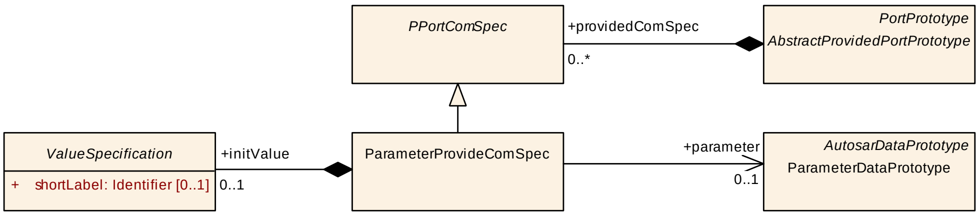


Figure 4.44: Communication attributes of ParameterDataPrototypes with respect to PPortPrototype

Class	**ParameterProvideComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the provided side of a connection.			
Base	*ARObject*, *PPortComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	0..1	ref	The ParameterDataPrototype to which the ParameterComSpec applies.

Table 4.82: ParameterProvideComSpec

[constr_1897] Existence of reference ParameterProvideComSpec.parameter ⌈For each ParameterProvideComSpec, the reference parameter shall exist **at the time when the contract phase generation is executed**.⌋*()*

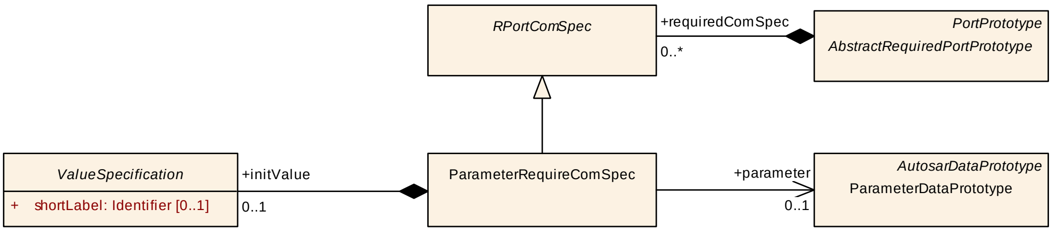


Figure 4.45: Communication attributes of ParameterDataPrototypes with respect to RPortPrototype

Class	**ParameterRequireComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the required side of a connection.			
Base	*ARObject*, *RPortComSpec*			
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	0..1	ref	The ParameterDataPrototype to which the ParameterRequireComSpec applies.

Table 4.83: ParameterRequireComSpec

[constr_1898] Existence of reference ParameterRequireComSpec.parameter ⌈For each ParameterRequireComSpec, the reference parameter shall exist **at the time when the contract phase generation is executed**.⌋()

4.5.5 Communication Specification for NV Data

4.5.5.1 NV Require ComSpec

[TPS_SWCT_01227] Unconnected AbstractRequiredPortPrototype typed by NvDataInterface ⌈For this purpose it is possible to let the AbstractRequiredPortPrototype own an NvRequireComSpec that in turn owns a ValueSpecification in the role of initValue.

It is therefore possible to provide an nvData with a reasonable value even if the corresponding AbstractRequiredPortPrototype remains unconnected.⌋*(RS_SWCT_03225)*

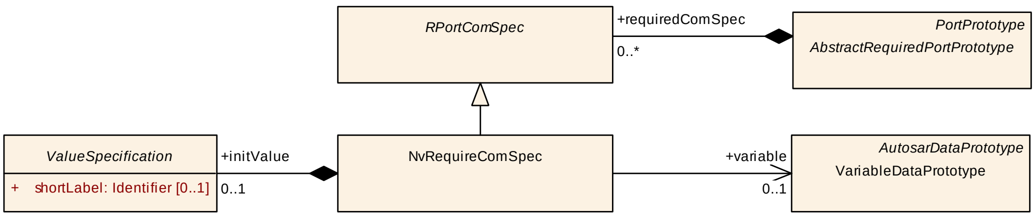


Figure 4.46: Communication attributes of a required VariableDataPrototypes used in the context of an NvDataInterface

[TPS_SWCT_01754] initValue defined in the context of a ComSpec ⌈Unless [TPS_SWCT_01755] applies, only the initValue defined in the context of a NvRequireComSpec is relevant for connections to the corresponding PortPrototype.

An initValue defined in the scope of a VariableDataPrototype shall be ignored anyway.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01755] Duplicate existence of initValue in the context of a PRPortPrototype typed by an NvDataInterface ⌈If an initValue is defined in a NvRequireComSpec owned by a PRPortPrototype, its value shall be ignored.

Instead, the initValue shall be taken from the NvProvideComSpec.ramBlockInitValue.⌋*(RS_SWCT_03225)*

Class	**NvRequireComSpec**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Communication attributes of RPortPrototypes with respect to Nv data communication on the required side.
Base	*ARObject*, *RPortComSpec*
Aggregated by	*AbstractRequiredPortPrototype*.requiredComSpec, PortPrototypeBlueprint.requiredComSpec

▽

△

Class	NvRequireComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initValue	ValueSpecification	0..1	aggr	The initial value owned by the NvComSpec
variable	VariableDataPrototype	0..1	ref	The VariableDataPrototype the ComSpec applies for.

Table 4.84: NvRequireComSpec

[constr_1899] Existence of reference NvRequireComSpec.variable ⌈For each NvRequireComSpec, the reference variable shall exist **at the time when the contract phase generation is executed**.⌋*()*

4.5.5.2 NV Provide ComSpec

[TPS_SWCT_01228] NvProvideComSpec ⌈As communication with an NvBlockSwComponentType is in most cases bi-directional it is also necessary to consider role-specific communication attributes for AbstractProvidedPortPrototypes typed by an NvDataInterface. For this purpose the NvProvideComSpec is defined.

The main purpose of this kind of ComSpec is the definition of initial values for the RAM Block and the ROM Block that corresponds to an nvData defined in the context of the NvDataInterface used to type the given AbstractProvidedPortPrototype.⌋*(RS_SWCT_03225)*

More information about NvProvideComSpec please refer to Figure 4.47.

Note that these initial values can be taken as an input for designing an NvBlockSwComponentType, in particular the ramBlocks and romBlocks of NvBlockDescriptors owned by the NvBlockSwComponentType. Further details are explained in Figure 11.8.

Also, note that the romBlockInitValue provided in the NvProvideComSpec does not necessarily have to be identical to the respective section within romBlock in the NvBlockDescriptor.

This could happen if an NvBlockSwComponentType is already existing and an ApplicationSwComponentType is connected to it. Finally, the romBlock inside the NvBlockDescriptor is the only relevant information for the RTE generation.

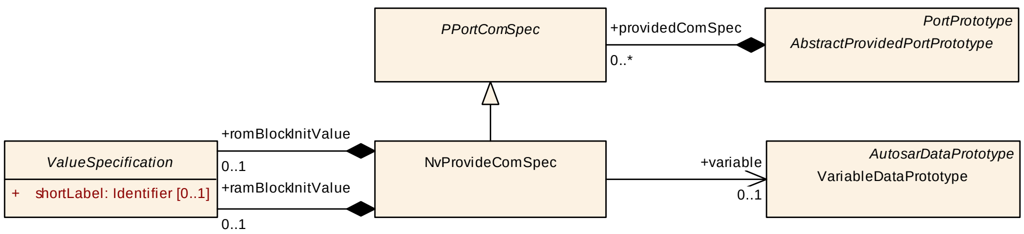


Figure 4.47: Communication attributes of a provided VariableDataPrototypes used in the context of an NvDataInterface

In other words, by means of the NvProvideComSpec the author of an ApplicationSwComponentType can express detailed requirements on the later design of a corresponding NvBlockSwComponentType.

Class	NvProvideComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of PPortPrototypes with respect to Nv data communication on the provided side.			
Base	*ARObject*, *PPortComSpec*			
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
ramBlockInitValue	ValueSpecification	0..1	aggr	This represents the initial value of the RAM Block that corresponds to the referenced variable.
romBlockInitValue	ValueSpecification	0..1	aggr	This represents the initial value of the ROM block that corresponds to the referenced variable.
variable	VariableDataPrototype	0..1	ref	This represents the variable for which the ComSpec is specified.

Table 4.85: NvProvideComSpec

[constr_1900] Existence of reference NvProvideComSpec.variable ⌈For each NvProvideComSpec, the reference variable shall exist **at the time when the contract phase generation is executed**.⌋*()*

4.5.6 Configuration of Data Transformation

The big picture of data transformation in AUTOSR is explained in the TPS System Template [10]. This chapter focuses on the aspects of data transformation that are related to the configuration of software-components.

Using the TransformationComSpecProps it is possible to define configuration options for specific transformers of inter-ECU communication which is subject to data transformation.

[TPS_SWCT_01594] Semantics of TransformationComSpecProps ⌈The definition of a TransformationComSpecProps can always be provided in the SWC description but the configuration shall **only** have an effect if

1. the actual communication involves at least two EcuInstances
2. the respective data transformer (given by the used TransformationComSpecProps) is used during data transformation (see DataTransformation)

⌋*(RS_SWCT_03221)*

For clarification, the configuration given in TransformationComSpecProps will simply be ignored if the conditions defined by [TPS_SWCT_01594] do not apply.

[TPS_SWCT_01597] PortPrototype-specific data transformation configuration ⌈Meta-class TransformationComSpecProps shall be used for the specification of

PortPrototype-specific configuration options for data transformation of inter-ECU communication.⌋*(RS_SWCT_03221)*

Please note that only some transformers offer PortPrototype-specific configuration (e.g. SOME/IP transformer doesn't have TransformationComSpecProps).

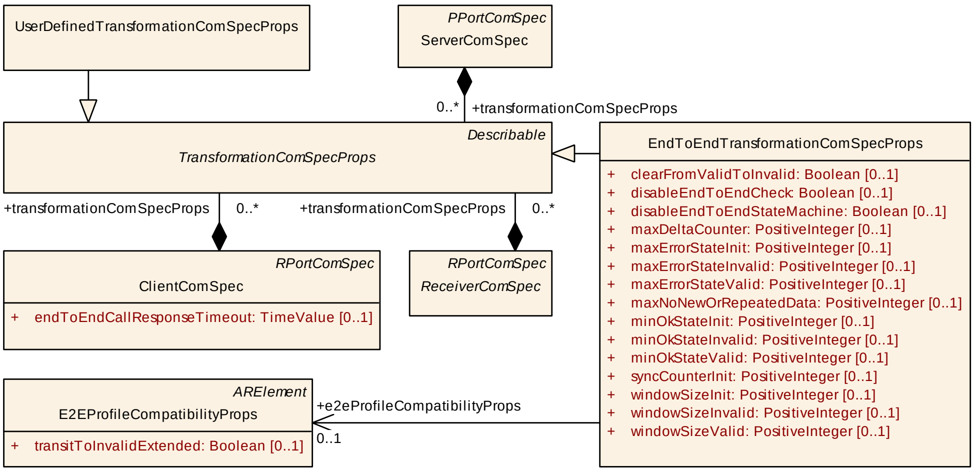


Figure 4.48: Specification of data transformation properties within ReceiverComSpec, ServerComSpec, and ClientComSpec

Class	***TransformationComSpecProps*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	TransformationComSpecProps holds all the attributes for transformers that are port specific.			
Base	*ARObject*, *Describable*			
Subclasses	EndToEndTransformationComSpecProps, UserDefinedTransformationComSpecProps			
Aggregated by	ClientComSpec.transformationComSpecProps, *ReceiverComSpec*.transformationComSpecProps, ServerComSpec.transformationComSpecProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.86: TransformationComSpecProps

It can be determined by the specific TransformationComSpecProps to which transformer this configuration is applicable:

- The configuration in EndToEndTransformationComSpecProps is applicable to E2E transformer (protocol of TransformationTechnology is set to EndToEnd).
- The configuration in UserDefinedTransformationComSpecProps is applicable to a user-defined transformer.

[TPS_SWCT_01598] More than one user-defined transformer is used within one transformer chain ⌈If more than one user-defined transformer is used within one transformer chain (defined by meta-class TransformationTechnology), the UserDefinedTransformationComSpecProps shall be assigned to the correct user-defined custom transformer in TransformationTechnology.⌋(*RS_SWCT_-03221*)

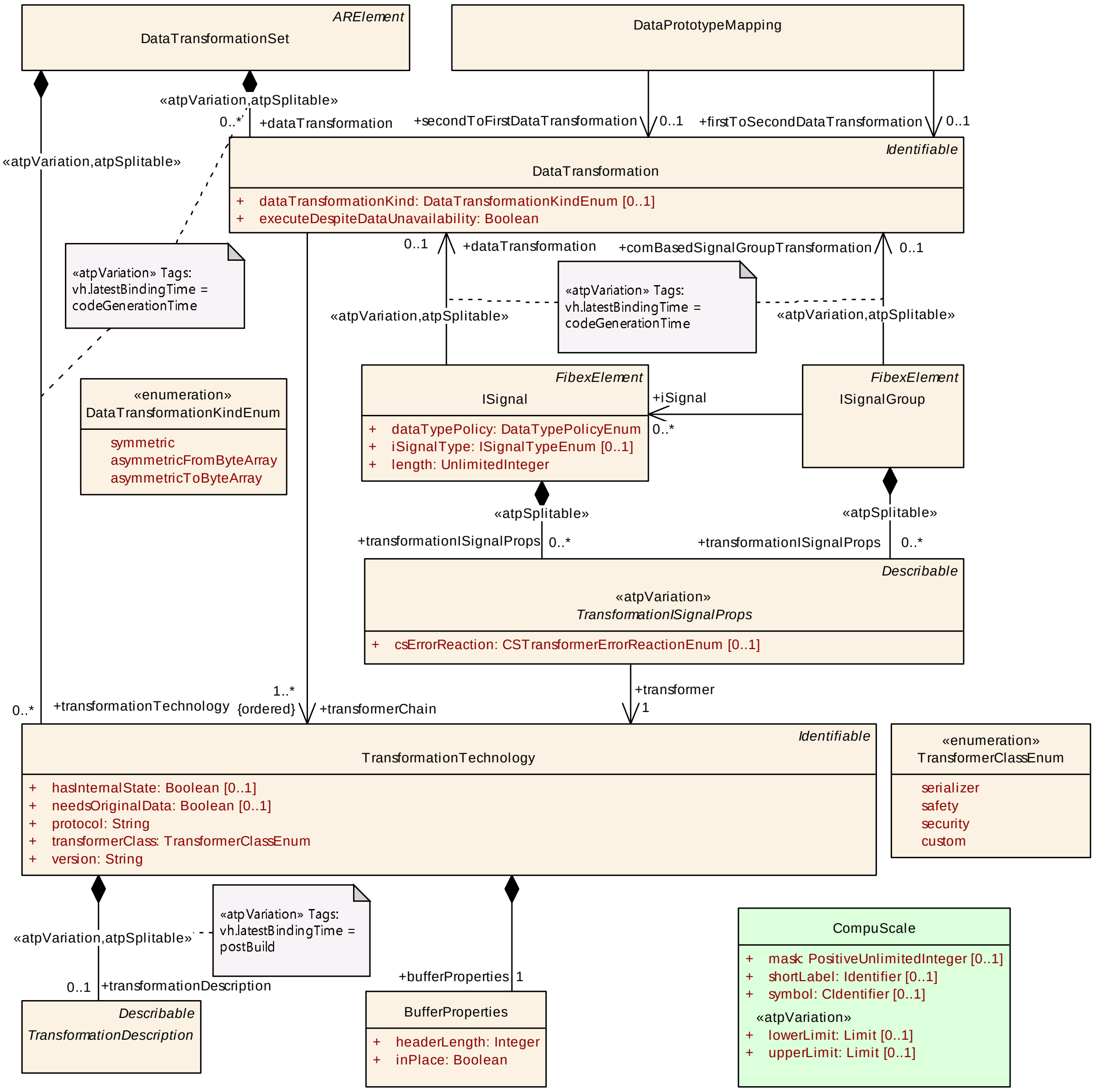


Figure 4.49: Big picture of data transformation in the AUTOSAR meta-model

[constr_1400] Reference to a specific DataTransformation ⌈A specific DataTransformation shall only be referenced by either

- a DataPrototypeMapping in the role firstToSecondDataTransformation (and potentially secondToFirstDataTransformation) **or**
- an ISignal in the role dataTransformation **or**
- an ISignalGroup in the role comBasedSignalGroupTransformation **or**
- a ClientServerOperationMapping in the role firstToSecondDataTransformation

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1401] Restrictions on the relation between DataPrototypeMapping and DataTransformation ⌈A VariableDataPrototype in the context of a PortPrototype shall – **at the time when the RTE is generated** – **not** be referenced by a DataPrototypeMapping that references a DataTransformation while a DataMapping exists that points to this VariableDataPrototype (via the SystemSignal) that also refers to an ISignal that in turn references a DataTransformation.⌋*()*

In other words: a VariableDataPrototype can either become a part of a DataPrototypeMapping-based data transformation or of an ISignal-based data transformation.

Please note that in a composite software structure the VariableDataPrototype can be delegated throughout the CompositionSwComponentType and [constr_1401] still applies.

Class	TransformationTechnology			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	A TransformationTechnology is a transformer inside a transformer chain. **Tags:**xml.namePlural=TRANSFORMATION-TECHNOLOGIES			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	DataTransformationSet.transformationTechnology			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bufferProperties	BufferProperties	1	aggr	Aggregation of the mandatory BufferProperties.
hasInternal State	Boolean	0..1	attr	This attribute defines whether the Transformer has an internal state or not.
needsOriginal Data	Boolean	0..1	attr	Specifies whether this transformer gets access to the SWC's original data.
protocol	String	1	attr	Specifies the protocol that is implemented by this transformer.
transformation Description	Transformation Description	0..1	aggr	A transformer can be configured with transformer specific parameters which are represented by the Transformer Description. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=transformationDescription, transformation Description.variationPoint.shortLabel vh.latestBindingTime=postBuild
transformer Class	TransformerClassEnum	1	attr	Specifies to which transformer class this transformer belongs.
version	String	1	attr	Version of the implemented protocol.

Table 4.87: TransformationTechnology

Class	UserDefinedTransformationComSpecProps
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	The UserDefinedTransformationComSpecProps is used to specify port specific configuration properties for custom transformers.
Base	*ARObject*, *Describable*, *TransformationComSpecProps*

▽

△

Class	UserDefinedTransformationComSpecProps			
Aggregated by	ClientComSpec.transformationComSpecProps, *ReceiverComSpec*.transformationComSpecProps, ServerComSpec.transformationComSpecProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 4.88: UserDefinedTransformationComSpecProps

Based on the user defined attributes inside UserDefinedTransformationComSpecProps (which are, of course, not standardized), the generator of the user-defined transformer shall determine to which user-defined transformer a UserDefinedTransformationComSpecProps belongs to.

[TPS_SWCT_01599] PortPrototype-specific configuration for custom transformers ⌈Meta-class UserDefinedTransformationComSpecProps shall be used for the specification of PortPrototype-specific configuration options for custom transformers.⌋*(RS_SWCT_03221)*

Please note that it is possible to add custom configuration items in UserDefinedTransformationComSpecProps by means of the attribute adminData.sdg.

Class	EndToEndTransformationComSpecProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The class EndToEndTransformationIComSpecProps specifies port specific configuration properties for EndToEnd transformer attributes.			
Base	*ARObject*, *Describable*, *TransformationComSpecProps*			
Aggregated by	ClientComSpec.transformationComSpecProps, *ReceiverComSpec*.transformationComSpecProps, ServerComSpec.transformationComSpecProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clearFromValidToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
disableEndToEndCheck	Boolean	0..1	attr	Disables/Enables the E2E check. The E2Eheader is removed from the payload independent from the setting of this attribute.
disableEndToEndStateMachine	Boolean	0..1	attr	Disables the E2EStateMachine (only E2E check functionality is performed)
e2eProfileCompatibilityProps	E2EProfileCompatibilityProps	0..1	ref	Reference to additional settings for the E2E state machine.
maxDeltaCounter	PositiveInteger	0..1	attr	Maximum allowed difference between two counter values of two consecutively received valid messages. For example, if the receiver gets data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
maxErrorStateInit	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INIT. The minimum value is 0.

▽

△

Class	EndToEndTransformationComSpecProps			
maxErrorState Invalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INVALID. The minimum value is 0.
maxErrorState Valid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_VALID. The minimum value is 0.
maxNoNewOr RepeatedData	PositiveInteger	0..1	attr	EndToEndTransformationDescription holds these attributes which are profile specific and have the same value for all E2E transformers.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT. The minimum value is 1.
minOkState Invalid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID. The minimum value is 1.
minOkState Valid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID. The minimum value is 1.
syncCounterInit	PositiveInteger	0..1	attr	EndToEndTransformationDescription holds these attributes which are profile specific and have the same value for all E2E transformers.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table 4.89: EndToEndTransformationComSpecProps

[TPS_SWCT_01600] `PortPrototype`-specific configuration for data transformers related to end-to-end protection ⌈Meta-class `EndToEndTransformationComSpecProps` shall be used for the specification of `PortPrototype`-specific configuration options for data transformers related to end-to-end protection.⌋*(RS_SWCT_03221)*

[TPS_SWCT_01812] Conditional relevance of attribute `EndToEndTransformationComSpecProps.disableEndToEndStateMachine` ⌈If `EndToEndTransformationComSpecProps.disableEndToEndCheck` is set to `true` then the value `EndToEndTransformationComSpecProps.disableEndToEndStateMachine` shall be ignored.⌋*(RS_SWCT_03221)*

[constr_5234] Existence of attribute `E2EProfileCompatibilityProps.transitToInvalidExtended` is mandatory for each `EndToEndTransformationComSpecProps` ⌈For each `EndToEndTransformationComSpecProps`, a reference in the role `e2eProfileCompatibilityProps` to meta-class `E2EProfileCompatibilityProps` shall exist and the referenced

E2EProfileCompatibilityProps shall define a value for the attribute transitToInvalidExtended **at the time when the RTE is generated.**⌋()

Class	E2EProfileCompatibilityProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	This meta-class collects settings for configuration of the E2E state machine. **Tags:**atp.recommendedPackage=E2EProfileCompatibilityPropsCollection			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
transitToInvalid Extended	Boolean	0..1	attr	E2E State machine behavior concerning transition from NODATA/INIT to INVALID value=0 (false): no direct transition from NODATA to INVALID, no transition from INIT to INVALID due to counter-related faults (Autosar R19-11 or former behavior) value=1 (true): direct transition from NODATA to INVALID covered, transition from INIT to INVALID due to counter-related faults covered (state machine extended)

Table 4.90: E2EProfileCompatibilityProps

4.6 Port Groups within Component Types

[TPS_SWCT_01063] **PortGroup** ⌈A SwComponentType can declare that some of its PortPrototypes belong to a PortGroup.

Such a port group defines a logical grouping of PortPrototypes which is used as input to configure the implementation of mode managers in the basic software, for example the communication of bus signals associated with the grouped ports maybe suppressed in a certain mode.⌋*(RS_SWCT_03200, RS_SWCT_03201)*

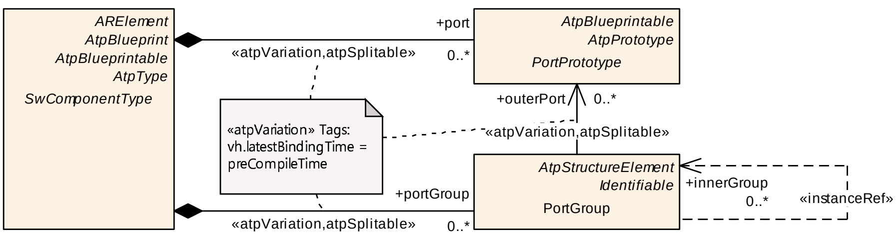


Figure 4.50: Declaration of PortGroups

[TPS_SWCT_01064] **PortGroups have to be defined on the VFB level** ⌈Though the declaration PortGroups is not relevant for the RTE, they have to be defined on the VFB level, because they represent design decisions taken on this level.

Accordingly, PortGroups can be defined for CompositionSwComponentTypes as well as for AtomicSwComponentTypes.⌋*(RS_SWCT_03200, RS_SWCT_03201)*

Class	**PortGroup**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Group of ports which share a common functionality , e.g. need specific network resources. This information shall be available on the VFB level in order to delegate it properly via compositions. When propagated into the ECU extract, this information is used as input for the configuration of Services like the Communication Manager. A PortGroup is defined locally in a component (which can be a composition) and refers to the "outer" ports belonging to the group as well as to the "inner" groups which propagate this group into the components which are part of a composition. A PortGroup within an atomic SWC cannot be linked to inner groups.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.portGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
innerGroup	PortGroup	*	iref	Links a PortGroup in a composition to another PortGroup, that is defined in a component which is part of this CompositionSwComponentType. **InstanceRef implemented by:**InnerPortGroupIn CompositionInstanceRef
outerPort	PortPrototype	*	ref	Outer PortPrototype of this AtomicSwComponentType which belongs to the group. A port can belong to several groups or to no group at all. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=outerPort.portPrototype, outerPort.variation Point.shortLabel vh.latestBindingTime=preCompileTime

Table 4.91: PortGroup

[TPS_SWCT_01065] `PortPrototype` may belong to more than one `PortGroups`
⌈A `PortPrototype` may belong to more than one `PortGroups` and `PortGroups` can be associated with the “inner” `PortGroups` of `SwComponentPrototypes` which are aggregated by the same `SwComponentType` as the `PortGroup`.

By this, `PortGroups` can be locally defined but still traced down the component hierarchy.⌋*(RS_SWCT_03200, RS_SWCT_03201)*

[TPS_SWCT_01066] `PortGroups` can be associated with certain `ServiceNeeds`
⌈`PortGroups` can be associated with certain `ServiceNeeds` in order to trace the information down to the configuration of the basic software.⌋*(RS_SWCT_03200, RS_SWCT_03201)*

For more details, see chapter 7.11.2.

[constr_1147] Standardized values for the attribute `category` of meta-class `PortGroup` ⌈

The following values of the attribute `category` of meta-class `PortGroup` are reserved by the AUTOSAR standard:

- `MODE_MANAGEMENT`: This represents the usage of the `PortGroup` for the purpose of mode management

- PARTIAL_NETWORKING: This represents the usage of the PortGroup for the purpose of partial networking

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

4.7 End to End Protection

The aspect of end-to-end protection has seen different support by the AUTOSAR meta-model.

On the one hand, there is the definition of dedicated meta-classes, e.g. EndToEndDescription, which aim at an implementation that uses a so-called E2E wrapper (an approach with a software component above RTE invoking the E2E library) or AUTOSAR Com module callout mechanism (with Com callouts used to invoke E2E library).

This approach is documented in chapter 4.7 of this document.

As an alternative approach, it is possible to implement end-to-end protection using so-called data transformers.

The detailed description of how this approach can be configured is beyond the scope of this document. Please refer to the TPS System Template [10] where the details of the alternative approach are explained.

Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR, see [18]). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.

As described in [19], there are cases where safety-related software-components protect the data exchanged between each other. For this purpose modeling support is provided by the software-component template.

Note that several end-to-end profiles are selectable for a specific application. The specific end-to-end profile is represented by the attribute category of meta-class EndToEndDescription.

Semantically, the category value represents an identification of the specific end-to-end profile applicable for the communication of the corresponding data element. According to [19] there are two pre-defined profiles that can be used.

[TPS_SWCT_01089] end-to-end communication protection ⌈The information specific to each profile is expressed by the set of attributes of EndToEndDescription owned by EndToEndProtection in the role endToEndProfile.⌋*(RS_SWCT_03240)*

Class	**EndToEndDescription**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This meta-class contains information about end-to-end protection. The set of applicable attributes depends on the actual value of the category attribute of EndToEndProtection.			
Base	*ARObject*			
Aggregated by	EndToEndProtection.endToEndProfile			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
category	NameToken	0..1	attr	The category represents the identification of the concrete E2E profile. The applicable values are specified in a semantic constraint and determine the applicable attributes of EndToEndDescription. **Tags:**xml.sequenceOffset=-100
counterOffset	PositiveInteger	0..1	attr	Bit offset of Counter from the beginning of the Array representation of the Signal Group/VariableDataPrototype (MSB order, bit numbering: bit 0 is the least important). The offset shall be a multiplicity of 4 and it should be 8 whenever possible. For example, offset 8 means that the counter will take the low nibble of the byte 1, i.e. bits 8 .. 11. If counterOffset is not present the value is defined by the selected profile. **Tags:**xml.sequenceOffset=-50
crcOffset	PositiveInteger	0..1	attr	Bit offset of CRC from the beginning of the Array representation of the Signal Group/VariableDataPrototype (MSB order, bit numbering: bit 0 is the least important). The offset shall be a multiplicity of 8 and it should be 0 whenever possible. For example, offset 8 means that the CRC will take the byte 1, i.e. bits 8..15. If crcOffset is not present the value is defined by the selected profile. **Tags:**xml.sequenceOffset=-60
dataId (ordered)	PositiveInteger	*	attr	This represents a unique numerical identifier. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd Protection. **Tags:**xml.sequenceOffset=-90
dataIdMode	PositiveInteger	0..1	attr	There are three inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC: • dataIDMode = 0: Two bytes are included in the CRC (double ID configuration) This is used in variant 1A. • dataIDMode = 1: One of the two bytes byte is included, alternating high and low byte, depending on parity of the counter (alternating ID configuration). For even counter low byte is included; For odd counters the high byte is included. This is used in variant 1B. • dataIDMode = 2: Only low byte is included, high byte is never used. This is applicable if the IDs in a particular system are 8 bits. • dataIdMode = 3: The low byte is included in the implicit CRC calculation, the low nibble of the high byte is transmitted along with the data (i.e. it is explicitly included), the high nibble of the high byte is not used. This is applicable for the IDs up to 12 bits. **Tags:**xml.sequenceOffset=-85

▽

△

Class	EndToEndDescription			
dataIdNibble Offset	PositiveInteger	0..1	attr	Bit offset of the low nibble of the high byte of Data ID. The applicability of this attribute is controlled by [constr_1261]. **Tags:**xml.sequenceOffset=-25
dataLength	PositiveInteger	0..1	attr	This attribute represents the length of the Array representation of the Signal Group/VariableDataPrototype including CRC and Counter in bits. **Tags:**xml.sequenceOffset=-80
maxDelta CounterInit	PositiveInteger	0..1	attr	Initial maximum allowed gap between two counter values of two consecutively received valid Data, i.e. how many subsequent lost data is accepted. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter Init is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. **Tags:**xml.sequenceOffset=-70
maxNoNewOr RepeatedData	PositiveInteger	0..1	attr	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions. **Tags:**xml.sequenceOffset=-40
syncCounterInit	PositiveInteger	0..1	attr	Number of Data required for validating the consistency of the counter that shall be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter. **Tags:**xml.sequenceOffset=-30

Table 4.92: EndToEndDescription

[constr_1901] Existence of attribute `EndToEndDescription.category` ⌈For each `EndToEndDescription`, attribute `category` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

[TPS_SWCT_01090] `EndToEndProtection` ⌈`EndToEndProtection` is the `Identifiable` class that owns specific elements for referencing the to-be-protected data elements and signals

- `EndToEndProtectionVariablePrototype`: a specific `dataElement` owned by a specific `PortPrototype`
- `EndToEndProtectionISignalIPdu`: a specific `ISignalGroup` in the context of an `ISignalIPdu`. For more details please refer to [10]

⌋*(RS_SWCT_03240)*

[TPS_SWCT_01091] Two cases for end-to-end protection ⌈In order to protect a VariableDataPrototype the `EndToEndProtectionVariablePrototype` shall be defined. If communication is defined between ECUs using AUTOSAR COM the `EndToEndProtectionISignalIPdu` shall be defined as well.⌋*(RS_SWCT_03240)*

The following features apply:

- **[constr_1000] End-to-end protection is limited to sender/receiver communication** ⌈A VariableDataPrototype referenced in the roles
  - EndToEndProtectionVariablePrototype.sender
  - EndToEndProtectionVariablePrototype.receiver

  shall be aggregated in the role dataElement at a SenderReceiverInterface **at the time when the contract phase generation is executed**.⌋*()*
- The value of the dataId is assigned by a central authority rather than by the developer of the software-component.
- The information about the dataId shall be available at both the sender and the receiver(s).
- **[TPS_SWCT_01508] Scope of end-to-end protection** ⌈End-to-end protection applies to local (i.e. within the ECU) as well as remote (i.e. ECU to ECU) communication.⌋*(RS_SWCT_03240)*

[TPS_SWCT_01092] EndToEndProtectionSet ⌈The meta-class EndToEndProtectionSet provides a container for EndToEndProtection. The aggregation is stereotyped ≪atpSplitable≫ because the information about end-to-end protection is added at a later step in the development workflow.⌋*(RS_SWCT_03240)*

It also has the stereotype ≪atpVariation≫ because this allows for implementing the software-component in two variants, one that uses end-to-end protection and one that does not use it. It also might happen that the communication ends themselves are variant.

EndToEndProtection maintains InstanceRefs to one dataElement in the role of sender and to one or many dataElements in the role of receiver. By this means it is possible to support a 1:n communication scenario.

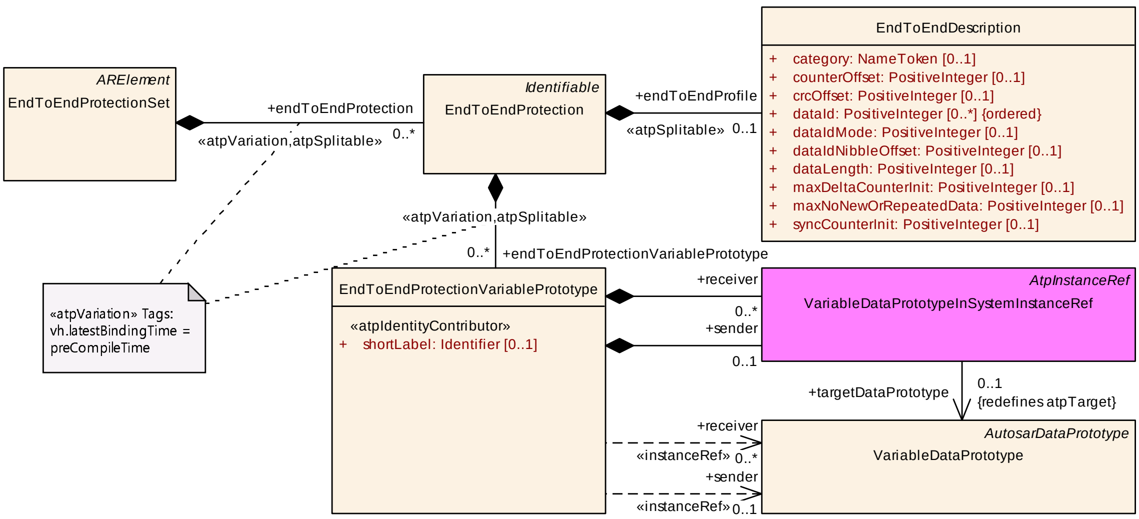


Figure 4.51: Details of the modeling of end-to-end protection

[TPS_SWCT_01093] Aggregation EndToEndProtection.endToEndProfile is splitable ⌈EndToEndProtection aggregates EndToEndDescription using stereotype ≪atpSplitable≫.

By this means it is for the integrator of an ECU possible to generally specify the nature of a specific end-to-end protection but leave the actual assignment of values (e.g. for dataId) to a later process step.⌋*(RS_SWCT_03240)*

According to [19] the following constraints apply on the attributes of EndToEndProtection (note that additional M1 constraints apply as described in [19]):

[TPS_SWCT_01094] Standardized values of attribute EndToEndDescription category ⌈The following values for the category of EndToEndDescription are standardized and reserved for being used in the way the AUTOSAR standard foresees:

- NONE
- PROFILE_01
- PROFILE_02

In addition, it is positively possible to use other than the standardized values for the category if it is ensured that the proprietary values do not clash with an extension of the standardized values, e.g. by using a company-specific prefix.⌋*(RS_SWCT_03240)*

The semantics of the standardized values of attribute EndToEndDescription.category is explained below:

NONE This value indicates that the E2E framework shall be enabled for the given sender/receiver respectively the given iSignalIPdu. The wrapper code shall be generated but it shall not invoke E2E library protection routines. E2E wrapper works as pass-through.

This may be used when a profile selection or profile options are not yet selected in a given system but it is required that the system can be built successfully under consideration of the E2E library. This would also be applicable for migrating from/to a system with/without E2E protection.

[TPS_SWCT_01095] category set to NONE ⌈If attributes exist in the presence of the category being set to NONE, the attributes shall be ignored.⌋*(RS_SWCT_-03240)*

PROFILE_01 This value indicates that the settings of E2E profile 1 (that uses a SAE CRC8, implicit 16-bit data ID, and a 4 bit alive counter) apply.

[constr_1113] Existence of attributes in PROFILE_01 ⌈In PROFILE_01, the following attributes shall exist:

- dataLength
- dataId

at the time when the contract phase generation is executed.⌋()

Please note that the attribute maxDeltaCounterInit is also part of PROFILE_01 but it does not necessarily have to exist provided that ReceiverComSpec.maxDeltaCounterInit exists.

[TPS_SWCT_01850] Usage of attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit vs. EndToEndProtection.endToEndProfile.maxDeltaCounterInit ⌈If

- the value of reference EndToEndProtection.endToEndProtectionVariablePrototype.receiver is identical to the value of the reference RPortPrototype.requiredComSpec.dataElement **and**
- attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit exists,

then the value of attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit **shall be preferred** over the value of attribute EndToEndProtection.endToEndProfile.maxDeltaCounterInit.⌋*(RS_SWCT_03240)*

[constr_1170] Existence of attribute EndToEndDescription.maxDeltaCounterInit for PROFILE_01 ⌈If the value of attribute EndToEndDescription.category is set to PROFILE_01 **and either**

- the condition described in [TPS_SWCT_01850] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit does not exist,

then attribute EndToEndProtection.endToEndProfile.maxDeltaCounterInit **shall exist at the time when the contract phase generation is executed**.⌋()

[constr_1111] Constraints of dataId in PROFILE_01 ⌈In PROFILE_01, there shall be only one element in the set and the applicable range of values is [0 .. 65535].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1112] Constraints of dataIdMode in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for dataIdMode is [0 .. 3].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1114] Constraints of crcOffset in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for crcOffset is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1115] Constraints of counterOffset in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for counterOffset is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1116] Constraints of dataLength in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for dataLength is [0 .. 240]. For the value of this attribute the constraint *value mod 8 = 0* applies.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1117] Constraints of maxDeltaCounterInit in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for EndToEndDescription.maxDeltaCounterInit and ReceiverComSpec.maxDeltaCounterInit is [0 .. 14].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1211] Constraints of maxNoNewOrRepeatedData in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for EndToEndDescription.maxNoNewOrRepeatedData and ReceiverComSpec.maxNoNewOrRepeatedData is [0 .. 14].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1212] Constraints of syncCounterInit in PROFILE_01 ⌈In PROFILE_01, the applicable range of values for EndToEndDescription.syncCounterInit and ReceiverComSpec.syncCounterInit is [0 .. 14].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[TPS_SWCT_01851] Usage of attribute RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData vs. EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData ⌈If

- the value of reference EndToEndProtection.endToEndProtectionVariablePrototype.receiver is identical to the value of the reference RPortPrototype.requiredComSpec.dataElement **and**
- attribute RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData exists,

then the value of RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData **shall be preferred** over the value of EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData.⌋*(RS_SWCT_03240)*

[constr_1215] Existence of attribute EndToEndDescription.maxNoNewOrRepeatedData for PROFILE_01 ⌈If the value of attribute EndToEndDescription.category is set to PROFILE_01 **and either**

- the condition described in [TPS_SWCT_01851] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData does not exist,

then attribute EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData **shall exist at the time when the contract phase generation is executed**.⌋()

[TPS_SWCT_01852] Usage of attribute RPortPrototype.requiredComSpec.syncCounterInit vs. attribute EndToEndProtection.endToEndProfile.syncCounterInit ⌈If

- the value of reference EndToEndProtection.endToEndProtectionVariablePrototype.receiver is identical to the value of the reference RPortPrototype.requiredComSpec.dataElement **and**
- attribute RPortPrototype.requiredComSpec.syncCounterInit exists,

then the value of RPortPrototype.requiredComSpec.syncCounterInit **shall be preferred** over the value of EndToEndProtection.endToEndProfile.syncCounterInit.⌋*(RS_SWCT_03240)*

[constr_1216] Existence of attribute EndToEndDescription.syncCounterInit for PROFILE_01 ⌈If the value of attribute EndToEndDescription.category is set to PROFILE_01 **and either**

- the condition described in [TPS_SWCT_01852] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.syncCounterInit does not exist,

then the attribute EndToEndProtection.endToEndProfile.syncCounterInit **shall exist at the time when the contract phase generation is executed**.⌋()

[constr_1261] Applicability for EndToEndDescription.dataIdNibbleOffset ⌈EndToEndDescription.dataIdNibbleOffset shall be used **only** if EndToEndDescription.dataIdMode is set to the value 3 **and** at the same time EndToEndDescription.category is set to PROFILE_01.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[TPS_SWCT_01529] Default value for EndToEndDescription.dataIdNibbleOffset ⌈If EndToEndDescription.dataIdMode is set to the value 3 **and** at the same time EndToEndDescription.category is set to the value PROFILE_01 **and** EndToEndDescription.dataIdNibbleOffset is not specified, then the default value of 12 (bits) shall be assumed for the attribute EndToEndDescription.dataIdNibbleOffset.⌋*(RS_SWCT_03240)*

PROFILE_02 This value indicates that the settings of E2E profile 2 apply.

[constr_1118] Existence of attributes in PROFILE_02 ⌈In PROFILE_02, only the following attributes shall exist:

- dataLength
- dataId

at the time when the contract phase generation is executed⌋
()

Please note that the attribute maxDeltaCounterInit is also part of PROFILE_01 but it does not necessarily have to exist provided that ReceiverComSpec.maxDeltaCounterInit exists.

[constr_1171] Existence of attribute EndToEndDescription.maxDeltaCounterInit for PROFILE_02 ⌈If the value of EndToEndDescription.category is set to PROFILE_02 **and either**

- the condition described in [TPS_SWCT_01850] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit does not exist,

then attribute EndToEndProtection.endToEndProfile.maxDeltaCounterInit **shall exist at the time when the contract phase generation is executed**.⌋()

[constr_1119] Constraints of dataLength in PROFILE_02 ⌈In PROFILE_02, the applicable range of values for dataLength is [0 .. 65535]. For the value of this attribute the constraint *value mod 8 = 0* applies.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1120] Constraints of dataId in PROFILE_02 ⌈In PROFILE_02, there shall be exactly ordered 16 elements in the set and the applicable range of values is [0 .. 255].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1121] Constraints of maxDeltaCounterInit in PROFILE_02 ⌈In PROFILE_02, the applicable range of values for EndToEndDescription.maxDeltaCounterInit and ReceiverComSpec.maxDeltaCounterInit is [0 .. 15].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1213] Constraints of maxNoNewOrRepeatedData in PROFILE_02 ⌈In PROFILE_02, the applicable range of values for EndToEndDescription.maxNoNewOrRepeatedData and ReceiverComSpec.maxNoNewOrRepeatedData is [0 .. 15].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1214] Constraints of syncCounterInit in PROFILE_02 ⌈In PROFILE_02, the applicable range of values for EndToEndDescription.syncCounterInit and ReceiverComSpec.syncCounterInit is [0 .. 15].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1217] Existence of attribute EndToEndDescription.maxNoNewOrRepeatedData for PROFILE_02 ⌈If the value of attribute EndToEndDescription.category is set to PROFILE_02 **and either**

- the condition described in [TPS_SWCT_01851] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData does not exist,

then attribute EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData **shall exist at the time when the contract phase generation is executed**.⌋*()*

[constr_1218] Existence of attribute EndToEndDescription.syncCounterInit for PROFILE_02 ⌈If the value of attribute EndToEndDescription.category is set to PROFILE_01 **and either**

- the condition described in [TPS_SWCT_01852] concerning the referenced VariableDataPrototype is not fulfilled **or**
- attribute RPortPrototype.requiredComSpec.syncCounterInit does not exist,

then the attribute EndToEndProtection.endToEndProfile.syncCounterInit **shall exist at the time when the contract phase generation is executed**.⌋*()*

Class	EndToEndProtectionSet			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This represents a container for collection EndToEndProtectionInformation. **Tags:**atp.recommendedPackage=EndToEndProtectionSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
endToEnd Protection	EndToEndProtection	*	aggr	This is one particular EndToEndProtection. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=endToEndProtection.shortName, endToEnd Protection.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 4.93: EndToEndProtectionSet

Class	EndToEndProtection			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This meta-class represents the ability to describe a particular end to end protection.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	EndToEndProtectionSet.endToEndProtection			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
endToEnd Profile	EndToEndDescription	0..1	aggr	This represents the particular EndToEndDescription. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=endToEndProfile
endToEnd Protection ISignalIPdu	EndToEndProtectionI SignalIPdu	*	aggr	Defines to which ISignalIPdu - ISignalGroup pair this End ToEndProtection shall apply. In case several ISignalGroups are used to transport the data (e.g. fan-out in the RTE) there may exist several End ToEndProtectionISignalIPdu definitions. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=endToEndProtectionISignalIPdu, endToEnd ProtectionISignalIPdu.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
endToEnd Protection Variable Prototype	EndToEndProtection VariablePrototype	*	aggr	Defines to which VariableDataPrototypes in the roles of one sender and one or more receivers this EndTo Endprotection applies. It shall be possible to aggregate several EndToEnd ProtectionVariablePrototype in case additional hierarchical decompositions are introduced subsequently. In this case one particular PortPrototype is split into multiple PortPrototypes and connectors, all representing the same data entity. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach. **Stereotypes:** atpSplitable; atpVariation **Tags:** ▽

▽

△

Class	EndToEndProtection			
				△ atp.Splitkey=endToEndProtectionVariablePrototype.shortLabel, endToEndProtectionVariablePrototype.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 4.94: EndToEndProtection

[constr_1902] Existence of attribute EndToEndProtection.endToEndProfile ⌈For each EndToEndProtection, attribute endToEndProfile shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**EndToEndProtectionVariablePrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	It is possible to protect the data exchanged between software components. For this purpose, for each communication to be protected, the user defines a separate EndToEndProtection (specifying a set of protection settings) and refers to a variableDataPrototype in the role of sender and to one or many variableDataPrototypes in the role of receiver. For details, see EndToEnd Library. Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.			
Base	*ARObject*			
Aggregated by	EndToEndProtection.endToEndProtectionVariablePrototype			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
receiver	VariableDataPrototype	*	iref	This represents the receiver. Note that 1:n communication is supported for this use case. **InstanceRef implemented by:**VariableDataPrototypeInSystemInstanceRef
sender	VariableDataPrototype	0..1	iref	This represents the sender. Can be optional if an ecu extract is provided and the sender is part of the extract. **InstanceRef implemented by:**VariableDataPrototypeInSystemInstanceRef
shortLabel	Identifier	0..1	attr	This serves as part of the split key in case of more than one EndToEndProtectionVariablePrototype is aggregated in the bound model. **Stereotypes:** atpIdentityContributor

Table 4.95: EndToEndProtectionVariablePrototype

Please note that using end-to-end protection it is explicitly supported that one sender may correspond to one or more receivers.

[constr_1183] EndToEndProtectionVariablePrototypes aggregated by EndToEndProtection ⌈All EndToEndProtectionVariablePrototypes aggregated by the same EndToEndProtection shall refer to the identical sender **at the time when the contract phase generation is executed**.⌋*()*

4.8 Partial Networking

[TPS_SWCT_01169] Support for partial networking ⌈On the level of the Software Component Template, partial networking is supported by means of the concept of a "Virtual Function Cluster" (VFC).

The latter groups all communication on the VFB with respect to a given function. However, the conceptual idea of a Virtual Function Cluster is not represented in the meta-model as such.

Instead, PortGroups are used to specify the grouping of PortPrototypes to the higher conceptual level of a Virtual Function Cluster.⌋*(RS_SWCT_03241, RS_SWCT_-03201)*

Please note that more information regarding the semantics of PortGroups can be found in chapter 4.6.

There are no restrictions regarding the structure of PortGroup definitions on M1. One PortPrototype may become a member of several PortGroups, thereby creating overlapping PortGroups.

[TPS_SWCT_01170] Purpose of Virtual Function Cluster ⌈The purpose of Virtual Function Cluster within the Software Component Template mainly has three aspects:

1. assign PortPrototypes (non service related) of Sender Receiver or Client Server communication to Virtual Function Clusters.
2. control the behavior of the corresponding function in terms of whether it is required at a given point in time. This aspect is implemented by the concept of a **control port**. Software-components that implement control ports of a Virtual Function Cluster conceptually become **VFC Controllers**.
3. allow for the application software to retrieve the status of a given Virtual Function Cluster. This aspect is implemented by the concept of a **status port**.

⌋*(RS_SWCT_03241)*

The usage of the generic concept of PortGroups for the purpose of partial networks shall be indicated by setting the value of the attribute category of PortGroup to PARTIAL_NETWORKING, see [constr_1147].

4.8.1 VFC Control Ports

[TPS_SWCT_01171] Purpose of a control port ⌈The purpose of a control port is to request or release a VFC. Requesting means that the VFC is actively using communication resources while *release* boils down to the VFC being inactive, i.e. the corresponding partial network may be shut down until further notice.

As the requesting and releasing semantics is implemented by means of interfacing the BSW the corresponding control ports need to be typed by a PortInterface that has the attribute isService set to true.⌋*(RS_SWCT_03241)*

[TPS_SWCT_01172] Requesting and releasing partial networks ⌈For requesting and releasing partial networks, the BSW can be interfaced in two alternative (i.e. either one or the other) ways:

- **ComM**: ClientServerInterface using the standardized ComM_UserRequest.RequestComMode [20]
- **BswM**: SenderReceiverInterface using the standardized AppModeRequestInterface.requestedMode [21]

⌋*(RS_SWCT_03241)*

[TPS_SWCT_01173] Control port shall not become a part of the PortGroup ⌈Please note that the control port shall **not** become a part of the PortGroup that defines the particular VFC the control port is going to service.

The relationship is implemented by means of a specific SwcServiceDependency that owns a RoleBasedPortAssignment to the intended control port **and** refers to a PortGroup (that comprises the VFC) in the role representedPortGroup.⌋*(RS_SWCT_03241, RS_SWCT_03201)*

For further information, please refer to [TPS_SWCT_01126].

4.8.2 VFC Status Ports

[TPS_SWCT_01175] Actively query the status of a partial network ⌈Very much like mode management, the concept of partial networking supports the ability to actively query the status of a partial network.

This can be done by means of interfacing the BSW in three alternative (as in "one of") ways:

- **ComM**: ClientServerInterface using the standardized ComM_UserRequest.GetCurrentComMode [20]
- **ComM**: ModeSwitchInterface using the standardized ComM_CurrentMode.currentMode [20]
- **BswM**: ModeSwitchInterface using the standardized AppModeInterface.currentMode [21]

⌋*(RS_SWCT_03241)*

As mentioned above, the status of the ComM can be retrieved by either a ClientServerInterface or a SenderReceiverInterface. Which of the two alternatives applies in a specific case is up to the author of a software-component[11].

When using one of the possible SenderReceiverInterfaces, the correspondence of the status port concept with mode management extends to the point that the status of the partial network is returned as an actual ModeDeclaration.

This implies that all mechanisms foreseen by the Software Component Template to react on mode changes are in place and can be used within the application software.

To assure that the communication via PortPrototypes that belong to a partial network is valid the software component shall consider the status of the partial network before communicating in order to assert its activity.

[TPS_SWCT_01174] Status port shall not become a member of the PortGroup ⌈A status port shall **not** become a member of the PortGroup that corresponds to the partial network subject to the status port.

The relationship is implemented by means of a specific SwcServiceDependency that owns a RoleBasedPortAssignment to the intended status port **and** refers to a PortGroup (that comprises the VFC) in the role representedPortGroup.⌋*(RS_SWCT_03241, RS_SWCT_03201)*

For further information, please refer to [TPS_SWCT_01126].

4.9 Formal Definition of implicit Communication Behavior

[TPS_SWCT_01509] Implicit communication behavior ⌈The purpose of the formal definition of the behavior of a SwComponentType with respect to the *implicit* communication can conceptually condense to two basic aspects:

- **Stable** data during the execution of a group of RunnableEntitys. This means that all data values read by different RunnableEntitys are from the same age. Therefore, the value is not changing during the execution of the chain of RunnableEntitys.
- **Coherent** data consumption and propagation for a group of DataPrototypes. This means that a set of interdependent data values are from the same calculation iteration. Therefore, the set of values has to be propagated at once to RunnableEntitys requiring the complete result of the calculation. RunnableEntitys which are part of the calculation chain may still consume partly updated values.

⌋*(RS_SWCT_03065)*

[11] The usage of the ClientServerInterface effectively implements a "pull" approach for the mode information while the usage of the SenderReceiverInterface resembles a "push" approach if it is used in combination with a SwcModeSwitchEvent.

[TPS_SWCT_01481] The meaning of the term *stability* with respect to ConsistencyNeeds ⌈The meaning of the term *stability* is that the values of a group of VariableDataPrototypes shall not change values during the execution of a group of RunnableEntitys.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01482] The meaning of the term *coherence* with respect to ConsistencyNeeds ⌈The meaning of the term *coherence* means that the values of a group of VariableDataPrototypes shall not be read by receiving RunnableEntitys until all the producing RunnableEntitys are terminated.⌋*(RS_SWCT_03065)*

In response to these goals the meta-model provides means to express the correlation between a group of RunnableEntitys and a group of DataPrototypes. These groups might be defined **hierarchically**.

The information (in terms of ConsistencyNeeds) can be defined primarily during the design of an AtomicSwComponentType but it is just as well possible to specify this ConsistencyNeeds during the definition of CompositionSwComponentTypes.

For example, the existence of stable data is typically expected for the execution of RunnableEntitys of several AtomicSwComponentTypes.

Please note that the two aspects *stability* and *coherence* are not necessarily connected to each other. It is possible to require *stability* without *coherence* and vice versa. For this purpose the roles dpgDoesNotRequireCoherency and regDoesNotRequireStability are needed.

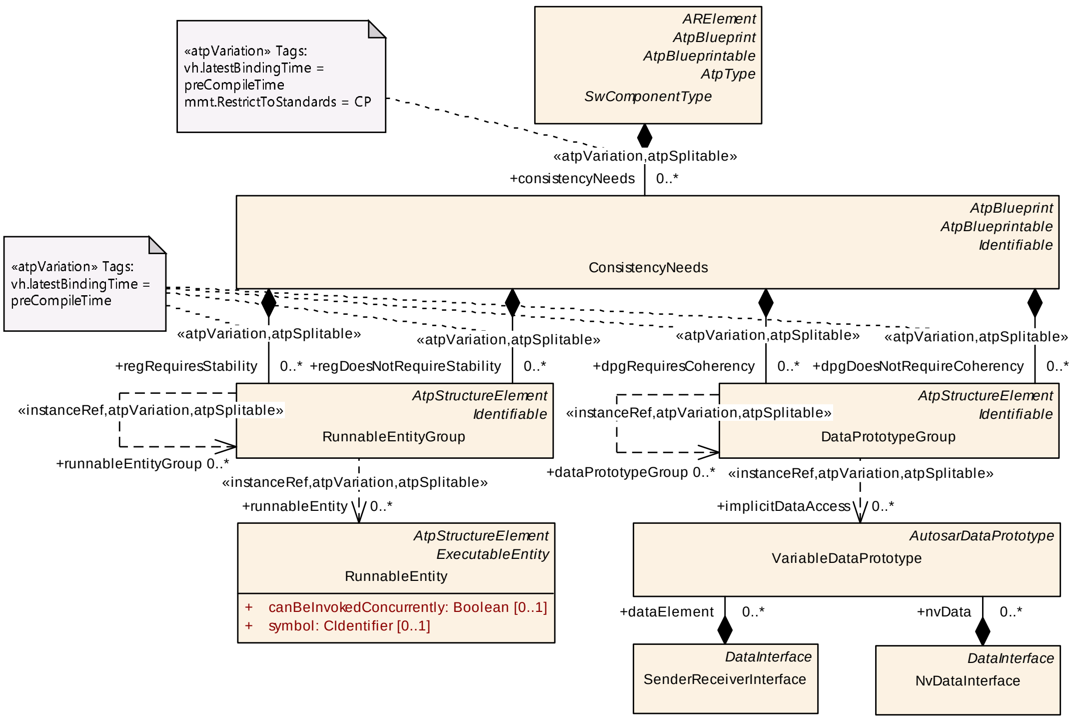


Figure 4.52: Formal definition of implicit communication behavior

[TPS_SWCT_01480] *Stability* and/or *coherence* is not required ⌈In order to be able to clearly separate the aspect of *stability* from *coherence* it is possible to use the roles dpgDoesNotRequireCoherency to express that a group of VariableDataPrototypes explicitly does not require *consistency*.

Likewise, regDoesNotRequireStability can be used to express that for a group of RunnableEntitys *stability* with respect to data access is not required.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01479] Applicability of ConsistencyNeeds ⌈ConsistencyNeeds can only be applied to RunnableEntitys that make use of "implicit" communication.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01466] ConsistencyNeeds applied on RunnableEntitys that do not use implicit communication ⌈If a ConsistencyNeeds is applied on RunnableEntitys that do not use implicit communication, it shall be ignored.⌋*(RS_SWCT_03065)*

The formal definition of the implicit communication behavior foresees the grouping of model elements in order to indicate their relevance for consistent implicit communication.

[TPS_SWCT_01470] RunnableEntityGroup ⌈A RunnableEntitys belongs to a specific RunnableEntityGroup if it is associated either directly with the given RunnableEntityGroup or if the RunnableEntityGroup the RunnableEntity belongs to is eventually (there can be more than one nesting level) referenced by the given RunnableEntityGroup.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01471] DataPrototypeGroup ⌈A VariableDataPrototypes belongs to a specific DataPrototypeGroup if it is associated either directly with the given DataPrototypeGroup or if the DataPrototypeGroup the VariableDataPrototype belongs to is eventually (there can be more than one nesting level) referenced by the given DataPrototypeGroup.⌋*(RS_SWCT_03065)*

[constr_1231] ConsistencyNeeds aggregated by CompositionSwComponentType ⌈If ConsistencyNeeds are aggregated by a CompositionSwComponentType the associations stereotyped ≪instanceRef≫ may only refer to context and target elements within the context of this CompositionSwComponentType.

This rule shall be imposed **at the time when the creation of the CompositionSwComponentType is finished**.⌋*()*

For clarification, [constr_1231] includes VariableDataPrototypes owned by delegation PortPrototypes of the owning CompositionSwComponentType, VariableDataPrototypes in delegation PortPrototypes of CompositionSwComponentType instantiated in the enclosing CompositionSwComponentType, or VariableDataPrototypes in PortPrototypes owned by AtomicSwComponentTypes instantiated inside the context of the enclosing CompositionSwComponentType.

[constr_1232] **ConsistencyNeeds** **aggregated by** **AtomicSwComponentType** ⌈If ConsistencyNeeds are aggregated by a AtomicSwComponentType the associations stereotyped ≪instanceRef≫ may only refer to context and target elements within the context of this AtomicSwComponentType **at the time when the contract phase generation is executed**.⌋*()*

Strictly speaking, these are the RunnableEntitys and PortPrototypes of this particular AtomicSwComponentType or RunnableEntityGroups and DataPrototypeGroups which are owned by the same AtomicSwComponentType.

Please note that pre-defined values for the category of RunnableEntityGroup and DataPrototypeGroup are described in [1].

Class	**ConsistencyNeeds**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior			
Note	This meta-class represents the ability to define requirements on the implicit communication behavior.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ConsistencyNeedsBlueprintSet.consistencyNeeds, *SwComponentType*.consistencyNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dpgDoesNotRequireCoherency	DataPrototypeGroup	*	aggr	This group of VariableDataPrototypes does not require coherency with respect to the implicit communication behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dpgDoesNotRequireCoherency.shortName, dpgDoesNotRequireCoherency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dpgRequiresCoherency	DataPrototypeGroup	*	aggr	This group of VariableDataPrototypes requires coherency with respect to the implicit communication behavior, i.e. all read and write access to VariableDataPrototypes in the DataPrototypeGroup by the RunnableEntitys of the RunnableEntityGroup need to be handled in a coherent manner. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dpgRequiresCoherency.shortName, dpgRequiresCoherency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
regDoesNotRequireStability	RunnableEntityGroup	*	aggr	This group of RunnableEntities does not require stability with respect to the implicit communication behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=regDoesNotRequireStability.shortName, regDoesNotRequireStability.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	ConsistencyNeeds			
regRequires Stability	RunnableEntityGroup	*	aggr	This group of RunnableEntities requires stability with respect to the implicit communication behavior, i.e. all read and write access to VariableDataPrototypes in the DataPrototypeGroup by the RunnableEntitys of the RunnableEntityGroup need to be handled in a stable manner. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=regRequiresStability.shortName, reg RequiresStability.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 4.96: ConsistencyNeeds

Class	**RunnableEntityGroup**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior			
Note	This meta-class represents the ability to define a collection of RunnableEntities. The collection can be nested.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ConsistencyNeeds.regDoesNotRequireStability, ConsistencyNeeds.reg RequiresStability			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
runnableEntity	RunnableEntity	*	iref	This represents a collection of RunnableEntitys that belong to the enclosing RunnableEntityGroup. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnableEntity.contextSwComponent Prototype, runnableEntity.targetRunnableEntity, runnable Entity.variationPoint.shortLabel vh.latestBindingTime=preCompileTime **InstanceRef implemented by:**RunnableEntityIn CompositionInstanceRef
runnableEntity Group	RunnableEntityGroup	*	iref	This represents the ability to define nested groups of RunnableEntitys. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnableEntityGroup.contextSwComponent Prototype, runnableEntityGroup.targetRunnableEntity Group, runnableEntityGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime **InstanceRef implemented by:**InnerRunnableEntity GroupInCompositionInstanceRef

Table 4.97: RunnableEntityGroup

Class	**DataPrototypeGroup**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior
Note	This meta-class represents the ability to define a collection of DataPrototypes that are subject to the formal definition of implicit communication behavior. The definition of the collection can be nested.
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	DataPrototypeGroup			
Aggregated by	*AtpClassifier*.atpFeature, ConsistencyNeeds.dpgDoesNotRequireCoherency, ConsistencyNeeds.dpgRequiresCoherency			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataPrototypeGroup	DataPrototypeGroup	*	iref	This represents the ability to define nested groups of VariableDataPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataPrototypeGroup.contextSwComponentPrototype, dataPrototypeGroup.targetDataPrototypeGroup, dataPrototypeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime **InstanceRef implemented by:**InnerDataPrototypeGroupInCompositionInstanceRef
implicitDataAccess	VariableDataPrototype	*	iref	This represents a collection of VariableDataPrototypes that belong to the enclosing DataPrototypeGroup **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implicitDataAccess.contextSwComponentPrototype, implicitDataAccess.contextPortPrototype, implicitDataAccess.targetVariableDataPrototype, implicitDataAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime **InstanceRef implemented by:**VariableDataPrototypeInCompositionInstanceRef

Table 4.98: DataPrototypeGroup

4.9.1 Consistency Needs on Receiver Side

[TPS_SWCT_01472] Receiving `SwComponentType` owns a `DataPrototypeGroup` in the role `dpgRequiresCoherency` ⌈If a receiving `SwComponentType` owns a `DataPrototypeGroup` in the role `dpgRequiresCoherency` for one or several of its `RunnableEntity`s it is required that `VariableDataPrototype`s belonging to the same `DataPrototypeGroup` are produced coherently. This means that the values of the `VariableDataPrototype`s shall be of the same age.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01473] Receiving `SwComponentType` owns a `RunnableEntityGroup` in the role `regRequiresStability` ⌈If a receiving `SwComponentType` owns a `RunnableEntityGroup` in the role `regRequiresStability` for one or several of its `RunnableEntity`s it is required that the values of implicitly communicated `VariableDataPrototype`s are kept stable over the execution of all `RunnableEntity`s belonging to the given `RunnableEntityGroup`.⌋*(RS_SWCT_03065)*

[TPS_SWCT_01474] Receiving `SwComponentType` owns a `RunnableEntityGroup` in the role `regRequiresStability` and also owns one or several `DataPrototypeGroup`s in the role `dpgRequiresCoherency` ⌈If a receiving `SwComponentType` owns a `RunnableEntityGroup` in the role `regRequiresStability` and also owns one or several DataPrototypeGroups in the role dpgRequiresCoherency it is required that values of VariableDataPrototypes belonging to the same DataPrototypeGroup are produced coherently.

This means that the values of the VariableDataPrototypes shall be of the same age **and** are kept stable over the execution of all RunnableEntitys belonging to the given RunnableEntityGroup.⌋*(RS_SWCT_03065)*

4.9.2 Consistency Needs on Sender Side

[TPS_SWCT_01475] Sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency ⌈If a sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency for one or several of its RunnableEntitys it is required that VariableDataPrototypes belonging to the same DataPrototypeGroup are propagated at the same point of time to RunnableEntitys which do not belong to the group of **producing** RunnableEntitys (which may, but don't have to be formally described as a RunnableEntityGroup).⌋*(RS_SWCT_03065)*

The coherence is created at the point in time when the RunnableEntitys of the producing group of RunnableEntitys terminate (and the implicit data get updated).

If those RunnableEntitys are reading the data also, those read accesses will not read the coherent values but the intermediary values written by RunnableEntitys of the same group.

For all other RunnableEntitys that are not member of the producing group of RunnableEntitys it appears as if the data have been updated at this very point coherently.

In order to avoid incorrect configurations its possible to explicitly define the group of RunnableEntitys for which the coherency does not apply.

[TPS_SWCT_01625] Sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency and also RunnableEntityGroups ⌈If a sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency, RunnableEntityGroups in the role regDoesNotRequireStability may exist.

Read accesses from RunnableEntitys in those RunnableEntityGroups will not read the coherent values but the intermediary values written by RunnableEntitys of the same group.⌋*(RS_SWCT_03065)*

4.9.3 Consistency Needs for Senders and receivers of the same Data inside on RunnableEntityGroup

[TPS_SWCT_01476] Sender and receiver of the same implicitly communicated `VariableDataPrototype`s are associated with the same `RunnableEntityGroup` ⌈For the case of sender and receiver of the same implicitly communicated `VariableDataPrototype`s are associated with the same `RunnableEntityGroup` [TPS_SWCT_01472], [TPS_SWCT_01473], [TPS_SWCT_01475] as well as [TPS_SWCT_01475] apply with the exception that updates of the values of implicitly communicated `VariableDataPrototype`s inside the given `RunnableEntityGroup` become visible **immediately** after the producing `RunnableEntity` was terminated.⌋*(RS_SWCT_03065)*

5 Data Description

5.1 Introduction

[TPS_SWCT_01229] Three different levels of abstraction regarding the definition of data types ⌈In the context of defining data types and prototypes, the AUTOSAR concept distinguishes between three different levels of abstraction as depicted in Table 5.1.⌋*(RS_SWCT_03215, RS_SWCT_03216, RS_SWCT_03217)*

Application Data Level
Implementation Data Level
Supplemental Base Type Level

Table 5.1: Abstraction Levels for Describing Data

[TPS_SWCT_01230] Application Data Level ⌈The ***Application Data Level*** is the common level at which `ApplicationSwComponentType`s specify a data type or prototype.

This level allows to define all the data attributes which are needed from the application point of view, in order to exchange data between software components or between a software component and a measurement and calibration tool. It is possible to specify data communication of a complete `Virtual Function Bus` based on this level only.

This level includes among other things the numerical range of values, the data structure as well as the physical semantics.

Data semantics (e.g. physical units) is not in the focus[1] for the RTE in order to make communication technically possible.

However, it is important for a unique interpretation of data in the application software and in measurement and calibration systems.⌋*(RS_SWCT_03216)*

Please note that `ApplicationDataType`s – by virtue of being platform-independent by definition – do not become visible as data types in the code implementation of software-components.

In former version of this specification, this level was not clearly separated from the implementation level. These had the following drawbacks which are now solved:

- The model of primitive types (like integer, boolean, real, opaque) was anticipating implementation aspects already on a very high level of design.
- The data type model used within ports, focusing on communication via the RTE, was not sufficient to model all type-aspects of variables and parameters which are

[1] There are some aspects that affect the RTE, e.g. scaling of `dataElement`s

visible within an AUTOSAR system for other purposes than RTE-communication, namely NvM-data access, calibration, measurement, diagnostics, BSW-module interfaces. Using a uniform type system covering all these aspects is now favored.

- Calibration parameters were not completely incorporated into the data type concept. Some of their attributes (especially for curves and maps) could be specified only on the level of prototypes or were not completely formalized within AUTOSAR (like `SwRecordLayout`).
- The data type system was not compatible with the usage in calibration standards like ASAM-MCD (namely the usage of `category`s).
- Adding implementation specific elements like a base type, was not possible without formally changing the data type used in a VFB design. A mapping mechanism that could be used in later project phases and is common in other parts of AUTOSAR (e.g. for mapping components to ECUs) was missing.
- The RTE Specification contained many default rules and assumptions on how to implement certain data types or prototypes in C. With a more formal description of all relevant implementation aspects, the generation of C-interfaces is better determined. But these aspects should be separated from the application level design.
- Since there could be many data types on the application level in a big system, the probability of name clashes in the interfaces to the RTE was rather high. Using a separate set of types to implement the RTE interfaces solves this issue.

[TPS_SWCT_01231] Application level may impose strong requirements on the design of the corresponding implementation level ⌈It should be pointed out, that with the specification of computation methods and record layouts, the application level imposes strong requirements on the design of the corresponding implementation level. It might even be the case, that when anticipating different implementations, these elements might be chosen differently.

This is due to the nature of these elements which form a bridge from the physical world to the numerical representation (and vice versa). Nonetheless, we consider the specification of these elements as belonging to the application level.

On the one hand, this information is required by MCD-tools and thus shall be part of a rather high-level design. On the other hand, this approach will allow to use a limited set of implementation data types.⌋*(RS_SWCT_03215, RS_SWCT_03216, RS_SWCT_03217)*

Further information about the compatibility requirements between application level and implementation level can be found in section 6.2.5.

[TPS_SWCT_01232] Implementation Data Level ⌈The ***Implementation Data Level*** is closer to the actual code implementation in a programming language like C, though it is still an abstraction of the code.

Its values correspond to the actual binary numbers handled by the programming language on the CPU. It contains concepts like pointers and unions which relate to the organization of data in memory and are not relevant for the application level.

This level also defines structure, but it can be more granular. For example, the application level may define a text to be transferred to an instrument cluster as a primitive type (if the structure is not relevant for the application), whereas on the implementation level it could be modeled as an array of bytes.⌋*(RS_SWCT_03217)*

[TPS_SWCT_01233] Use case for the Implementation Data Level ⌈There are several use cases for this level in AUTOSAR:

- First, the *Implementation Data* level can be used in the description of interfaces, and data (e.g. debug data) within the basic software, see [6] for more details on these use cases.
- `ImplementationDataType`s should also be used to describe the interfaces of libraries which operate on a purely numerical level.
- *Implementation Data* 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.
- It is possible to define communication in a VFB system directly on this level if the physical and semantical abstraction is not of interest.
- Last not least the input for the RTE generator is defined by data descriptions on this level. This means that in case a SWC defines its data only on application level a corresponding set of implementation data types shall be created (or generated) as part of the ECU extract before the RTE can be generated.

⌋*(RS_SWCT_03217)*

[TPS_SWCT_01234] Base Type Level ⌈The ***Base Type Level*** is used to contribute "low-level" aspects in terms of bits and bytes from which the implementation data is built up. It is considered as a separate level in order to allow for reuse of the basic types defined on this level.

These base types still do not completely determine the actual implementation on a programming language, but they impose strong restrictions for this as they define, for example, the number of bits and bytes to be used, as well as the byte order and the encoding of primitive "leaf" objects.

The base type layer does not represent a "standalone" way to define a data type in AUTOSAR. A `DataPrototype` may be typed by an `AutosarDataType`, i.e. either an `ApplicationDataType` or an `AbstractImplementationDataType`. There is no option to use the base type level exclusively to type a `DataPrototype`.⌋*()*

[TPS_SWCT_01235] Mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level ⌈It is important to understand, that the mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level depends on the medium on which the data is transported.

For example, if a physical value can be expressed with sufficient accuracy and range by a 16-bit unsigned integer, it still might look very different when sent over CAN, when seen by a software-component on a *big-endian* 32-bit machine or when seen by a software-component on a *little-endian* 16-bit processor.

Conversion between several data implementations of the same application data type might be necessary in case of communication between components on different ECUs. AUTOSAR COM [22] is responsible for this.

It implies that the configuration depends on the definition of the data that are transmitted between components[2].⌋*(RS_SWCT_03215, RS_SWCT_03216, RS_SWCT_03217)*

AUTOSAR COM might need to convert a 16-bit integer between *little-endian* and *big-endian* representations; whereas an array of 16 bytes does not need to be swapped even if the endianess changes. In case of intra-ECU communication byte order conversion is not necessary, since the software-components reside on the same machine.

[TPS_SWCT_01236] Big picture of data types ⌈Another way of approaching the concept of data types in AUTOSAR (especially with respect to the question of what "kind" of data type in related to which modeling meta-level) is to sketch the following "big picture" of data types:

ApplicationDataType Defined on **M2** - provides the meta-model for data types on application level. It covers the application-relevant aspects of a data type.

An `ApplicationDataType` shall finally be mapped to an `ImplementationDataType`.

ImplementationDataType Defined on **M2** - provides the meta-model for data types on implementation level. With respect to C source code, an `ImplementationDataType` finally boils down to a `typedef`.

BaseType Defined on **M2** - provides the platform-dependent part of an `ImplementationDataType`. the dependency on the platform covers the following aspects:

- Definition on the level of the C language - using `nativeDeclaration`
- Technical representation on the target platform (byte order, alignment, encoding) as required for the support of MCD systems.
- The modeling of this level based on `SwBaseType` is not restricted to the definition of data types in the software domain. It is used in several different contexts in AUTOSAR whenever exact knowledge of the bits and bytes is required.

Platform Data Type Defined on **M1** - provided by AUTOSAR. Platform types shall be available on each platform on which an AUTOSAR-System can run.

The name of the `Platform Data Type` and the properties with respect to the interface between modules / components is the same on every platform.

[2]More exactly speaking, the data shall be converted to and from a so-called `SystemSignal`.

The particular representation varies from platform to platform.

Platform Data Types shall be **modeled** using Implementation-DataTypes.

Note that in AUTOSAR R3.x the platform types are implemented manually and could even not be expressed on ARXML model (see [SRS_Rte_00150]). In AUTOSAR R4.1 the Platform Data Types can be represented in the ARXML model. Subsequent releases of AUTOSAR may generate the Platform Data Types directly from the ARXML Model.

Standard Type Defined on **M1** - provided by AUTOSAR. Standard types are defined by referring to platform types.

⌋*(RS_SWCT_03215, RS_SWCT_03216, RS_SWCT_03217)*

[TPS_SWCT_01237] SwDataDefProps ⌈The properties of data are summarized in the meta-class SwDataDefProps. This meta-class itself is the superset of all applicable properties.⌋*(RS_SWCT_03216, RS_SWCT_03217)*

Subsets of SwDataDefProps are applicable in specific case, for a summary please refer to the following tables:

- The data categorys are summarized in [constr_1006].
- Properties for ApplicationDataTypes are summarized in [constr_1007].
- Properties for ImplementationDataTypes are summarized in [constr_1009].
- Properties for DataPrototypes typed by ApplicationDataTypes are summarized in [constr_1289].
- Properties for DataPrototypes typed by ImplementationDataTypes are summarized in [constr_1288].
- Applicability of SwDataDefProps is summarized in [constr_1015].

5.2 Data Types

5.2.1 Overview

As explained in section 5.1 it is possible to describe data provided by a software-component from the application as well as from the implementation point of view.

[TPS_SWCT_01072] ApplicationDataType and ImplementationDataType ⌈The common concept behind this is expressed by the abstract meta-class AutosarDataType, from which an ApplicationDataType and an ImplementationDataType is derived.⌋*(RS_SWCT_03215, RS_SWCT_03216, RS_SWCT_03217)*

[TPS_SWCT_01073] Composite ApplicationDataType ⌈An ApplicationDataType can be composed (in form of a record or an array) of elements which themselves are typed by another ApplicationDataType.⌋(RS_SWCT_03215, RS_-SWCT_03216)

[TPS_SWCT_01074] Composite ImplementationDataType ⌈An ImplementationDataType can also be composed of elements but in this case no type/prototype concept (see [11]) has been applied. Both concepts will be explained in the following chapters in more detail.⌋(RS_SWCT_03215, RS_SWCT_03217)

Figure 5.1 shows a summary of the basic meta-classes used for the definition of AutosarDataTypes.

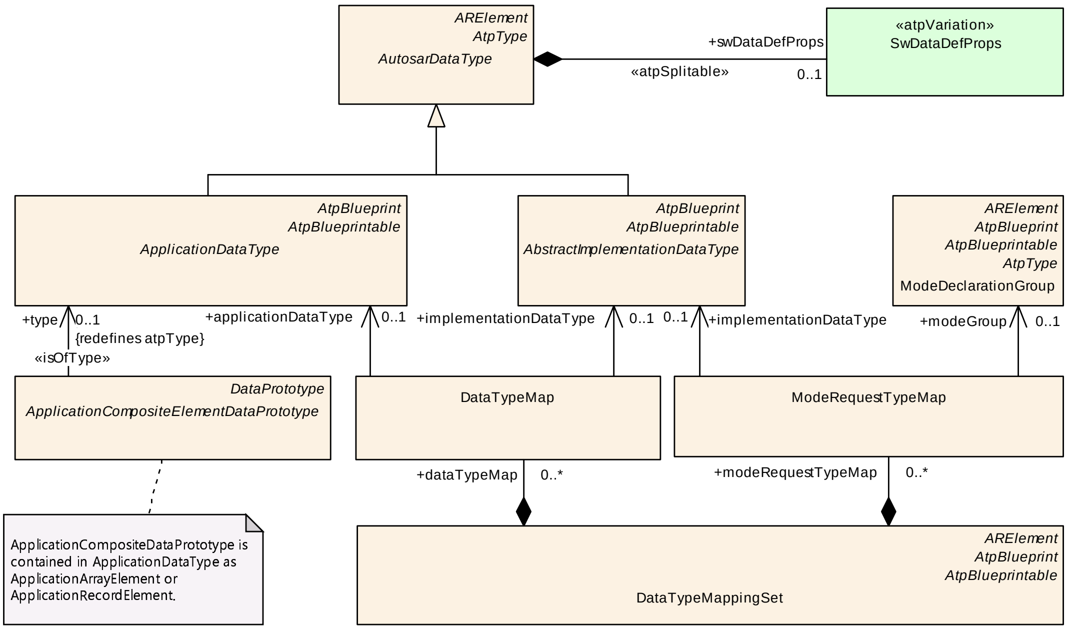


Figure 5.1: Summary of AutosarDataType

Class	***AutosarDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for software.			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractImplementationDataType*, *ApplicationDataType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps

Table 5.2: AutosarDataType

Class	***ApplicationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationData Types only.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*ApplicationCompositeDataType*, ApplicationPrimitiveDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.3: ApplicationDataType

5.2.2 Data Type Mapping

As explained above, the concept of application data types as well as that of implementation data types can be used to instantiate a data prototype in an M1 model. However, there are use cases, especially in order to generate the RTE contract for ApplicationSwComponentTypes, where it is required to consider both levels for one given data prototype.

[TPS_SWCT_01189] **DataTypeMap** ⌈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 dataElement.⌋*(RS_SWCT_03216, RS_SWCT_03217, RS_SWCT_03215)*

Class	**DataTypeMap**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing Abstract ImplementationDataType.			
Base	*ARObject*			
Aggregated by	DataTypeMappingSet.dataTypeMap			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationData Type	ApplicationDataType	0..1	ref	This is the corresponding ApplicationDataType
implementation DataType	AbstractImplementation DataType	0..1	ref	This is the corresponding AbstractImplementationData Type.

Table 5.4: DataTypeMap

If, for example, a dataElement in a SenderReceiverInterface is typed by an ApplicationDataType it shall additionally be associated to an ImplementationDataType in order to be able to generate the RTE.

[constr_1903] Existence of reference DataTypeMap.applicationDataType ⌈For each DataTypeMap, reference applicationDataType shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1904] Existence of reference DataTypeMap.implementationDataType ⌈For each DataTypeMap, reference implementationDataType shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01190] ModeRequestTypeMap ⌈Another mapping class, ModeRequestTypeMap, has been introduced in order to allow the transport of mode related information via "normal" sender-receiver communication. Apart from this, mode information is not handled by the usual type system but needs special meta-classes.⌋ *(RS_SWCT_03110)*

This aspect is explained in more detail in chapter 4.2.5.

Note that the mapping classes instead of direct associations have been introduced for process reasons: It allows maintaining application and implementation types in separate M1 artifacts without direct links.

For example, if a software component is moved to another hardware platform the mapping between application and implementation types might be changed in the scope of the specific component without changing the overall VFB model.

[TPS_SWCT_01191] mapped ApplicationDataType and ImplementationDataType shall be compatible ⌈In order to set up a valid DataTypeMap between an ApplicationDataType and an ImplementationDataType the two types shall be compatible.

Of course, if ImplementationDataTypes are generated from existing ApplicationDataTypes it is expected that they will be automatically compatible.⌋*(RS_SWCT_03216, RS_SWCT_03217)*

Please note that the compatibility between an ApplicationDataType and an ImplementationDataType mapped onto each other is clarified in chapter 6.2.5.

Furthermore, the various mappings are aggregated in a container DataTypeMappingSet for easier maintenance in artifacts.

Class	**DataTypeMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups. **Tags:**atp.recommendedPackage=DataTypeMappingSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	DataTypeMappingSet			
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an ApplicationDataType and its AbstractImplementationDataType.
modeRequestTypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an ModeDeclarationGroup and its AbstractImplementationDataType.

Table 5.5: DataTypeMappingSet

Note that the meta-classes AutosarDataType, ModeDeclarationGroup and DataTypeMappingSet are derived from ARElement. This means that these and the meta-classes derived from them can be declared on the M1 level as part of an ARPackage and thus can be used in some Software Component or Basic Software Module Descriptions.

How to organize DataTypeMappingSets for a software system, for example whether there is a separate mapping set for each ECU or even for each software component, is considered as project specific. However, the RTE generator needs a well-defined DataTypeMappingSet as input in relation those artifacts which might define data typed as ApplicationDataTypes.

[TPS_SWCT_01192] Meta-classes that have an association to a DataTypeMappingSet ⌈Therefore, the following meta-classes in the scope of this document have an association to a DataTypeMappingSet:

- InternalBehavior, because it represents the interface between the software component's code and the RTE and all data types belonging to the particular component type have to be uniquely provided on implementation level.
- ParameterSwComponentType, for the same reason (this component type doesn't have an InternalBehavior).
- NvBlockDescriptor, because this meta-class also leads to generation of code from data types and is not associated to an InternalBehavior.
- CompositionSwComponentType, to support the definition of ComSpecs in the context of a CompositionSwComponentType. Please note that this definition of a data type mapping is informal (i.e. it shall be taken as a hint for delegation PortPrototypes that are not yet referenced by a DelegationSwConnector or PassThroughSwConnector) and shall **not** be regarded as a binding contract towards the inner elements of the CompositionSwComponentType.

⌋*()*

For more details about this aspect please refer to figure 5.75.

[TPS_SWCT_01193] Mappings between application and implementation types do not necessarily have to form a 1:1 relation ⌈In general, it is not required that the sum of all mappings between ApplicationDataType and ImplementationDataType in a given system form a 1:1 relation. Depending on the use case and on the scope, 1:n as well as n:1 mappings are possible :

- Several ApplicationDataTypes may be mapped to the same ImplementationDataType in the scope of a system, an ECU, or even a single InternalBehavior of an atomic software component.

  Of course, this requires that the different ApplicationDataTypes are used for different DataPrototypes and thus that the DataPrototypes are typed by them (and not by the ImplementationDataTypes). This allows to establish a more simple type system on the implementation level, than on the application model level.

- The same ApplicationDataTypes may be mapped to different ImplementationDataTypes for different ECUs. This scenario allows to choose the implementation data types according to the needs of specific ECUs.

- The same ApplicationDataTypes may be mapped to different ImplementationDataTypes even in the scope of a single ECU (more exactly speaking, a single RTE), but only for different AtomicSwComponentTypes (see [constr_1004]).

  This improves the portability of software components which were developed independently or are ported between ECUs.

⌋*()*

[constr_1004] Mapping of ApplicationDataTypes in the scope of single AtomicSwComponentTypes ⌈In the scope of AtomicSwComponentType.internalBehavior.dataTypeMapping, each ApplicationDataType shall be mapped to exactly one ImplementationDataType **at the time when the contract phase generation is executed**.⌋*()*

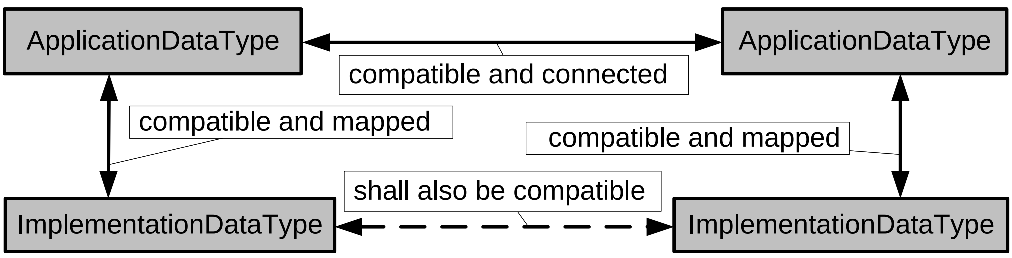


Figure 5.2: Compatibility of Data Types

[constr_1005] Compatibility of ImplementationDataTypes mapped to the same ApplicationDataType ⌈It is required that ImplementationDataTypes which are taken for connecting corresponding elements of PortInterfaces and thus refer to compatible ApplicationDataTypes are also compatible among each other (so that RTE is able to cope with possible connections by converting the data accordingly) **at the time when the contract phase generation is executed**.⌋*()*

This constraint is visualized in figure 5.2.

[constr_1636] Mapping of data types that represent an Optional Element Structure ⌈An ApplicationRecordDataType with at least one element where attribute isOptional is set to true shall only be mapped to an ImplementationDataType that fulfills the structural requirements to represent an Optional Element Structure (see [TPS_SWCT_01774]) **at the time when the contract phase generation is executed**.⌋*()*

5.2.3 Data Categories

An AutosarDataType is derived from Identifiable, thus having a longName, a shortName, a category, and several further attributes for administrative and documentation purposes (for details see [11]).

[TPS_SWCT_01238] Attribute category used in the context of AutosarDataType ⌈The category attribute is used to set constraints for the various properties which can be specified for an AutosarDataType. These properties are defined by aggregating the meta-class SwDataDefProps which contains several attributes and references.⌋*()*

Detailed explanations about the semantics of meta-class SwDataDefProps can be found in chapter 5.4.

This approach avoids a very deep and complicated inheritance tree which otherwise would be needed on the M2 level for AutosarDataType.

There is to some extent a redundancy between setting the category and defining the attributes of AutosarDataType.swDataDefProps. This redundancy is intended and allows to for a tool to rule out senseless configurations via simple rules.

In former version of this specification the categories were only used for calibration parameters. Due to several extensions the categories are now applicable for all use cases of the AutosarDataType.

An overview on all valid categorys defined for AutosarDataType is shown in [constr_1006].

Some categorys are also applied to sub-elements of the type system (column "Applicable to ..." in [constr_1006]). This is explained in more detail in the following sections.

Please note that the column "RTE + BSW" of [constr_1006] is only applicable for categorys that are relevant either for ImplementationDataTypes and/or the aspect of measurement and calibration in McDataInstance.

[constr_1006] Applicable data categories, depending on specific model elements related to data definition properties ⌈

Category	Applicable to ...											Use Case				Description
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ApplicationRuleBasedValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
VALUE			x	x	x	x		x	x	x	x	x	x	x	x	Contains a single value.
VAL_BLK			x	x	x	x	x				x	x		x		A value block defines values stored together within one calibration parameter object. It is similar to an value array but it stores the values by means of an axis instead (only important for calibration data handling).
DATA_REFERENCE								x	x	x				x[3]	x	Contains an address of another DataPrototype (whose type is given via SwDataDefProps.swPointerTargetProps).
FUNCTION_REFERENCE								x	x	x					x	Contains an address of a function prototype (whose signature is given via SwDataDefProps.swPointerTargetProps.functionPointerSignature).
TYPE_REFERENCE								x	x	x				x	x	The element is defined via reference to another data type (via SwDataDefProps.implementationDataType).
STRUCTURE		x		x	x			x	x		x	x	x	x	x	Holds one or several further elements which can have different AutosarDataTypes. The underlying elements are defined in the same manner as normal data except for the association to SwAddrMethod: This has to be the same for all underlying elements. Corresponds to a Record if used in the application domain.
UNION								x	x		x	x	x	x	x	Can hold values of different data types. It is similar to STRUCTURE except that all of its members start at the same location in memory. A UNION data prototype can contain only one of its elements at a time. The size of the UNION is at least the size of the largest member. Please find more information in [TPS_SWCT_01700].
ARRAY	x			x	x		x	x	x		x	x	x	x	x	An array of sub-elements which are of the same type.
BIT											x	x	x		x	One or several bits within a host variable, which are treated as an own data object.

▽

[3][constr_1295] applies!

△

Category	Applicable to ...											Use Case				Description
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ApplicationRuleBasedValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
HOST											x	x	x		x	A HOST data type is like a simple VALUE, but it is used for packed bit definition. That means it can host several BIT variables which have their own description and measurement access.
STRING			x	x	x	x					x	x	x	x		Contains a single value interpreted as a text string (note that it appears as a single value for the application domain; the internal representation can be an array).
BOOLEAN			x	x	x	x					x	x	x	x		Contains one boolean state. Depending on the CPU direct addressing of single bits may not be available. So a byte or a word can be used to store only one logical state.
COM_AXIS			x	x	x	x	x				x	x		x		An axis definition as separate calibration parameter which can be referenced by any CURVE, MAP, CUBOID, CUBE_4, and CUBE_5. The benefits by using a common axis is that it saves memory space; because it is stored only one time and can be used in multiple CURVEs, MAPs, CUBOIDs, CUBE_4s, and CUBE_5s.
RES_AXIS			x	x	x	x	x				x	x		x		A RES_AXIS (rescale axis) is also a shared axis like COM_AXIS, the difference is that this kind of axis can be used for rescaling. Note that the RES_AXIS is by nature a CURVE which is used to implement a non linear scaling (rescale) of the axis. In addition to saving memory space via the shared usage like a COM_AXIS, it can compress a huge range to a non-linear distributed axis points thus retaining the required accuracy.
CURVE			x	x	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.

▽

△

Category	Applicable to ...											Use Case				Description
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ApplicationRuleBasedValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
MAP			x	x	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.
CUBOID			x	x	x	x	x				x	x		x		Calibration parameter with three 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 the input axes. A CUBOID has always three input axes and one output axis. The output axis is a characteristic of the CUBOID and every time present but the input axes can be defined within the CUBOID definition or separately.
CUBE_4			x	x	x	x	x				x	x		x		Calibration parameter with four 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 the input axes. A CUBE_4 has always four input axes and one output axis. The output axis is a characteristic of the CUBE_4 and every time present but the input axes can be defined within the CUBE_4 definition or separately.

▽

△

Category	Applicable to ...											Use Case				Description
	ApplicationArrayDataType	ApplicationRecordDataType	ApplicationPrimitiveDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ApplicationRuleBasedValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
CUBE_5			x	x	x	x	x				x	x		x		Calibration parameter with five 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 the input axes. A CUBE_5 has always five input axes and one output axis. The output axis is a characteristic of the CUBE_5 and every time present but the input axes can be defined within the CUBE_5 definition or separately.

This rule shall be imposed **at any time in the workflow**.

⌋()

Please note further that the description of SwServiceArg is available in the TPS BSW Module Description Template [6].

[TPS_SWCT_01239] default value for attribute category used in the context of SwSystemconst ⌈The default value for the category of a SwSystemconst shall be VALUE. This has to be applied if no explicit definition of the category can be found.⌋ *()*

5.2.4 Application Data Type

[TPS_SWCT_01240] Subclasses of ApplicationDataType ⌈The abstract meta-class ApplicationDataType is further derived into an ApplicationPrimitiveDataType and an ApplicationCompositeDataType which are further explained in the following sub-chapters.⌋*(RS_SWCT_03216)*

This aspect is further explained in Figure 5.3.

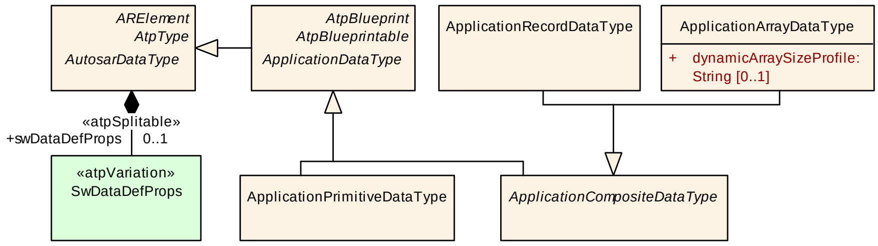


Figure 5.3: Basic Meta-Model for ApplicationDataType

Class	**ApplicationPrimitiveDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	A primitive data type defines a set of allowed values. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.6: ApplicationPrimitiveDataType

Class	***ApplicationCompositeDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for all application data types composed of other data types.			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ApplicationArrayDataType, ApplicationRecordDataType			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.7: ApplicationCompositeDataType

[TPS_SWCT_01241] Applicable categorys for subclasses of ApplicationDataType ⌈Like any AutosarDataType, also the primitive and composite types on application level are characterized by their category and their SwDataDefProps. For a given category, only a limited set of attributes of the SwDataDefProps makes sense.⌋*(RS_SWCT_03216)*

[constr_1007] Allowed attributes of SwDataDefProps for ApplicationDataTypes ⌈

Attributes of SwDataDefProps	Root Elem.			Attribute Existence per Category												
	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP	CUBOID	CUBE_4	CUBE_5
additionalNativeTypeQualifier																
annotation	x	x	x	*	*	*	*	*	*	*	*	*	*	*	*	*
baseType																
compuMethod	x			0..1	0..1				0..1			0..1	0..1	0..1	0..1	0..1
dataConstr.dataConstrRule.physConstrs	x	x	x	0..1	0..1		0..1		0..1			0..1	0..1	0..1	0..1	0..1
dataConstr.dataConstrRule.internalConstrs	x	x	x	d/c[4]	d/c		d/c		d/c			d/c	d/c	d/c	d/c	d/c
displayFormat	x	x	x	0..1	0..1		0..1	0..1	0..1			0..1	0..1	0..1	0..1	0..1
displayPresentation	x	x	x	0..1	0..1		0..1			0..1	0..1	0..1	0..1	0..1	0..1	0..1
implementationDataType																
invalidValue	x			0..1				0..1	0..1							
stepSize	x	x	x	0..1	0..1		0..1			0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAddrMethod	x			0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment																
swBitRepresentation																
swCalibrationAccess	x	x		0..1	0..1	0..1	0..1	0..1	0..1	1	1	1	1	1	1	1
swCalprmAxisSet	x									1	1	1	1	1	1	1
swComparisonVariable																
swDataDependency																
swHostVariable																
swImplPolicy	x			0..1	0..1	0..1	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			0..1						0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIsVirtual																
swPointerTargetProps																
swRecordLayout	x			0..1	0..1[5]			0..1		1	1	1	1	1	1	1
swRefreshTiming	x			0..1	0..1			0..1	0..1							
swTextProps	x							1								
swValueBlockSize	x				1											
swValueBlockSizeMult	x				1											
unit	x			0..1	0..1			0..1	0..1			0..1	0..1	0..1	0..1	0..1
valueAxisDataType	x				0..1					0..1	0..1	0..1	0..1	0..1	0..1	0..1
Other Attributes below the Root Element																

▽

[4] don't care
[5] This is required by [TPS_SWCT_01179].

△

Attributes of SwDataDefProps	Root Elem.			Attribute Existence per Category												
	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP	CUBOID	CUBE_4	CUBE_5
element: ApplicationRecordElement	x	x	x			1..*										
element: ApplicationArrayElement	x	x	x				1									
ApplicationArrayElement.arraySizeSemantics	x						0..1									
ApplicationArrayElement.maxNumberOfElements	x						1									

This rule shall be imposed **at any time in the workflow**.

⌋*()*

This list makes use of the SwDataDefProps and other meta-model elements which are explained in detail in the further sections of this chapter.

5.2.4.1 Application Primitive Data Types

5.2.4.1.1 Data Types for Single Values

In contrast to prior versions (R3.x) of the AUTOSAR standard, the primitive application data types on M2 level are no longer specified. Instead of this, the meta-class ApplicationPrimitiveDataType in combination with the attached swDataDefProps is used on the level of the M2 (meta-) model to specify the details on M1 modeling level.

[TPS_SWCT_01242] category characterizes the nature of a data type on application level ⌈The category is used in addition to characterize the nature of a data type on application level.⌋*(RS_SWCT_03216)*

For example, the IntegerType as of AUTOSAR R3.x allows for specifying lower and upper ranges that constrain the applicable value interval. That aspect is still supported by this version of AUTOSAR, but the meta-model is different from the former approach. Especially it is no more considered of importance to specify that an ApplicationPrimitiveDataType is actually represented by “integer” numbers.

Figure 5.4 provides a sketch of how limits are defined now. The key feature is the aggregation of SwDataDefProps at AutosarDataType. The meta-class SwDataDefProps allows for creating a reference to a DataConstr that in turn aggregates a DataConstrRule.

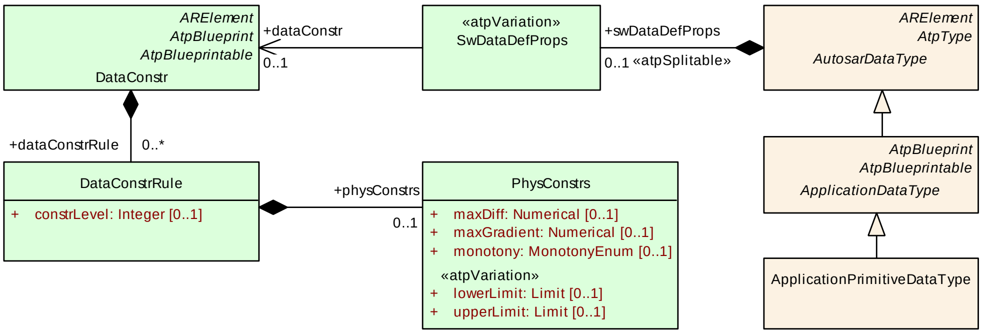


Figure 5.4: Specification of Physical Limits

The latter aggregates PhysConstrs and this meta-class finally owns two Limits in the roles lowerLimit and upperLimit.

Another example is shown in Figure 5.5. By making again use of SwDataDefProps, this figure shows how semantics in form of a CompuMethod and a Unit can be attached.

Also, an initValue can be defined which is used by the RTE in order to initialize values of VariableDataPrototypes/ParameterDataPrototypes defined locally in a software-component.

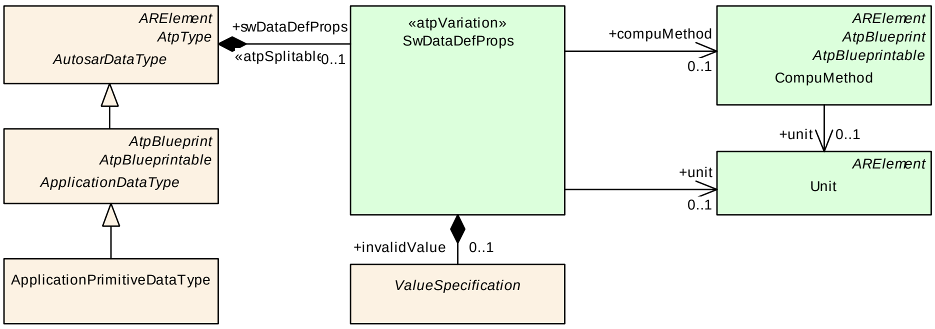


Figure 5.5: Some Properties of ApplicationPrimitiveDataTypes

Figure 5.6 (which is is only applicable for linear and rational CompuMethods) illustrates the relationship between the data constraints for ApplicationDataType, CompuMethod, ImplementationDataType, BaseType for the case of an entirely linear or rational conversion.

Please note that the relation of CompuMethod to invalidValue is discussed in detail in section 5.4.2.

If an ApplicationPrimitiveDataType does not define dataConstr, then implicit constraints can be derived from physical meaning of the ApplicationDataType.

For example, if the data type represents a temperature the lower bound will not be able to exceed 0K.

For other physical meanings, it could be possible that the implicitly assumed limits go from `-INF` to `+INF`.

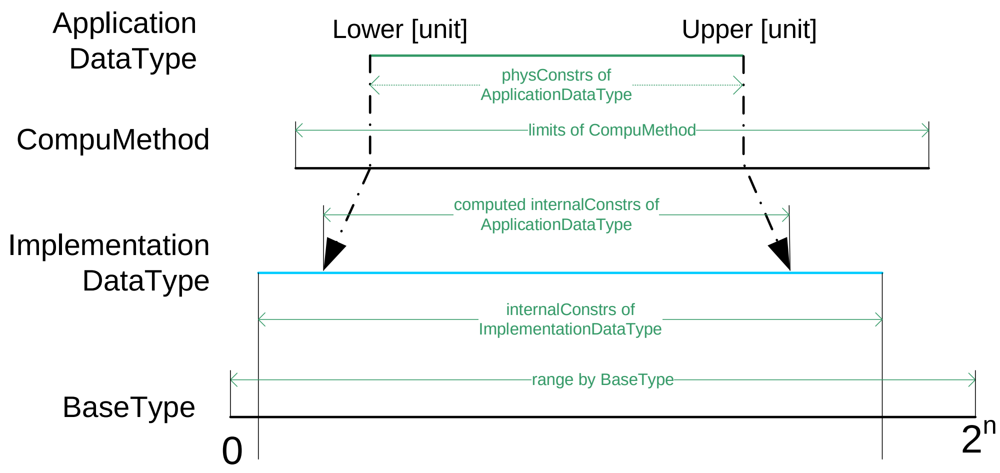


Figure 5.6: Value ranges for linear and rational `CompuMethod`

In order to avoid ambiguity regarding the values of limits it is **strongly recommended** defining a reasonable limit for `ApplicationPrimitiveDataType`s.

[constr_2544] Limits need to be consistent ⌈

- The limits of `ApplicationDataType` shall be inside the definition range of the `CompuMethod`

  The `CompuMethod` needs to be applicable for limits of an `ApplicationDataType`. The reason is that the internal representation of the limits for the `ApplicationDataType` are calculated by applying the `CompuMethod`.

- The such defined internal limits of the `ApplicationDataType` shall be within or equal the `internalConstrs` of the mapped `ImplementationDataType`.

- The limits of the `ImplementationDataType` shall be within or equal to the limits defined by the size of the `BaseType`.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

For a more detailed description of the properties that can be defined for data types (and data prototypes as well), see sections 5.4. The handling of `invalidValue` is explained in more detail in section 5.4.2.

[TPS_SWCT_01760] Defining the dimension of an ApplicationPrimitiveDataType of category VAL_BLK ⌈An ApplicationPrimitiveDataType of category VAL_BLK that has only one dimension shall be described using the attribute SwDataDefProps.swValueBlockSize.

An ApplicationPrimitiveDataType of category VAL_BLK that has more than one dimension shall be described using the attribute SwDataDefProps.swValueBlockSizeMult.⌋*(RS_SWCT_03216)*

[constr_1610] Existence of SwDataDefProps.swValueBlockSize and SwDataDefProps.swValueBlockSizeMult ⌈Attributes SwDataDefProps.swValueBlockSize and SwDataDefProps.swValueBlockSizeMult shall not exist at the same time in the context of a given SwDataDefProps.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

5.2.4.1.2 About Enumerations

[TPS_SWCT_01243] Definition of enumeration types ⌈In the AUTOSAR meta-model, an enumeration is not implemented by means of an ApplicationCompositeDataType.

Instead, a discrete set of integer numbers can be used as a structural description for a single ApplicationPrimitiveDataType or an ImplementationDataType of category VALUE or TYPE_REFERENCE that boils down to an ImplementationDataType of category VALUE.

The mapping of the integer numbers to *labels* in the scope of the definition of an enumeration is considered part of the semantical definition via an attached CompuMethod rather than part of the structural description.⌋*(RS_SWCT_03216)*

[TPS_SWCT_01562] Specification of values of an enumeration ⌈For the specification of values of an enumeration on the basis of the labels defined in the applicable CompuMethod it is necessary to distinguish two approaches based on the used AutosarDataType:

- ImplementationDataType: as mentioned by [constr_1225], the definition of the labels of an enumeration shall only be done by using TextValueSpecification.
- ApplicationPrimitiveDataType: use the ApplicationValueSpecification.swValueCont.swValuesPhys.vt or ApplicationRuleBasedValueSpecification.swValueCont.ruleBasedValues.arguments.vt.

⌋*(RS_SWCT_03216)*

The relevant meta-classes in the context of SwDataDefProps are sketched in Figure 5.7. This includes all meta-classes that may contribute to the definition of the symbol of a CompuScale in C code, see [TPS_SWCT_01431].

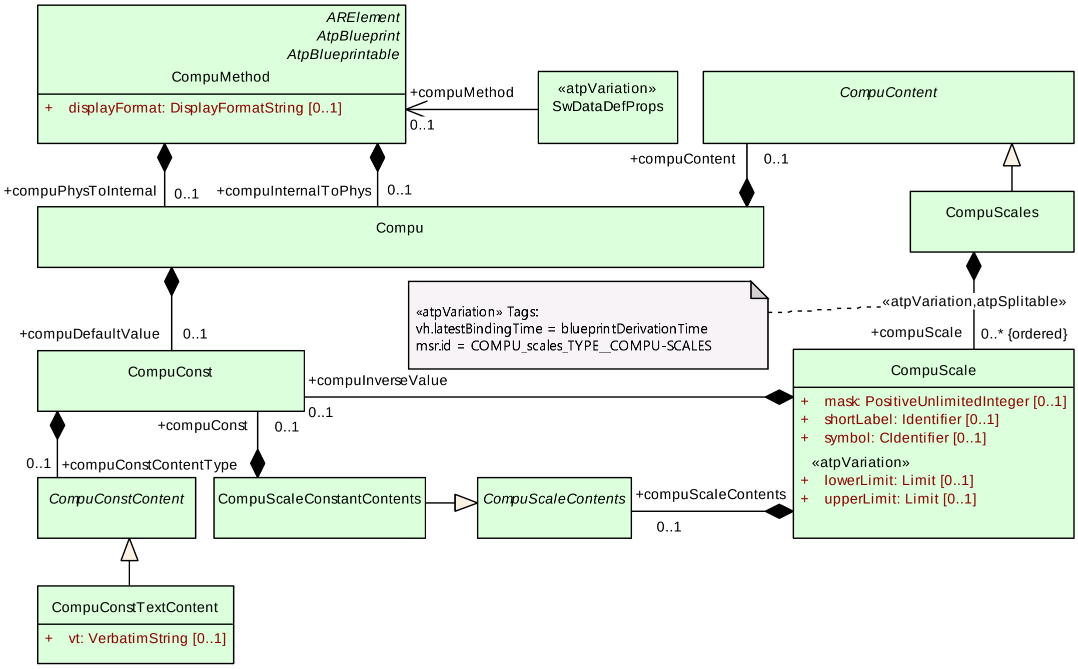


Figure 5.7: Relevant meta-classes for the specification of enumerations

An example of how an enumeration looks like in ARXML is contained in section 5.5.1.4.

5.2.4.1.3 Data Types for Calibration Parameters

[TPS_SWCT_01244] Data types for calibration parameters are also described as primitive types ⌈Data types for calibration parameters are from the application perspective also described as primitive types. This is obvious, if they are simple values (category VALUE). Also, the category STRING is treated as a primitive type on application level.

Less obvious is the fact that ApplicationDataTypes of the categories VAL_BLK, COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, and CUBE_5 are not described as composite data types (as far as the application level is concerned) although they admittedly possess some kind of internal structure.

In contrast to ApplicationCompositeDataTypes, they are **not** composed similarly of other AutosarDataTypes. Their substructure needs a special description in order to be compatible with existing calibration techniques.⌋*(RS_SWCT_03216)*

[TPS_SWCT_01245] SwDataDefProps control the structure of calibration parameters ⌈The substructure of these types is attached to the SwDataDefProps. By this means it is possible to define on the level of DataPrototypes or other artifacts, where the SwDataDefProps come into play.⌋*(RS_SWCT_03216)*

For details on these part of the SwDataDefProps see chapters 5.4.4 and 5.5.5.

5.2.4.1.4 Data Types for Textual Strings

[constr_1093] Definition of textual strings ⌈An ApplicationPrimitiveDataType of category STRING shall have a swTextProps which determines the arraySizeSemantics and swMaxTextSize.

This rule shall be imposed **at the time when the contract phase generation is executed**⌋*()*

[TPS_SWCT_01488] ApplicationPrimitiveDataType shall be interpreted as a string of a particular encoding ⌈To indicate that an ApplicationPrimitiveDataType shall be interpreted as a string of a particular encoding it shall reference swDataDefProps.swTextProps.baseType and the only attribute of the referenced SwBaseType relevant for this purpose is the BaseTypeDirectDefinition.baseTypeEncoding.⌋*()*

[constr_1905] Existence of attribute SwTextProps.arraySizeSemantics ⌈For each SwTextProps, attribute arraySizeSemantics shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1906] Existence of attribute SwTextProps.swMaxTextSize ⌈For each SwTextProps, attribute swMaxTextSize shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01127] Byte array with variable size ⌈SwTextProps can be used to define byte arrays of variable size.⌋*(RS_SWCT_03182, RS_SWCT_03181)*

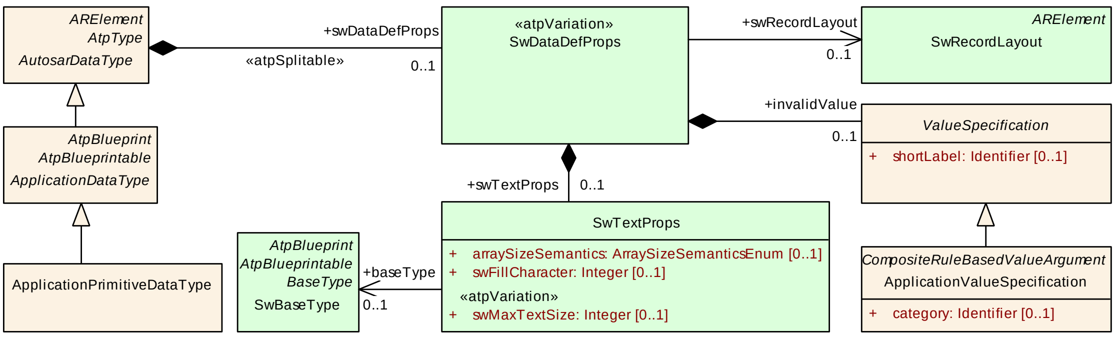


Figure 5.8: Specification of textual strings

[TPS_SWCT_01246] SwRecordLayout may also be required for A2L generation ⌈A SwRecordLayout may also be required for the generation of A2L if the string is part of calibration data.⌋*()*

As stated by [TPS_SWCT_01128], the definition of SwDataDefProps.swRecordLayout is considered mandatory anyway for ApplicationPrimitiveDataTypes of category STRING.

Class	SwTextProps			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This meta-class expresses particular properties applicable to strings in variables or calibration parameters.			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swTextProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the semantics of the arraysize for the array representing the string in an Implementation DataType. It is there to support a safe conversion between ApplicationDatatype and ImplementationDatatype, even for variable length strings as required e.g. for Support of SAE J1939.
baseType	SwBaseType	0..1	ref	This is the base type of one character in the string. In particular this baseType denotes the intended encoding of the characters in the string on level of ApplicationData Type. **Tags:**xml.sequenceOffset=30
swFillCharacter	Integer	0..1	attr	Filler character for text parameter to pad up to the maximum length swMaxTextSize. The value will be interpreted according to the encoding specified in the associated base type of the data object, e.g. 0x30 (hex) represents the ASCII character zero as filler character and 0 (dec) represents an end of string as filler character. The usage of the fill character depends on the arraySize Semantics. **Tags:**xml.sequenceOffset=40
swMaxTextSize	Integer	0..1	attr	Specifies the maximum text size in characters. Note the size in bytes depends on the encoding in the corresponding baseType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table 5.8: SwTextProps

The following series of XML fragments exemplifies the definition of a data type for the representation of a textual string. First, the applicable ApplicationPrimitiveDataType is defined (see Figure 5.8).

Note that the category is set to the value STRING. Also, the ApplicationPrimitiveDataType.swDataDefProps.swTextProps indicate the width of the string and also define (by means of the reference to baseType) the encoding this string data type is supposed to utilize.

Note further that the fact that an ApplicationDataType directly references (across the implementation level) to a SwBaseType **represents an exception to the rule that ApplicationDataType should not be concerned about the lowest level of data type definition** in AUTOSAR.

If the bridging of the implementation level were accepted as a general pattern for the modeling of ApplicationDataType it would easily be possible to bypass the implementation level to some extent and this would render ApplicationDataTypes less versatile.

[TPS_SWCT_01128] SwRecordLayout needed for ApplicationPrimitiveDataType of category STRING ⌈As mentioned in [TPS_SWCT_01179], an ApplicationPrimitiveDataType of category STRING is considered a Compound Primitive Data Type.

Therefore, it needs a reference to the definition of a SwRecordLayout that presets the approach for creating a matching ImplementationDataType.⌋*()*

In this specific example the definition of the SwRecordLayout foresees the ApplicationPrimitiveDataType of category STRING to be implemented as a structured data type that consists of:

1. the **size** of an instance of the string data type in terms of the number of characters plus
2. an **array** that can be used to store the individual characters contained in an instance of the string data type.

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyApplicationStringType</SHORT-NAME>
  <CATEGORY>STRING</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-TEXT-PROPS>
          <ARRAY-SIZE-SEMANTICS>VARIABLE-SIZE</ARRAY-SIZE-SEMANTICS>
          <SW-MAX-TEXT-SIZE>50</SW-MAX-TEXT-SIZE>
          <BASE-TYPE-REF BASE="default" DEST="SW-BASE-TYPE">BaseTypes/
              MyTextBaseType</BASE-TYPE-REF>
        </SW-TEXT-PROPS>
        <INVALID-VALUE>
          <APPLICATION-VALUE-SPECIFICATION>
            <CATEGORY>STRING</CATEGORY>
            <SW-VALUE-CONT>
              <SW-VALUES-PHYS>
                <VT>inv</VT>
              </SW-VALUES-PHYS>
            </SW-VALUE-CONT>
          </APPLICATION-VALUE-SPECIFICATION>
        </INVALID-VALUE>
        <SW-RECORD-LAYOUT-REF BASE="default" DEST="SW-RECORD-LAYOUT">
            RecordLayouts/StringDescriptor</SW-RECORD-LAYOUT-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.1: Example for the definition of a string ApplicationPrimitiveDataType

Depending on the used encoding the **array** may need to be bigger (in terms of the number of elements) than the corresponding value of the **size**.

Furthermore, the definition of the SwRecordLayout already takes into account that the implementation of an array data type by means of an ImplementationDataType requires the definition of an ImplementationDataTypeElement.

The meaning of the standardized values of SwRecordLayoutV.swRecordLayoutVProp are documented in [TPS_SWCT_01489]. In the scope of this example the values COUNT and VALUE are used.

The fact that the swRecordLayoutGroupTo contains the value -1 means that the iteration ends at the last element of the array.

Please note further that the discussed example of an ApplicationPrimitiveDataType of category STRING also contains the definition of an invalidValue for the string data type.

The next step is the definition of an ImplementationDataType that represents the string type on the implementation level.

The definition of the ImplementationDataType can be derived from the definition of the applicable SwRecordLayout.

```
<SW-RECORD-LAYOUT>
  <SHORT-NAME>StringDescriptor</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">String by descriptor</L-4>
  </LONG-NAME>
  <INTRODUCTION>
    <VERBATIM>
      <L-5 L="EN" xml:space="default">
struct{
  size,
  char[]
}
      </L-5>
    </VERBATIM>
  </INTRODUCTION>
  <SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-V>
      <SHORT-LABEL>size</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-AXIS>STRING</SW-RECORD-LAYOUT-V-AXIS>
      <SW-RECORD-LAYOUT-V-PROP>COUNT</SW-RECORD-LAYOUT-V-PROP>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-GROUP>
      <SHORT-LABEL>chars</SHORT-LABEL>
      <SW-RECORD-LAYOUT-GROUP-AXIS>STRING</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>0</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SHORT-LABEL>char</SHORT-LABEL>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
```

```
    </SW-RECORD-LAYOUT-GROUP>
</SW-RECORD-LAYOUT>
```

Listing 5.2: Example for the definition of a SwRecordLayout applicable for an ApplicationPrimitiveDataType of category STRING

The next listing describes the data type of one character within the string data type.

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint8</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">BaseTypes/uint8BT</BASE-TYPE-REF
            >
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 5.3: Example for the definition of a possible character data type for an ImplementationDataType that represents as string

Please note that the ImplementationDataType **also** defines an invalidValue. As mentioned in [TPS_SWCT_01487], the consistency of the invalidValue defined in the scope of the ApplicationPrimitiveDataType of category STRING and the invalidValue defined in the scope of the corresponding ImplementationDataType cannot formally be checked.

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>MyImplementationStringType</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <INVALID-VALUE>
          <RECORD-VALUE-SPECIFICATION>
            <FIELDS>
              <NUMERICAL-VALUE-SPECIFICATION>
                <VALUE>3</VALUE>
              </NUMERICAL-VALUE-SPECIFICATION>
              <ARRAY-VALUE-SPECIFICATION>
                <ELEMENTS>
                  <NUMERICAL-VALUE-SPECIFICATION>
                    <VALUE>105</VALUE>
                  </NUMERICAL-VALUE-SPECIFICATION>
                  <NUMERICAL-VALUE-SPECIFICATION>
                    <VALUE>110</VALUE>
                  </NUMERICAL-VALUE-SPECIFICATION>
                  <NUMERICAL-VALUE-SPECIFICATION>
                    <VALUE>118</VALUE>
                  </NUMERICAL-VALUE-SPECIFICATION>
                </ELEMENTS>
              </ARRAY-VALUE-SPECIFICATION>
            </FIELDS>
```

```
          </RECORD-VALUE-SPECIFICATION>
        </INVALID-VALUE>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <DYNAMIC-ARRAY-SIZE-PROFILE>VSA_LINEAR</DYNAMIC-ARRAY-SIZE-PROFILE>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>size</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">
                ImplementationDataTypes/uint8</IMPLEMENTATION-DATA-TYPE-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>string</SHORT-NAME>
      <CATEGORY>ARRAY</CATEGORY>
      <SUB-ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
          <SHORT-NAME>character</SHORT-NAME>
          <CATEGORY>TYPE_REFERENCE</CATEGORY>
          <ARRAY-SIZE>200</ARRAY-SIZE>
          <ARRAY-SIZE-HANDLING>ALL-INDICES-SAME-ARRAY-SIZE</ARRAY-SIZE-
              HANDLING>
          <ARRAY-SIZE-SEMANTICS>VARIABLE-SIZE</ARRAY-SIZE-SEMANTICS>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-
                    TYPE">ImplementationDataTypes/uint8</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-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 5.4: Example for the definition of a string ImplementationDataType

The ImplementationDataType with the shortName MyImplementation-StringType (as exemplified in Listing 5.4) represents a variable-size string, i.e. the number of characters may change at run-time. Of course, it is also possible to model a fixed-size string, but in that case a size-indicator is not required.

Please note further that the size of the payload array in the definition of the ImplementationDataType in Listing 5.4 has been set to the value 200 in order to accommodate for the definition of swMaxTextSize in the definition of the corresponding

ApplicationDataType in combination with the fact that the value of baseTypeEncoding has been set to UTF-8.

For background, the value of attribute SwTextProps.swMaxTextSize shall be specified as the number of *code points* in the string.

Each *code point* will be encoded by a sequence of bytes, depending on the applicable encoding. In the case of UTF-8, for example, each *code point* will be encoded by up to four bytes.

On the level of ImplementationDataType, an array designed to hold a string consisting of *code points* encoded using UTF-8 needs to be big enough to carry the number of *code points* (which may have been described by SwTextProps.swMaxTextSize) times 4 bytes.

The interesting part about this definition is the fact that on the implementation level, it was (driven by the definition of the SwRecordLayout) decided to implement the string as a structure of a size element (that goes by the shortName “size”) and a value element (that goes by the shortName “string”).

The value element is defined as an array data type and therefore has a sub-element that goes by the shortName “character”.

The latter references (in the role swDataDefProps.implementationDataType) the Platform Data Type “uint8” (that, according to the rules of Platform Data Types, is realized by an ImplementationDataType “uint8”).

Please note that the ApplicationPrimitiveDataType named “MyApplicationStringType” references the SwBaseType named “MyTextBaseType” which is defined in the following XML fragment:

```
<SW-BASE-TYPE>
  <SHORT-NAME>MyTextBaseType</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>UTF-8</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
<SW-BASE-TYPE>
  <SHORT-NAME>uint8BT</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
</SW-BASE-TYPE>
```

Listing 5.5: Example for the definition of a string SwBaseType

The contribution of this definition of SwBaseType to the overall definition of a string data type is represented by the definition of the character encoding (which is set to UTF-8).

```
<DATA-TYPE-MAPPING-SET>
  <SHORT-NAME>theExample</SHORT-NAME>
  <DATA-TYPE-MAPS>
    <DATA-TYPE-MAP>
```

```
        <APPLICATION-DATA-TYPE-REF BASE="default" DEST="APPLICATION-PRIMITIVE
            -DATA-TYPE">ApplicationDataTypes/MyApplicationStringType</
            APPLICATION-DATA-TYPE-REF>
        <IMPLEMENTATION-DATA-TYPE-REF BASE="default" DEST="IMPLEMENTATION-
            DATA-TYPE">ImplementationDataTypes/MyImplementationStringType</
            IMPLEMENTATION-DATA-TYPE-REF>
      </DATA-TYPE-MAP>
    </DATA-TYPE-MAPS>
</DATA-TYPE-MAPPING-SET>
```

Listing 5.6: Example for the definition of the applicable DataTypeMappingSet

However, there is still one important part missing, i.e. the definition of the mapping of ApplicationPrimitiveDataType to ImplementationDataType (and vice versa, see Listing 5.6).

As mentioned before, the definition of an ImplementationDataType that corresponds to an ApplicationPrimitiveDataType of category STRING can be to some extent derived from the ApplicationPrimitiveDataType.swDataDefProps.swRecordLayout.

[TPS_SWCT_01570] DataTypeMap is mandatory in the presence of ApplicationPrimitiveDataType.swDataDefProps.swRecordLayout ⌈The definition of a DataTypeMap is mandatory even if an ImplementationDataType has been derived from an ApplicationPrimitiveDataType that defines a SwRecordLayout.⌋*()*

One motivation for the existence of [TPS_SWCT_01570] is that the integrator of an AUTOSAR ECU may rightfully decide to take a different ImplementationDataType other than the one that has been generated on the basis of the SwRecordLayout.

5.2.4.2 Application Composite Data Types

[TPS_SWCT_01247] ApplicationArrayDataType and ApplicationRecordDataType ⌈The meta-classes ApplicationArrayDataType and ApplicationRecordDataType provide the means to define composite data types.

Such a composite data type is required if the application software wants to have access to the individual elements of the composite as well as to do operations with the whole composite, e.g. wants to communicate the complete record or array in a single transaction.

It is possible to use a combination of ApplicationArrayDataType and ApplicationRecordDataType, so that an ApplicationArrayDataType could be defined as ApplicationRecordElement of a ApplicationRecordDataType and in the same manner a ApplicationRecordDataType could be used as the base-type of an ApplicationArrayDataType. The creation of nested ApplicationCompositeDataTypes is also possible.⌋*(RS_SWCT_03215, RS_SWCT_03216)*

Details about meta-class ApplicationRecordDataType are depicted in Figure 5.9.

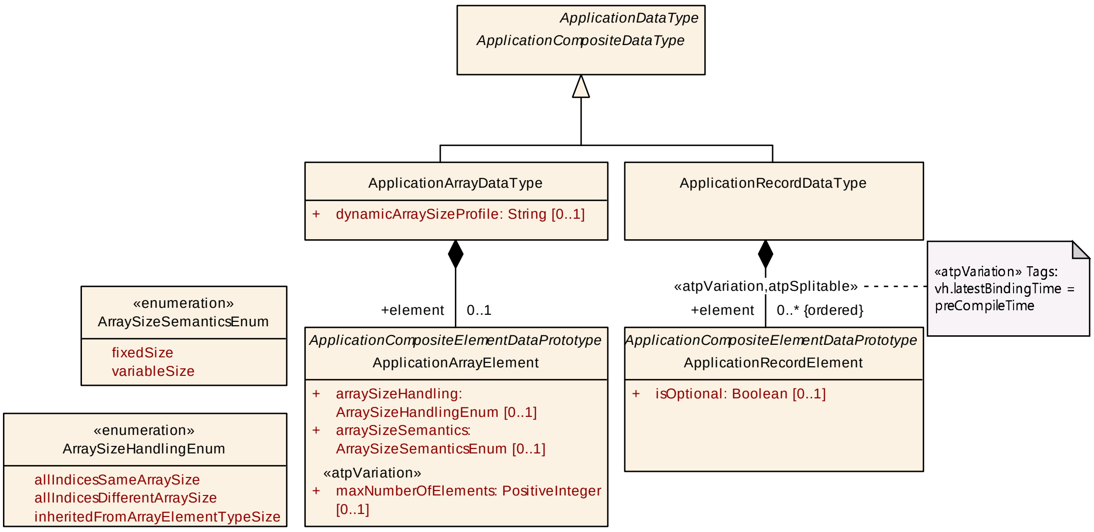


Figure 5.9: Summary of ApplicationCompositeDataType

5.2.4.2.1 ApplicationArrayDataType

[TPS_SWCT_01078] Configurable array size ⌈The size of an ApplicationArrayDataType (in terms of the number of elements) is configured by means of the value of attribute ApplicationArrayElement.maxNumberOfElements.

For the purpose of referring to an element of an ApplicationArrayDataType within a software-component description, the element's index runs from 0 to the value of maxNumberOfElements-1.⌋*(RS_SWCT_03144, RS_SWCT_03215)*

Class	**ApplicationArrayDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement, ARObject, ApplicationCompositeDataType, ApplicationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArrayElement	0..1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.

Table 5.9: ApplicationArrayDataType

[constr_1907] Existence of attribute ApplicationArrayDataType.element ⌈For each ApplicationArrayDataType, the aggregation of ApplicationArrayElement in the role element shall exist **at the time when the RTE is generated**.⌋*()*

Class	ApplicationArrayElement			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of the elements of an application array data type.			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationArrayDataType.element, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls how the information about the array size shall be interpreted.
indexDataType	ApplicationPrimitive DataType	0..1	ref	This reference can be taken to assign a CompuMethod of category TEXTTABLE to the array. The texttable entries associate a textual value to an index number such that the element with that index number is represented by a symbolic name.
maxNumberOf Elements	PositiveInteger	0..1	attr	The maximum number of elements that the array can contain. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 5.10: ApplicationArrayElement

Please note that the information about the number of elements of a specific ApplicationArrayDataType is not absolute but allows for further interpretation.

[TPS_SWCT_01076] Number of elements of a specific ApplicationArrayDataType might vary at run-time ⌈That is, there are cases where the number of elements of a specific ApplicationArrayDataType might vary at run-time.

To be precise, the number of elements might vary between 0 and the value denoted by maxNumberOfElements.

For this purpose an additional attribute arraySizeSemantics is available that can be used to clarify the meaning of maxNumberOfElements.

For clarification, it might indeed happen that the actual number of elements in a specific ApplicationArrayDataType yields 0 simply because the respective DataPrototype is part of a higher-level protocol where under certain circumstances the DataPrototype of ApplicationArrayDataType is simply not required for expressing a given semantics.⌋*(RS_SWCT_03180, RS_SWCT_03181, RS_SWCT_03215, RS_SWCT_03144)*

Please note that the ability to define the semantic meaning of maxNumberOfElements is not only limited to the application data type level. The same approach also applies for ImplementationDataType.

Enumeration	ArraySizeSemanticsEnum
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	This type controls how the information about the number of elements in an ApplicationArrayDataType is to be interpreted.

▽

△

Enumeration	**ArraySizeSemanticsEnum**
Aggregated by	ApplicationArrayElement.arraySizeSemantics, DiagnosticDataElement.arraySizeSemantics, ImplementationDataTypeElement.arraySizeSemantics, SwTextProps.arraySizeSemantics
Literal	***Description***
fixedSize	This means that the ApplicationArrayDataType will always have a fixed number of elements. **Tags:**atp.EnumerationLiteralIndex=0
variableSize	This implies that the actual number of elements in the ApplicationArrayDataType might vary at run-time. The value of arraySize represents the maximum number of elements in the array. **Tags:**atp.EnumerationLiteralIndex=1

Table 5.11: ArraySizeSemanticsEnum

[constr_1152] `category` of `ApplicationArrayElement` and `AutosarDataType` referenced in the role `type` shall be kept in sync ⌈The value of `category` of an `ApplicationArrayElement` shall always be identical to the value of `category` of the `AutosarDataType` referenced by the `ApplicationArrayElement`.⌋*()*

Enumeration	**ArraySizeHandlingEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes
Note	This enumeration defines different ways to handle the sizes of variable size arrays.
Aggregated by	ApplicationArrayElement.arraySizeHandling, ImplementationDataTypeElement.arraySizeHandling
Literal	***Description***
allIndicesDifferent ArraySize	All elements of the variable size array may have different sizes. **Tags:**atp.EnumerationLiteralIndex=0
allIndicesSame ArraySize	All elements of the variable size array have the same size. **Tags:**atp.EnumerationLiteralIndex=1
inheritedFromArray ElementTypeSize	The size of all dimensions of the variable size array is determined by the size of the contained array element. **Tags:**atp.EnumerationLiteralIndex=2

Table 5.12: ArraySizeHandlingEnum

5.2.4.2.1.1 Variable Size Array

[TPS_SWCT_01604] Enable `Size Indicator` ⌈To enable the RTE's ability to consider the number of valid elements inside a `Variable-Size Array Data Type` the `ApplicationArrayDataType.dynamicArraySizeProfile` of `ApplicationArrayDataType` and `ApplicationArrayElement.arraySizeHandling` shall be set.⌋*(RS_SWCT_03181)*

[TPS_SWCT_01601] `Size Indicator` shall be updated by software-component ⌈If a software-component changes the number of valid elements in a variable size array, it shall also update the `Size Indicator` in the `ImplementationDataType`.⌋*(RS_SWCT_03181)*

[TPS_SWCT_01602] Size Indicator shall be read by the software-component ⌈If a software-component receives a variable size array, it shall use the Size Indicator in the ImplementationDataType to determine the number of valid elements in the array.⌋*(RS_SWCT_03181)*

[TPS_SWCT_01605] Semantics of ApplicationArrayElement.arraySizeHandling ⌈The attribute ApplicationArrayElement.arraySizeHandling specifies how the size is determined in case of multi-dimensional variable size array.⌋*(RS_SWCT_03181)*

This allows to specify coherent relations between the sizes of the nested variable size arrays in case of multiple dimensions.

With a suitable ImplementationDataType, it is possible to enable other software-components, RTE, and other BSW modules to make use of the Size Indicator and only transfer the valid data elements from the sender to the receiver.

[TPS_SWCT_01606] Internal structure of mapped ImplementationDataType ⌈The attribute dynamicArraySizeProfile specifies which internal structure the ImplementationDataType that is mapped to the ApplicationDataType shall follow.⌋*(RS_SWCT_03181)*

[TPS_SWCT_01607] Profiles for internal structure of mapped ImplementationDataType ⌈For the structure of the ImplementationDataType that is mapped to the ApplicationDataType the following profiles are defined for dynamicArraySizeProfile: VSA_LINEAR, VSA_SQUARE, VSA_RECTANGULAR, and VSA_FULLY_FLEXIBLE.⌋*(RS_SWCT_03181)*

[TPS_SWCT_01608] Custom profiles for internal structure of mapped ImplementationDataType ⌈Custom profiles can be added to dynamicArraySizeProfile. They shall have a company-specific prefix.⌋*(RS_SWCT_03181)*

As it is a general rule for the definition of custom profiles or values of category, the custom value should start with a company-specific prefix in order to avoid clashes with later extensions of the AUTOSAR standard.

dynamicArraySizeProfile is used to specify how the number of elements of the multiple dimensions of a variable size array correlate. They could be totally independent (VSA_FULLY_FLEXIBLE) on the one hand or each dimension has the same number of valid elements (VSA_SQUARE).

[TPS_SWCT_01623] Justification for the existence of attributes ApplicationArrayDataType.dynamicArraySizeProfile and ApplicationArrayElement.arraySizeHandling ⌈At the first glance, the two attributes ApplicationArrayDataType.dynamicArraySizeProfile and ApplicationArrayElement.arraySizeHandling seem equivalent.

However, both are needed because they have to be used if multi dimensional variable size arrays have to be described. In this case, multiple combinations of sizes could occur which cannot be specified beforehand.⌋*(RS_SWCT_03181)*

The ImplementationDataType has to follow certain rules depending on the chosen profile. See chapter 5.2.5 for details.

[constr_1314] Profile VSA_LINEAR for ApplicationArrayDataType ⌈If the dynamicArraySizeProfile of ApplicationArrayDataType is set to VSA_LINEAR, the contained ApplicationArrayElement shall fulfill **all** the following conditions **at the time when the contract phase generation is executed**:

- The attribute ApplicationArrayElement.arraySizeSemantics shall set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationDataType that is not an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists.

⌋*()*

The part of [constr_1314] that demands that *the ApplicationArrayElement shall be typed by an ApplicationDataType that is not an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists* basically boils down to the simple explanation that the “leaf” data type of the Variable-Size Array Data Type can be anything but a Variable-Size Array Data Type.

[constr_1315] Profile VSA_SQUARE for ApplicationArrayDataType ⌈If the dynamicArraySizeProfile of ApplicationArrayDataType is set to VSA_SQUARE, the contained ApplicationArrayElement shall fulfill **all** the following conditions :

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall not be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

The referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the “root” ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists.

The last ApplicationArrayDataType in that chain shall have an ApplicationArrayElement that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling set to the value allIndicesSameArraySize.

All ApplicationArrayDataTypes before shall have an ApplicationArrayElement that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall not be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The part of [constr_1315], [constr_1316], and [constr_1317] that demands that *the referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the "root" ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists* basically boils down to the simple explanation that the "leaf" data type of the Variable-Size Array Data Type can be anything but a Variable-Size Array Data Type.

[constr_1316] Profile VSA_RECTANGULAR for ApplicationArrayDataType ⌈If the dynamicArraySizeProfile of ApplicationArrayDataType is set to VSA_RECTANGULAR the contained ApplicationArrayElement shall fulfill **all** the following conditions :

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

- The ApplicationArrayElement shall be typed by an ApplicationArray-DataType.

The referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the “root” ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists.

The last ApplicationArrayDataType in that chain shall have an Application-ArrayElement that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

All ApplicationArrayDataTypes before shall have an ApplicationArrayEle-ment that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall set to the value variableSize
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationArray-DataType.

This rule shall be imposed v**at the time when the contract phase generation is executed**.⌋*()*

[constr_1317] Profile VSA_FULLY_FLEXIBLE for ApplicationArrayDataType
⌈If the dynamicArraySizeProfile of ApplicationArrayDataType is set to VSA_FULLY_FLEXIBLE, the contained ApplicationArrayElement shall fulfill **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.

- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

The referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the "root" ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exist.

The last ApplicationArrayDataType in that chain shall have an ApplicationArrayElement that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

All ApplicationArrayDataTypes before shall have an ApplicationArrayElement that fulfills **all** the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

For examples, see Appendix E.1.

5.2.4.2.1.2 Multi-Dimensional Arrays

Figure 5.10 shows a three-dimensional array described with a set of ApplicationArrayDataTypes on the left-hand side. The array element is typed by an ApplicationPrimitiveDataType of category BOOLEAN. On the right-hand side the implementation of the three-dimensional array is described with an ImplementationDataType which contains three nested ImplementationDataTypeElements.

Matching ApplicationArrayElements and ImplementationDataTypeElements are shown on the same layer. For the sake of clarity correlating maxNumberOfElements and arraySize attributes are described with the identical instance of a SwSystemconst instead of a value. Further details of variant rich M1 models are not in the scope of this example.

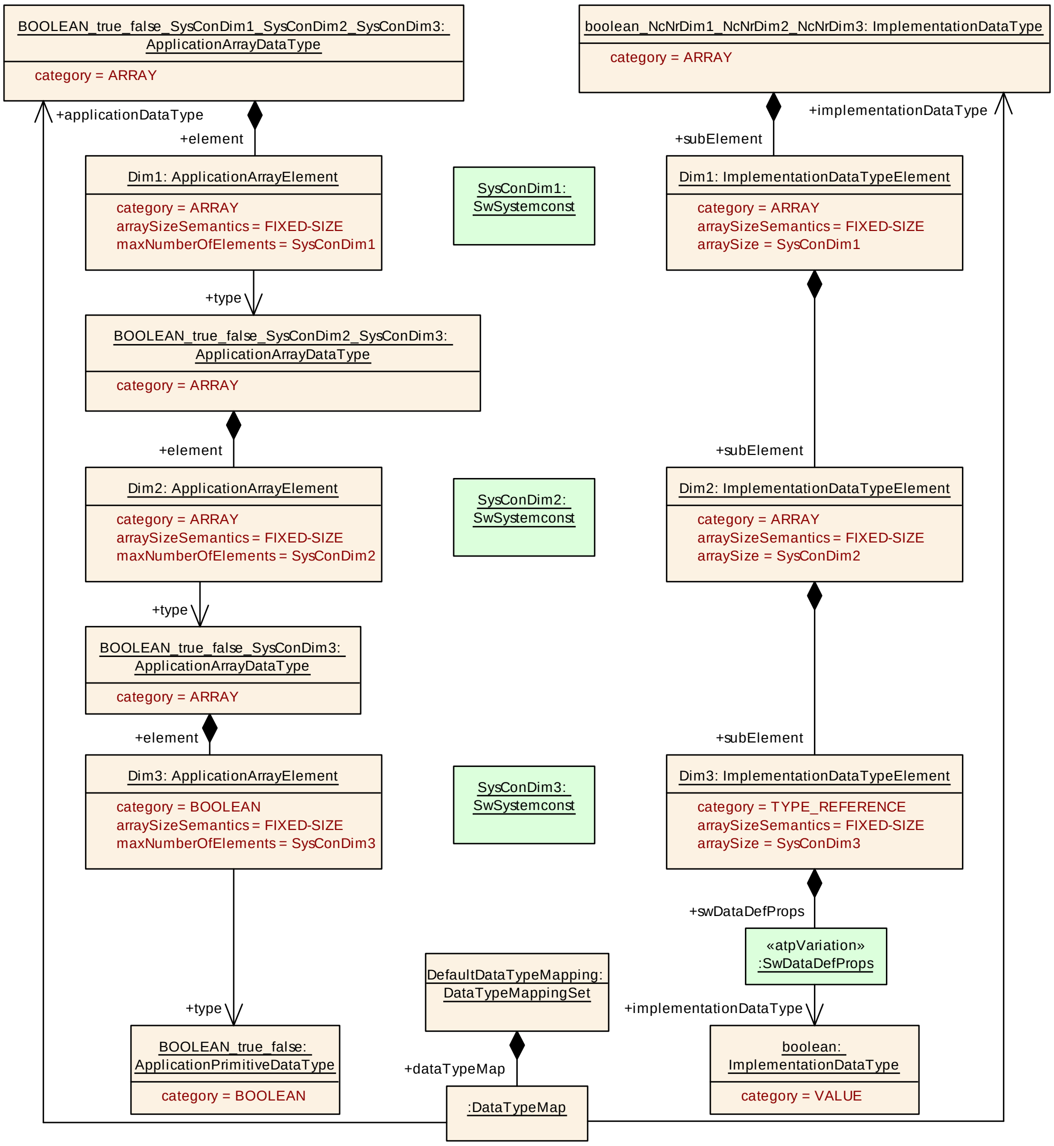


Figure 5.10: Example of a three-dimensional array type

The data type of the array element is described by the ApplicationArrayDataType with the means of a ApplicationPrimitiveDataType of category BOOLEAN. In order to fulfill [constr_1152] the category of ApplicationArrayElement “Dim3” is set to BOOLEAN.

This ApplicationPrimitiveDataType “BOOLEAN” correlates to the ImplementationDataType “boolean” of category VALUE which is typically the boolean type of the AUTOSAR Platform Types. Please note here [constr_1063].

[TPS_SWCT_01256] Definition of multi-dimensional array data types ⌈In order to describe multi dimensional arrays an ApplicationArrayElement references again another ApplicationArrayDataType. Hereby, one ApplicationArrayDataType per dimension is required.

This multiple dimensions do have a well-defined correlation to the individual dimensions of an ImplementationDataType of category ARRAY when the ApplicationArrayDataType is mapped to an ImplementationDataType.

The ApplicationArrayElements are mapping in the order of the ApplicationArrayElement to ApplicationArrayDataType references to ImplementationDataTypeElements in the order of first ImplementationDataTypeElement of the ImplementationDataType to leaf ImplementationDataTypeElement.

In other words the ApplicationArrayElement of the top-level ApplicationArrayDataType relates to the first ImplementationDataTypeElement of the ImplementationDataType.

The ApplicationArrayElement of the referenced ApplicationArrayDataTypes relates to the sub ImplementationDataTypeElements in the order of the ApplicationArrayElement to ApplicationArrayDataType references.⌋ *(RS_SWCT_03215, RS_SWCT_03216)*

5.2.4.2.1.3 Index Data Type

The usage of an array represents an elegant way to group data with identical properties. This allows for an easy processing of the same functionality by iterating over the array elements.

From a functional point of view, however, each array element may have a distinct meaning that could be visible to the application software. To create this visibility, it is possible to take advantage of an existing mechanism: CompuMethods of category TEXTTABLE.

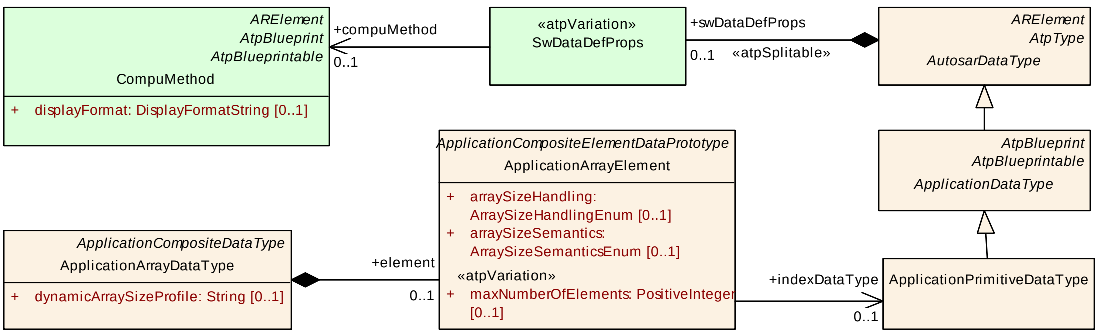


Figure 5.11: Modeling of the ApplicationArrayElement.indexDataType

[TPS_SWCT_01699] Usage of ApplicationArrayElement.indexDataType
⌈The primary use case of the attribute ApplicationArrayElement.indexDataType is the creation of composite data type mappings or the description of measurement and calibration. Furthermore, the information could be used for documentation purposes.⌋*(RS_SWCT_03230)*

[constr_1438] ApplicationArrayElement.indexDataType needs to refer to a CompuMethod of category TEXTTABLE ⌈The reference ApplicationArrayElement.indexDataType shall only point to an ApplicationPrimitiveDataType that in turn refers to a CompuMethod of category TEXTTABLE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1440] Size of the CompuMethod of category TEXTTABLE referenced by ApplicationArrayElement.indexDataType ⌈The interval defined by the CompuScales contained in the CompuMethod referenced by ApplicationArrayElement.indexDataType shall start at 0 and include all integer values until ApplicationArrayElement.maxNumberOfElements - 1.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1439] Requirements on ApplicationArrayElement if attribute indexDataType exists ⌈If ApplicationArrayElement.indexDataType exists then the attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value fixedSize and attribute arraySizeHandling shall not exist **at the time when the contract phase generation is executed**.⌋*()*

Listing 5.7 exemplifies the definition of an indexDataType.

```xml
<APPLICATION-ARRAY-DATA-TYPE>
  <SHORT-NAME>CylinderArray</SHORT-NAME>
  <ELEMENT>
    <SHORT-NAME>CylinderArrayElement</SHORT-NAME>
    <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
    <INDEX-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
       myIndexDataType</INDEX-DATA-TYPE-REF>
  </ELEMENT>
</APPLICATION-ARRAY-DATA-TYPE>
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>myIndexDataType</SHORT-NAME>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <COMPU-METHOD-REF DEST="COMPU-METHOD">cylinders</COMPU-METHOD-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.7: Example for array index data type

Listing 5.8 contains an example of a CompuMethod eligible for an indexDataType.

```xml
<COMPU-METHOD>
  <SHORT-NAME>cylinders</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>Cylinder1</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>Cylinder2</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">2</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>Cylinder3</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">3</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">3</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>Cylinder4</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Listing 5.8: Example for a compu method used by an array index data type

5.2.4.2.2 ApplicationRecordDataType

[TPS_SWCT_01249] **ApplicationRecordDataType** ⌈A declaration of ApplicationRecordDataType describes a non-empty set of objects, each of which has a unique identifier with respect to the ApplicationRecordDataType and each has an own ApplicationDataType.

The shortName of each ApplicationRecordElement within the scope of an ApplicationRecordDataType shall be unique.⌋*(RS_SWCT_03216)*

Class	ApplicationRecordDataType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
element (ordered)	ApplicationRecordElement	*	aggr	Specifies an element of a record. The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element.shortName, element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 5.13: ApplicationRecordDataType

[constr_1908] Existence of attribute ApplicationRecordDataType.element ⌈For each ApplicationRecordDataType, the aggregation of ApplicationRecordElement in the role element shall exist **at the time when the RTE is generated**.⌋*()*

Class	ApplicationRecordElement			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationRecordDataType.element, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ApplicationRecordElement as optional. This means the that, at runtime, the ApplicationRecordElement may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the ApplicationRecordElement as not valid at the sending end of a communication and determine its validity at the receiving end.

Table 5.14: ApplicationRecordElement

[TPS_SWCT_01771] Definition of optional elements on the level of ApplicationDataType ⌈The modeling approach for the definition of optional elements on the level of ApplicationDataType is to set the attribute ApplicationRecordElement.isOptional to the value true.

If the attribute is not set or set to the value false then the respective ApplicationRecordElement **shall be considered mandatory.**⌋*(RS_SWCT_03320)*

5.2.5 Implementation Data Type

5.2.5.1 Overview

[TPS_SWCT_01250] ImplementationDataType has been introduced to optimize the formal support for data type handling on the implementation level ⌈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 (please find an explanation of this term in [2]) of a software-component and the RTE.⌋*(RS_SWCT_03217)*

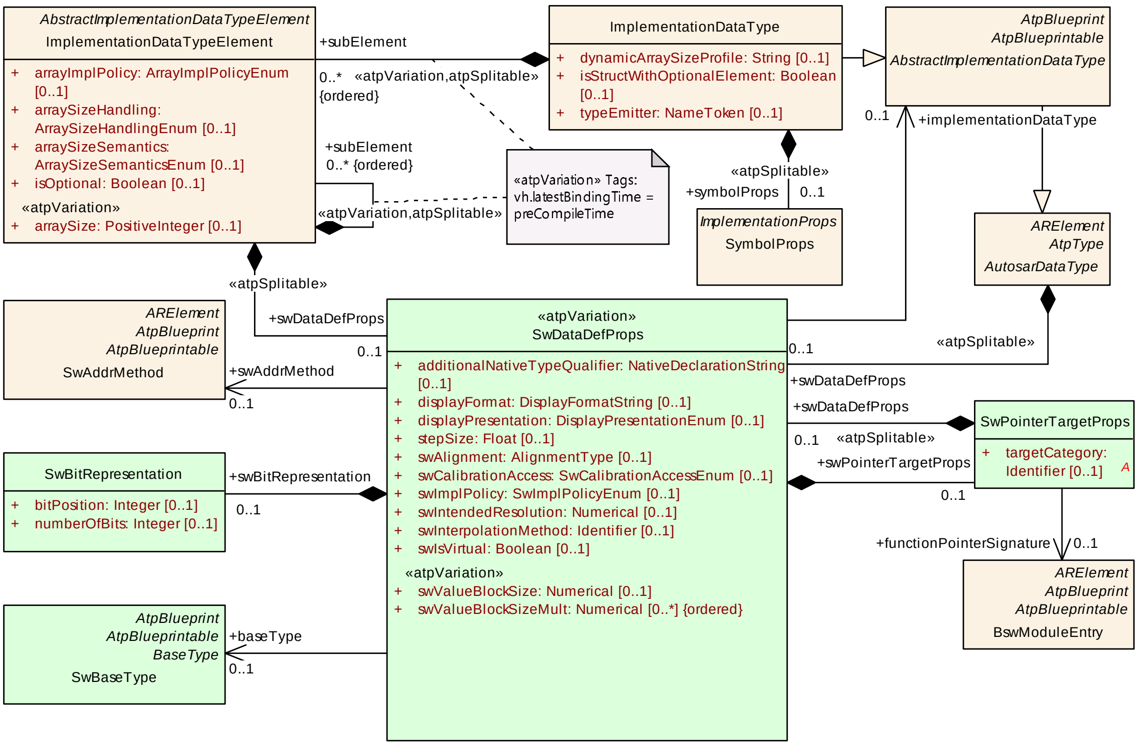


Figure 5.12: SwDataDefProps used in the context of ImplementationDataType

The allowed existence and multiplicity of all the attributes of SwDataDefProps and other properties depend on the category of the ImplementationDataType.

[constr_1178] Existence of attributes of SwDataDefProps in the context of ImplementationDataType ⌈For the sake of removing possible sources of ambiguity, SwDataDefProps used in the context of ImplementationDataType can **only have one of**

- baseType
- swPointerTargetProps
- implementationDataType

at the time when the contract phase generation is executed.⌋()

[TPS_SWCT_01251] Limited set of values for category are applicable for ImplementationDataType ⌈Like any AutosarDataType, also the data types on implementation level are characterized by its category and its SwDataDefProps. For a given category, only a limited set of attributes of the SwDataDefProps makes sense.⌋*(RS_SWCT_03217)*

[constr_10373] ImplementationDataType of category VALUE shall not refer to SwBaseType of category VOID ⌈An ImplementationDataType where attribute category is set to VALUE shall not reference (in the role swDataDefProps.baseType) a SwBaseType where attribute category is set to VOID.⌋()

Rationale for the existence of [constr_10373]: The programming language C does not support a variable of type void:

```
void * MyVoidPtr /* works */

void MyVoidVar; /* does not work!*/
```

[constr_1009] SwDataDefProps applicable to ImplementationDataTypes ⌈

Attributes of SwDataDefProps	Root Element				Attribute Existence per Category						
	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
additionalNativeTypeQualifier	x	x	x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1
annotation	x	x	x	x	*	*	*	*	*	*	*
baseType	x	x	x	x	1						
compuMethod	x	x	x	x	0..1			0..1			
dataConstr.dataConstrRule.physConstrs	x	x	x	x	d/c[6]			d/c			d/c
dataConstr.dataConstrRule.internalConstrs	x	x	x	x	0..1			0..1			0..1
displayFormat	x	x			0..1				0..1	0..1	0..1
displayPresentation	x	x			0..1						0..1
implementationDataType	x	x	x	x				1			

▽

[6] don't care

△

Attributes of SwDataDefProps	Root Element				Attribute Existence per Category						
	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
invalidValue	x	x	x		0..1			0..1	0..1[7]		0..1[8]
stepSize	x	x			0..1						
swAddrMethod	x	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x				0..1	0..1	0..1		0..1	0..1	0..1
swBitRepresentation											
swCalibrationAccess	x	x			0..1			0..1	0..1	0..1	0..1
swCalprmAxisSet											
swComparisonVariable											
swDataDependency											
swHostVariable											
swImplPolicy	x		x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution											
swInterpolationMethod											
swIsVirtual											
swPointerTargetProps	x	x	x	x		1	1				
swPointerTargetProps .swDataDefProps	x	x	x	x		1					
swPointerTargetProps .functionPointerSignature	x	x	x	x			1				
swRecordLayout											
swRefreshTiming	x	x	x	x	0..1				0..1	0..1	0..1
swTextProps											
swValueBlockSize											
swValueBlockSizeMult											
unit											
valueAxisDataType											
Other Attributes											
subElement: ImplementationDataTypeElement	x	x							1..*	1..*	1
subElement.arraySizeSemantics	x	x									0..1
subElement.arraySize	x	x									1

This rule shall be imposed **at any time in the workflow**.

[7]There is a use case for the definition of an invalidValue for category ARRAY and therefore category STRUCTURE is also supported for the sake of symmetry.

[8]This represents an exception such that it would make sense to use an entire ArrayValueSpecification as the invalidValue because a string semantically is more than just a bunch of characters in a row.

⌋()

[constr_1694] Allowed target of SwDataDefProps.implementationDataType ⌈The reference SwDataDefProps.implementationDataType shall only refer to an ImplementationDataType. Any other subclass of AbstractImplementationDataType is not supported as a reference target.

This rule shall be applied **at the time when the contract phase generation is executed**.⌋()

This list makes use of the SwDataDefProps and other meta-model elements which are explained in detail in the further sections of this chapter.

Regulations regarding the applicable categorys for attribute ImplementationDataType.swDataDefProps.compuMethod can be found in [constr_1158] inside section 5.5.1.3.2.

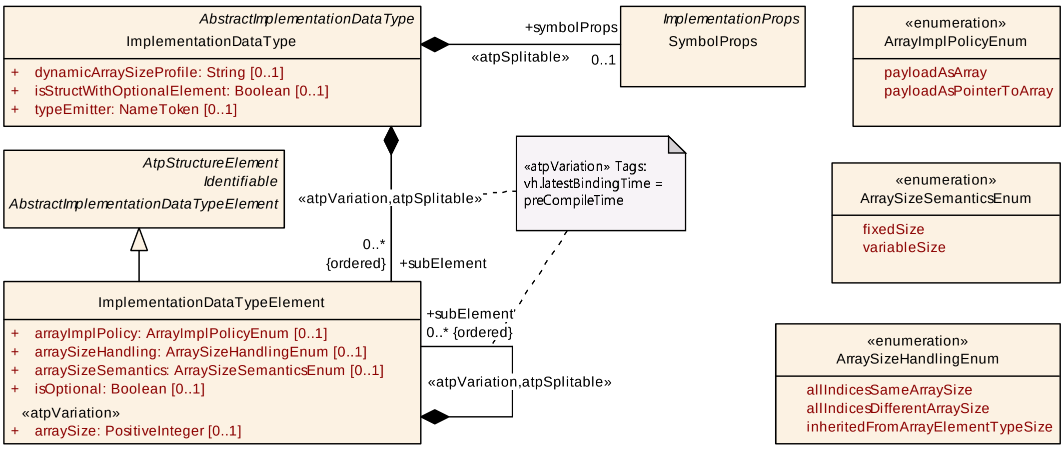


Figure 5.13: ImplementationDataType overview

[TPS_SWCT_01252] ImplementationDataType can express concepts not available on application level ⌈As a consequence of the specific focus, it is possible to express concepts with an ImplementationDataType that are not supported on the application level, i.e. by ApplicationDataType:

- ImplementationDataType supports the definition of pointers
- It is possible to define "alias" names just as in a typedef
- It is possible to define nested ImplementationDataTypes but in contrast to the concept implemented for ApplicationDataType these implement a direct aggregation of sub-elements rather than applying the type-prototype pattern.

⌋*(RS_SWCT_03217)*

The general structure of ImplementationDataType is sketched in Figure 5.13. If a specific ImplementationDataType is supposed to define a composite data type, the ImplementationDataType aggregates ImplementationDataTypeElements.

Class	*AbstractImplementationDataType* (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	This meta-class represents an abstract base class for different flavors of ImplementationDataType.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	ImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.15: AbstractImplementationDataType

Class	ImplementationDataType			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:**atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=subElement.shortName, sub Element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbolProps.shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table 5.16: ImplementationDataType

[TPS_SWCT_01248] Nested definition of `ImplementationDataType` ⌈If an `ImplementationDataTypeElement` also represents a composite data type, it can aggregate `ImplementationDataTypeElement`s in the role of `subElement`. Again, the type-prototype pattern does not apply in this case.⌋*(RS_SWCT_03217)*

[TPS_SWCT_01253] Rules apply for the usage of the attribute ImplementationDataType.typeEmitter ⌈The following set of values is supported for the usage of the attribute ImplementationDataType.typeEmitter:

- attribute typeEmitter is NOT defined.
- attribute typeEmitter is set to "RTE".
- attribute typeEmitter is set to the name of a header file.
- attribute typeEmitter is set to anything else.

⌋*(RS_SWCT_03217)*

The consequence of setting the value of typeEmitter is explained in [2].

The usage of ImplementationDataTypes within an AnyInstanceRef is described in detail in [11].

Class	***AbstractImplementationDataTypeElement*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	This meta-class represents the ability to act as an abstract base class for specific derived meta-classes that support the modeling of ImplementationDataTypes for a particular language binding.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ImplementationDataTypeElement			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.17: AbstractImplementationDataTypeElement

Class	**ImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ImplementationDataType.subElement, ImplementationDataTypeElement.subElement			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arrayImplPolicy	ArrayImplPolicyEnum	0..1	attr	This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing ImplementationDataType constitutes an array.

▽

△

Class	ImplementationDataTypeElement			
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this Implementation DataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ImplementationDataTypeElement as optional. This means that, at runtime, the ImplementationDataType Element may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the CppImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=subElement.shortName, sub Element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps

Table 5.18: ImplementationDataTypeElement

5.2.5.2 Modeling of Type Reference using Implementation Data Type

[TPS_SWCT_01441] Nature of a `TYPE_REFERENCE` ⌈A type reference (formally represented by an `ImplementationDataType` of `category` `TYPE_REFERENCE`) implements a redirection to common `ImplementationDataType`s.⌋*()*

[TPS_SWCT_01442] `ImplementationDataType` of `category` `TYPE_REFERENCE` does not define own properties ⌈As long as an `ImplementationDataType` of `category` `TYPE_REFERENCE` does not define own properties, the properties of the refined `ImplementationDataType` apply.⌋*(RS_SWCT_03217)*

[TPS_SWCT_01443] `ImplementationDataType` of `category` `TYPE_REFERENCE` overwrites properties of refined `ImplementationDataType` ⌈If an implementation data types of `category` `TYPE_REFERENCE` defines own properties (e.g. CompuMethod), these properties overwrite the properties of the refined `ImplementationDataType`.⌋*(RS_SWCT_03217)*

As explained by [constr_1050], Compatibility checks of ImplementationDataType require a prior resolution of possible type references, i.e. the compatibility shall be checked on the resolved ImplementationDataType.

[constr_1383] Existence of CompuMethod and DataConstr for ImplementationDataTypes of category TYPE_REFERENCE ⌈The existence of ImplementationDataType.swDataDefProps.compuMethod and ImplementationDataType.swDataDefProps.dataConstr for ImplementationDataTypes of category TYPE_REFERENCE is only allowed, if the respective ImplementationDataType, after all type references are resolved, ends up in an ImplementationDataType of category VALUE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note that, as a consequence of the existence of [constr_1383], it is possible that the elements of a composite ImplementationDataType define individual CompuMethods. However, the definition of **one** CompuMethod that applies to the **entire** composite ImplementationDataType is not supported.

5.2.5.3 Modeling of Structure using Implementation Data Type

[constr_1106] Structure shall have at least one element ⌈An ImplementationDataType or ImplementationDataTypeElement of category STRUCTURE shall own at least one ImplementationDataTypeElement **at the time when the contract phase generation is executed**.⌋*()*

5.2.5.3.1 Modeling of Optional Element Structure with ImplementationDataType

The definition of an ImplementationDataType that represents an Optional Element Structure shall not only rely on the existence of optional elements.

Also, the definition of the enclosing ImplementationDataType shall clearly signal the intention by means of the dedicated attribute ImplementationDataType.isStructWithOptionalElement.

[TPS_SWCT_01772] Semantics of attribute ImplementationDataType.isStructWithOptionalElement ⌈If attribute ImplementationDataType.isStructWithOptionalElement is set to true then the ImplementationDataType advertises the intention to represent an Optional Element Structure such that the fulfillment of structural requirements for the existence of optional elements can be formally checked.

Again, this attribute represents a formal specification that optionality is intended as opposed to an ImplementationDataType that fulfills the structural requirements out of different motivations.⌋*(RS_SWCT_03217, RS_SWCT_03320)*

[TPS_SWCT_01773] Definition of `Optional Element Structure` on the level of `ImplementationDataType` ⌈The modeling approach for the definition of an `Optional Element Structure` on the level of `ImplementationDataType` is to set the attribute `ImplementationDataTypeElement.isOptional` to the value `true`.

If the attribute is not set or set to the value `false`, then the respective `ImplementationDataTypeElement` shall be considered mandatory.⌋*(RS_SWCT_03217, RS_-SWCT_03320)*

[constr_1637] Existence of `ImplementationDataTypeElement.isOptional` vs. `ImplementationDataType.isStructWithOptionalElement` ⌈If one `ImplementationDataType.subElement` sets attribute `isOptional` to the value `true` then the enclosing `ImplementationDataType` shall also set attribute `isStructWithOptionalElement` to `true`.

This rule shall be imposed **`at the time when the contract phase generation is executed`**.⌋*()*

In order to be able to generate a proper RTE API for the access to optional elements of data types in general it is necessary to impose structural requirements on the definition of `ImplementationDataType`.

In particular, it is necessary at runtime to store the information about the availability of a specific `ImplementationDataTypeElement` where attribute `isOptional` has been set to the value `true` in the context of an `ImplementationDataType` of `category STRUCTURE`.

An `ImplementationDataType` that represents an `Optional Element Structure` shall contain a special element which represents an *availability bitfield*.

This bitfield is implemented as an array of uint8 and shall hold one bit for each optional element contained in the structured data type.

In particular, the applicable structural requirements for an `ImplementationDataType` that represents an `Optional Element Structure` are described in the following specification items.

[TPS_SWCT_01774] Modeling of `ImplementationDataType` with optional elements ⌈

The following approach shall be taken to model an `ImplementationDataType` that represents an `Optional Element Structure`:

- The first `ImplementationDataTypeElement` of `ImplementationDataType` where attribute `isStructWithOptionalElement` is set to `true` shall have the `shortName` `availabilityBitfield`. [constr_1638] applies.
- This `ImplementationDataTypeElement` shall be of `category ARRAY`
- The `ImplementationDataTypeElement` shall set attribute `arraySizeSemantics` to the value `fixedSize`.

- The ImplementationDataTypeElement shall aggregate a further ImplementationDataTypeElement in the role subElement for which the following requirements apply:
  - The ImplementationDataTypeElement shall be of category TYPE_REFERENCE that eventually refers to an ImplementationDataType that - one way or the other - implements an array of unsigned bytes, e.g. take the Platform Data Type named uint8 as the element type[9].
  - The ImplementationDataTypeElement shall set the value of attribute arraySize to max(1,ceil(numberOfOptionalElements / 8)).

⌋*(RS_SWCT_03320)*

[constr_1638] First ImplementationDataTypeElement of ImplementationDataType that represents an Optional Element Structure ⌈The first ImplementationDataTypeElement of ImplementationDataType that represents an Optional Element Structure, i.e. the availabilityBitfield according to [TPS_SWCT_01774], shall not set attribute isOptional to true **at the time when the contract phase generation is executed**.⌋*()*

A further structural requirement applies.

[constr_1639] ImplementationDataTypeElement with attribute isOptional set to true ⌈ImplementationDataTypeElement with attribute isOptional set to true shall **not** be of category STRUCTURE **at the time when the contract phase generation is executed**.⌋*()*

Instead, nested structures shall be created by modeling ImplementationDataTypeElements of category TYPE_REFERENCE that in turn refer to ImplementationDataTypes of category STRUCTURE.

Rationale: the existence of [constr_1639] simplifies the concept of the availability bitfield.

The bitfield shall **only** contain information of the availability of the direct child elements and **not** of elements of sub-structures.

By using the category TYPE_REFERENCE it is assured that a separate ImplementationDataType of category STRUCTURE is generated for the sub-structure.

Since the AUTOSAR RTE provides the APIs to access the availability information on the basis of an ImplementationDataType of category STRUCTURE, the usage of anonymous structures with optional elements is not possible.

[9] this relation could be expressed in a more formal way. But it would be a very expansive formal way in an already complicated specification item. It is assumed that it is sufficient to convey the general idea.

5.2.5.4 Modeling of Union using Implementation Data Type

[constr_1107] Union shall have at least one element ⌈An ImplementationDataType or ImplementationDataTypeElement of category UNION shall own at least one ImplementationDataTypeElement **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01759] Use cases for unions ⌈There are different use cases for the definition of a union data type:

1. The DataPrototypes derived from the union data type shall be transported over a communication network. For this purpose, it is necessary to apply a special modeling in the form of a wrapped union data type, as explained by [TPS_SWCT_01700].
2. The DataPrototypes created from the union data type are used internally within the same ECU, e.g. as a PerInstanceMemory, romBlock, or ramBlock. In this case the modeling of the union data type does not depend on specific constraints.

⌋*(RS_SWCT_03217)*

In summary, there are cases where unions can be used in PortInterfaces, but these are restricted to the fulfillment of certain conditions that are explained in [constr_1607].

[constr_1607] Only Wrapped Union Data Types in PortInterface ⌈Within the scope of a PortInterface the usage of a Union data type is only supported

- for Wrapped Union Data Types.
- for a PortInterface that is used to type a PortPrototype that does not appear as a context in an instanceRef owned by a DataMapping. See also [constr_1441].

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

5.2.5.5 Modeling of Array using Implementation Data Type

5.2.5.5.1 Overview

[TPS_SWCT_01254] ImplementationDataType with array semantics ⌈Of course, it is also possible to define an ImplementationDataType that provides array semantics.⌋*(RS_SWCT_03217)*

[TPS_SWCT_01006] ImplementationDataType.subElement.arraySize shall be used to define the size of the array ⌈The primitive attribute ImplementationDataType.subElement.arraySize shall be used to define the size of the array.⌋ *(RS_SWCT_03217)*

[TPS_SWCT_01007] Semantics of array index ⌈For an ImplementationDataType that implements an array data type, the semantics of the array index is such that

- it shall start with the value 0
- it shall run to the value of arraySize -1

⌋*(RS_SWCT_03217)*

[constr_1105] Value of arraySize ⌈The value of the attribute arraySize of an ImplementationDataTypeElement owned by an ImplementationDataType or ImplementationDataTypeElement of category ARRAY shall be greater than 0 unless attribute ImplementationDataTypeElement.arraySizeHandling exists and is set to the value inheritedFromArrayElementTypeSize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01478] Array size is defined as an attribute of the ImplementationDataTypeElement ⌈Please note that the array size is **not** defined as an attribute of the ImplementationDataType which stands for the whole array.

It is actually defined as an attribute of the ImplementationDataTypeElement which is describing the array element (note that the same pattern is used in ApplicationArrayDataType).⌋*(RS_SWCT_03217)*

Consequently, if a "struct" element represents an array this specific struct-element is given by an ImplementationDataTypeElement of category ARRAY which in turn aggregates another ImplementationDataTypeElement of e.g. category VALUE representing the array element and containing the size.

[TPS_SWCT_01255] Indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time ⌈It is also possible to indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time.⌋*(RS_SWCT_03217)*

[constr_1783] Existence of attribute ImplementationDataTypeElement.arrayImplPolicy ⌈Attribute ImplementationDataTypeElement.arrayImplPolicy shall only exist **at the time when the contract phase generation is executed** if the enclosing ImplementationDataType or ImplementationDataTypeElement is of category ARRAY.⌋*()*

The usage of attribute ImplementationDataTypeElement.arrayImplPolicy does not have an impact on model semantics, i.e. it does not impose further conditions on the modeling of arrays.

The attribute merely influences the implementation of the respective array in the programming language binding in the form of generated C code.

Enumeration	**ArrayImplPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	This meta-class provides values to configure the implementation of the payload part of an array.
Aggregated by	ImplementationDataTypeElement.arrayImplPolicy
Literal	**Description**
payloadAsArray	This configuration demands the implementation of the payload as an array. **Tags:**atp.EnumerationLiteralIndex=0
payloadAsPointerToArray	This configuration demands the implementation of the payload as a pointer to an array. **Tags:**atp.EnumerationLiteralIndex=1

Table 5.19: ArrayImplPolicyEnum

5.2.5.5.2 Modeling of Variable-Size Array using Implementation Data Type

In the same way as for ApplicationDataTypes, it is also possible to specify a Size Indicator of a variable size array which holds the number of valid elements of the array in the ImplementationDataType.

Please find more information about this topic in section 5.2.4.2.1.1.

[TPS_SWCT_01622] Modeling of a Variable-Size Array Data Type only with ImplementationDataType ⌈The modeling of a Variable-Size Array Data Type does not require the existence of an ApplicationCompositeDataType and a DataTypeMap. A Variable-Size Array Data Type can be created by just setting up an ImplementationDataType.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01610] Modeling of a Variable-Size Array Data Type with Size Indicator enabled ⌈An ImplementationDataType with category STRUCTURE where the attribute ImplementationDataType.dynamicArraySizeProfile exists represents a Variable-Size Array Data Type with Size Indicator enabled.

For the sake of a proper definition of terminology, this ImplementationDataType shall be called the VSA ImplementationDataType.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01650] Structure of the VSA ImplementationDataType ⌈The VSA ImplementationDataType shall consist of

- an ImplementationDataTypeElement representing the Size Indicator and
- an ImplementationDataTypeElement representing the Payload of the Variable-Size Array Data Type.

For the sake of a proper definition of terminology, these ImplementationDataTypeElements shall be called the VSA Size Indicator ImplementationDataTypeElement and the VSA Payload ImplementationDataTypeElement respectively.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01612] arraySizeHandling specifies how the size is determined ⌈arraySizeHandling specifies how the size is determined in case of multi-dimensional variable size array.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

The statement made by [TPS_SWCT_01612] allows the specification of coherency between the sizes of the nested variable size arrays in case of multiple dimensions.

[TPS_SWCT_01613] Internal structure of mapped ImplementationDataType ⌈The attribute dynamicArraySizeProfile specifies which internal structure the ImplementationDataType shall follow.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01614] Profiles for internal structure of mapped ImplementationDataType ⌈For the structure of the ImplementationDataType the following profiles are defined for dynamicArraySizeProfile: VSA_LINEAR, VSA_SQUARE, VSA_RECTANGULAR and VSA_FULLY_FLEXIBLE.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01615] Custom profiles for internal structure of mapped ImplementationDataType ⌈Custom profiles can be added to dynamicArraySizeProfile. They shall have a company-specific prefix.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

For reasons of readability and comprehensibility the following constraints focus on the payload of the Variable-Size Array Data Type only. For the Size Indicator additional individual constraints do apply.

[constr_1318] Profile VSA_LINEAR for ImplementationDataType ⌈If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to VSA_LINEAR, the ImplementationDataType shall aggregate a VSA Payload ImplementationDataTypeElement that fulfills all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The VSA Payload ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement that shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.

- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1319] Profile VSA_SQUARE for ImplementationDataType ⌈If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to VSA_SQUARE, the ImplementationDataType shall aggregate a VSA Payload ImplementationDataTypeElement that fulfills all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The VSA Payload ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement (representing the first dimension) that shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.

All **intermediate** ImplementationDataTypeElements in the aggregation chain that do not terminate the chain shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.

- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.

The **terminating** ImplementationDataTypeElement in the aggregation chain shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1320] Profile VSA_RECTANGULAR for ImplementationDataType ⌈If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to VSA_RECTANGULAR, the ImplementationDataType shall aggregate a VSA Payload ImplementationDataTypeElement that fulfills all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The VSA Payload ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement (representing the first dimension) that shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

All **intermediate** ImplementationDataTypeElements in the aggregation chain that do not terminate the chain shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

The **terminating** ImplementationDataTypeElement in the aggregation chain shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1321] Profile VSA_FULLY_FLEXIBLE for ImplementationDataType ⌈If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to the value VSA_FULLY_FLEXIBLE, the ImplementationDataType shall aggregate a VSA Payload ImplementationDataTypeElement that fulfills all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The VSA Payload ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement (representing the first dimension) that shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.category shall be set to STRUCTURE
- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.

The ImplementationDataTypeElement shall aggregate another ImplementationDataTypeElement that fulfills the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The **aggregation chain is continued** by a (possible empty) sequence of a pair of ImplementationDataTypeElements with the following characteristics:

- The first ImplementationDataTypeElement in the pair shall fulfill all the following conditions:
  - The attribute ImplementationDataTypeElement.category shall be set to STRUCTURE.
  - The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
  - The attribute ImplementationDataTypeElement.arraySize shall be defined.
  - The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.
- The second ImplementationDataTypeElement in the pair shall fulfill all the following conditions:
  - The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
  - The attribute ImplementationDataTypeElement.category shall be set to the value ARRAY.

  - The attribute ImplementationDataTypeElement.arraySize shall not be defined.
  - The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The **terminating** ImplementationDataTypeElement in the aggregation chain shall fulfill all the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1396] Restriction for the value of attribute category for non-terminating ImplementationDataTypeElements taken to model a Variable-Size Array Data Type ⌈The value of attribute category for non-terminating ImplementationDataTypeElements taken to model a Variable-Size Array Data Type shall **not** be set to TYPE_REFERENCE **at the time when the contract phase generation is executed**.⌋*()*

[constr_1322] Size Indicator for undefined dynamicArraySizeProfile ⌈If the ImplementationDataType.dynamicArraySizeProfile does not exist but the ImplementationDataType is mapped to an ApplicationArrayDataType where the attribute ApplicationArrayDataType.dynamicArraySizeProfile exists, then the ImplementationDataType shall have the category STRUCTURE, representing a Variable-Size Array Data Type with Size Indicator enabled **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01617] Structure of an ImplementationDataType that represents a variable-sized array data type ⌈The ImplementationDataType that represents a Variable-Size Array Data Type shall have the category STRUCTURE that has two subElements.

The role of the subElements with the definition of a Variable-Size Array Data Type is defined by [TPS_SWCT_01618], [TPS_SWCT_01619], [TPS_SWCT_01620], and [TPS_SWCT_01621].⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01618] Size Indicator for dynamicArraySizeProfile set to VSA_LINEAR, VSA_SQUARE, or VSA_FULLY_FLEXIBLE ⌈If an ImplementationDataType is mapped to an ApplicationArrayDataType which has the attribute dynamicArraySizeProfile set to the value VSA_LINEAR, VSA_SQUARE or VSA_FULLY_FLEXIBLE, the **first** ImplementationDataType.subElement shall be an integer large enough to hold the maximum number of valid elements of the variable size array (according to maxNumberOfElements).

This is the Size Indicator which holds the current number of valid elements of the variable size array.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01647] Size Indicator for dynamicArraySizeProfile set to VSA_LINEAR, VSA_SQUARE, or VSA_FULLY_FLEXIBLE if only Implementation-DataType is present ⌈For each ImplementationDataType which has the attribute dynamicArraySizeProfile set to the value VSA_LINEAR, VSA_SQUARE, or VSA_-FULLY_FLEXIBLE, the first ImplementationDataType.subElement shall be an integer large enough to hold the maximum number of valid elements of the variable size array (according to arraySize).

This is the Size Indicator which holds the current number of valid elements of the Variable-Size Array Data Type.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01619] Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR ⌈If an ImplementationDataType is mapped to an ApplicationArrayDataType where the attribute ApplicationArrayDataType.dynamicArraySizeProfile exists and is set to the value VSA_RECTANGULAR, the **first** ImplementationDataType.subElement shall be a ImplementationDataTypeElement with the category set to ARRAY and the attribute arraySize set to a value equal to the number of the according dimension of the corresponding ApplicationDataType.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01648] Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR if only ImplementationDataType is present ⌈For each ImplementationDataType where the attribute ImplementationDataType.dynamicArraySizeProfile exists and is set to the value VSA_RECTANGULAR, the **first** ImplementationDataType.subElement shall be a ImplementationDataTypeElement with the category set to ARRAY and the attribute arraySize set to a value equal to the size of the according dimension of the rectangular array.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01620] Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR ⌈The elements of this Size Indicator array shall consist of integers large enough to hold the maximum number of valid elements (according to maxNumberOfElements).⌋*(RS_SWCT_03217, RS_SWCT_03181)*

This array holds the Size Indicators of all dimensions.

[TPS_SWCT_01621] Payload for dynamicArraySizeProfile ⌈If an ImplementationDataType is mapped to an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists, the **second** ImplementationDataType.subElement shall be an array which can hold the data of the variable size array with all dimensions defined for the ApplicationDataType.

The category shall be set to ARRAY and arraySize shall be set to maxNumberOfElements of the corresponding ApplicationArrayDataType.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

[TPS_SWCT_01649] Payload for dynamicArraySizeProfile if only ImplementationDataType is present ⌈Each ImplementationDataType where the attribute dynamicArraySizeProfile exists shall aggregate a second ImplementationDataType.subElement with the category set to ARRAY.⌋*(RS_SWCT_03217, RS_SWCT_03181)*

For examples, see Appendix E.1.

An ImplementationDataType is also allowed to have SwDataDefProps (this feature is inherited from AutosarDataType), i.e. it can define various specific structural and semantical attributes. [constr_1015] shows which SwDataDefProps will be typically used here.

[TPS_SWCT_01257] ImplementationDataType or the aggregated ImplementationDataTypeElements do not form closed sets ⌈An ImplementationDataType or the aggregated ImplementationDataTypeElements do not form closed sets but refer to further type definitions in one of four distinctive ways, depending on whether the type is implemented via a base type, a data or function pointer, or a reference to another implementation data type:

1. Reference to an underlying SwBaseType corresponds to category VALUE.
2. Reference to BswModuleEntry in SwPointerTargetProps corresponds to category FUNCTION_REFERENCE.
3. SwDataDefProps in SwPointerTargetProps corresponds to category DATA_REFERENCE.
4. Reference to another ImplementationDataType corresponds to category TYPE_REFERENCE.

⌋*(RS_SWCT_03217)*

At the end, all the “leaves” of the complete tree formed by these references shall end up in SwBaseTypes.

Figures 5.14, 5.15, and Figure 5.16 illustrate more examples about Typedefs and references.

5.2.5.6 Modeling of Pointer using Implementation Data Type

[TPS_SWCT_01258] Definition of a pointer to data ⌈The definition of a data pointer requires a special meta-class SwPointerTargetProps which aggregates another SwDataDefProps. This mechanism allows to describe the category and properties of the pointer object itself as well as the category and properties of its target data type.⌋*(RS_SWCT_03217)*

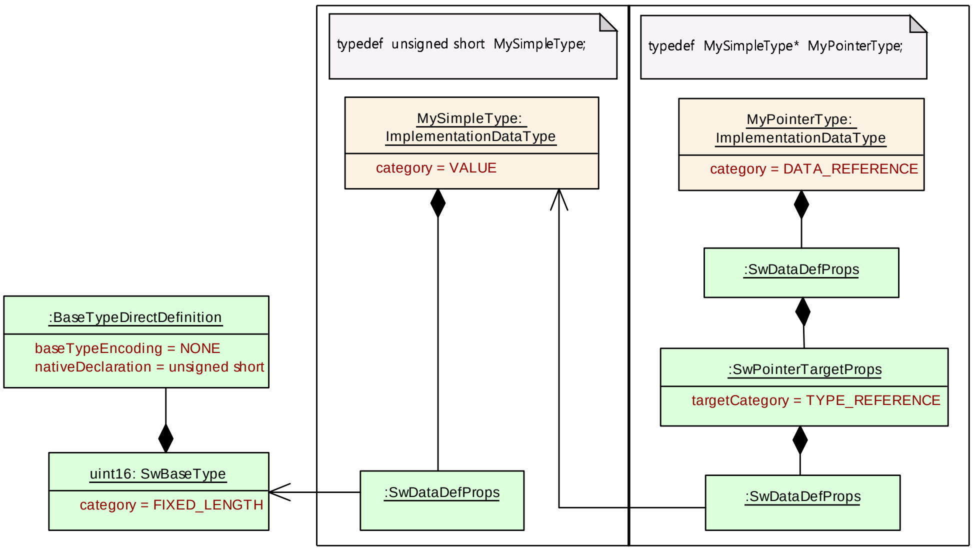


Figure 5.14: Example (1) for TypeDefs

[constr_1177] Allowed targetCategory for SwPointerTargetProps ⌈If the value of attribute targetCategory exists, then it shall be set to one of the following values:

- TYPE_REFERENCE
- FUNCTION_REFERENCE
- VALUE (only applicable if the SwPointerTargetProps.swDataDefProps refers to a SwBaseType where attribute nativeDeclaration is set to the value "void")

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

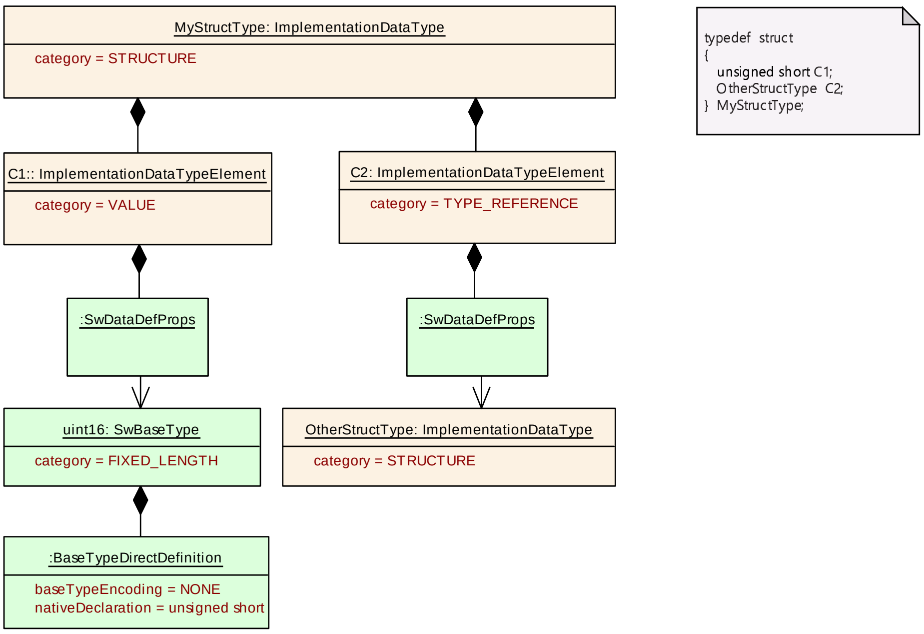


Figure 5.15: Example (2) for TypeDefs

As far as the AUTOSAR meta-model is concerned, a pointer to a pointer **could** in principle be implemented in two ways:

1. by defining an ImplementationDataType of category DATA_REFERENCE that aggregates SwDataDefProps in the role swDataDefProps that in turn aggregate SwPointerTargetProps in the role swPointerTargetProps with attribute targetCategory set to TYPE_REFERENCE that aggregates SwDataDefProps in the role swDataDefProps that references an ImplementationDataType of category DATA_REFERENCE.

2. by defining an ImplementationDataType of category DATA_REFERENCE that aggregates SwDataDefProps in the role swDataDefProps that in turn aggregate SwPointerTargetProps in the role swPointerTargetProps with attribute targetCategory set to DATA_REFERENCE (which is not allowed according to [constr_1177]) that in turn aggregates SwDataDefProps in the role swDataDefProps that aggregates SwPointerTargetProps in the role swPointerTargetProps that references an ImplementationDataType of category e.g. VALUE.

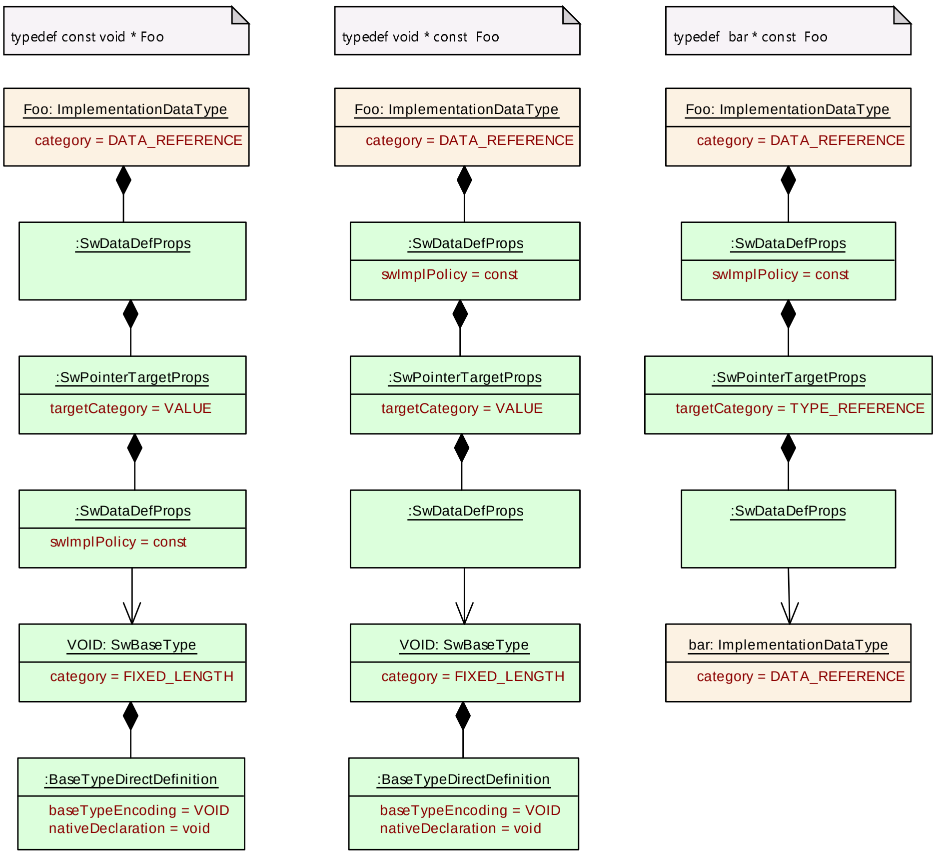


Figure 5.16: Example (3) for TypeDefs

[constr_1254] Definition of a pointer to a pointer ⌈AUTOSAR does **not** support the definition of a pointer to a pointer by defining an ImplementationDataType of category DATA_REFERENCE that aggregates SwDataDefProps in the role swDataDefProps that in turn aggregate SwPointerTargetProps in the role swPointerTargetProps with attribute targetCategory set to DATA_REFERENCE that in turn aggregates SwDataDefProps in the role swDataDefProps that aggregates SwPointerTargetProps in the role swPointerTargetProps that references an ImplementationDataType of category e.g. VALUE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

For clarification, The AUTOSAR RTE does not support a definition of a pointer to a pointer by way of option 2 anyway. For all intents and purposes, [constr_1254] merely reflects this restriction on the level of AUTOSAR models.

Option 1 (which is also featured in Figure 5.16) is the only viable way that is positively supported by the AUTOSAR RTE [2].

[TPS_SWCT_01259] Definition of a pointer to a function ⌈An ImplementationDataType or one of its sub-elements can also describe a function pointer. This completes its ability to declare all kinds of local data and of possible arguments used in library calls.

A function pointer is defined by the category FUNCTION_REFERENCE and the association SwPointerTargetProps.functionPointerSignature that refers to a BswModuleEntry. The latter essentially describes the signature of a function as explained in [6].⌋*(RS_SWCT_03217)*

Class	**SwPointerTargetProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This element defines, that the data object (which is specified by the aggregating element) contains a reference to another data object or to a function in the CPU code. This corresponds to a pointer in the C-language. The attributes of this element describe the category and the detailed properties of the target which is either a data description or a function signature.			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swPointerTargetProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
functionPointer Signature	BswModuleEntry	0..1	ref	The referenced BswModuleEntry serves as the signature of a function pointer definition. Primary use case: function pointer passed as argument to other function. **Tags:**xml.sequenceOffset=40
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of the target data type. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps xml.sequenceOffset=30

▽

△

Class	**SwPointerTargetProps**			
targetCategory	Identifier	0..1	attr	This specifies the category of the target: • In case of a data pointer, it shall specify the category of the referenced data. • In case of a function pointer, it could be used to denote the category of the referenced Bsw ModuleEntry. **Tags:**xml.sequenceOffset=5

Table 5.20: SwPointerTargetProps

Please note that an ImplementationDataType manifests itself in the source code of an RTE into which a DataPrototype typed by the ImplementationDataType is deployed. This implies potential naming conflicts if ImplementationDataTypes that have identical shortNames are deployed into a specific RTE.

[TPS_SWCT_01194] Symbolic name of an ImplementationDataType ⌈To mitigate this potential hazard it is possible to provide the ImplementationDataType along with an accompanying symbolic name that can be used for resolving the name clash.

The symbolic name is provided by means of the attribute symbol of the meta-class SymbolProps owned by ImplementationDataType in the role symbolProps.⌋*()*

For more information about symbolProps, please refer to Figure 5.13.

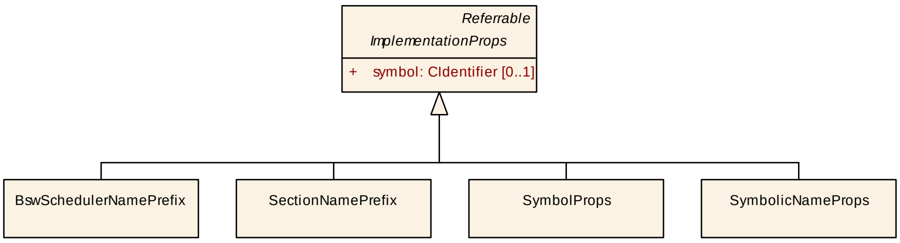


Figure 5.17: ImplementationProps and its subclasses

Class	***ImplementationProps*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
symbol	CIdentifier	0..1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table 5.21: ImplementationProps

[constr_1909] Existence of attribute ImplementationProps.symbol ⌈For each ImplementationProps, the attribute symbol shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Aggregated by	Allocator.namespace, ApApplicationErrorDomain.namespace, *AtomicSwComponentType*.symbolProps, *CppImplementationDataType*.namespace, ImplementationDataType.symbolProps, *PortInterface*.namespace, SecurityEventDefinition.eventSymbolName			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 5.22: SymbolProps

5.2.6 Base Type

5.2.6.1 Big Picture

From the terminology point of view, the definition of a base type as a “type” is slightly misleading in that it certainly has a type-nature, but does not necessarily represent a full-scale data type.

In the context of the software domain, this base type is not a self-contained level of definition of a data type. Instead, some of the attributes of a base type (represented by meta-class SwBaseType) provide “low-level”[10] aspects to the definition of an ImplementationDataType.

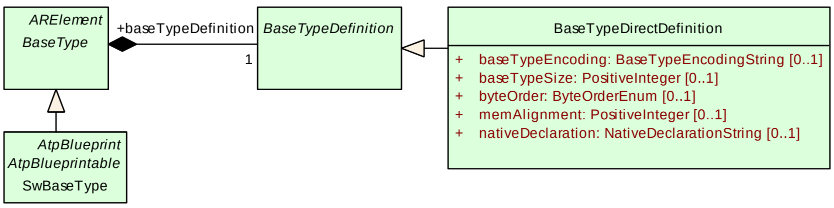


Figure 5.18: BaseType

However, there are other corners of the AUTOSAR meta-model beyond the definition of data types and similar “low-level” information is required elsewhere as well, such that SwBaseType is consequently also used for the modeling of these aspects, for example:

- Definition of axes for maps and curves
- Definition of record layout

[10] As in: close to the bare metal

- Definition of signal properties on communication buses (see TPS System Template [10])
- Definition of properties of diagnostic telegrams (see TPS Diagnostic Extract Template [23])
- Definition of log and trace messages (see TPS Log and Trace Extract [24])
- Definition of properties relevant for measurement and calibration (see TPS Basic SoftwareModuleDescriptionTemplate [6])

Attribute of meta-class BaseTypeDirectDefinition	ImplementationDataType	SwTextProps	ComSpec	SwCalprmAxis	SwRecordLayoutV	SystemSignal	ISignal	DiagnosticDataElement	DltArgument	McDataInstance
baseTypeEncoding	X	X	N/A	X	X	X	X	X	X	X
baseTypeSize	X	N/A	X	X	X	N/A	X	X	X	X
byteOrder	X	N/A	X	X	X	N/A	N/A	X	X	X
memAlignment	X	N/A	N/A	X	X	N/A	N/A	N/A	N/A	X
nativeDeclaration	X	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 5.23: Usage of attributes of BaseTypeDirectDefinition depending on the area of applicability

The diversity of the areas of applicability (as hinted by the list in the previous paragraph) of SwBaseType is also reflected in the set of attributes that are relevant for each of the different areas of applicability, as indicated by Table 5.23.

Please note that in the context of using SwBaseType to define an ImplementationDataType, the attributes

- byteOrder
- memAlignment

are used only if the ImplementationDataType represents a Platform Type.

In this document, the description of the utilization of SwBaseType is focused on the areas of applicability that are in the scope of this document, i.e.

- Definition of data types
- Definition of base points for axes for maps and curves
- Definition of the values of a particular record layout group
- Definition of network representation in the scope of a ComSpec

[constr_1910] Existence of attribute BaseType.baseTypeDefinition ⌈For each BaseType (which will be utilized in the form of SwBaseType), the aggregation in the role baseTypeDefinition shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01260] Applicability of SwBaseType for the definition of data types ⌈BaseType is used to contribute low-level aspects to the definition of an ImplementationDataType. Consequently, the information carried by SwBaseType is used to generate the corresponding C-code typedefs in case the attribute BaseTypeDirectDefinition.nativeDeclaration exists at the "leaves" of data type definitions.⌋*()*

Class	**SwBaseType**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:**atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *BaseType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.24: SwBaseType

Class	***BaseTypeDefinition*** (abstract)			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents the ability to define a basetype.			
Base	*ARObject*			
Subclasses	BaseTypeDirectDefinition			
Aggregated by	*BaseType*.baseTypeDefinition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.25: BaseTypeDefinition

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *BaseTypeDefinition*			
Aggregated by	*BaseType*.baseTypeDefinition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseType Encoding	BaseTypeEncoding String	0..1	attr	This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **Tags:**xml.sequenceOffset=90
baseTypeSize	PositiveInteger	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:**xml.sequenceOffset=70

▽

△

Class	BaseTypeDirectDefinition			
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:**xml.sequenceOffset=110
memAlignment	PositiveInteger	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified". **Tags:**xml.sequenceOffset=100
native Declaration	NativeDeclarationString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example BaseType with shortName: "MyUnsignedInt" native Declaration: "unsigned short" Results in typedef unsigned short MyUnsignedInt; If the attribute is not defined the referring Implementation DataTypes will not be generated as a typedef by RTE. If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseType Size. This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems. **Tags:**xml.sequenceOffset=120

Table 5.26: BaseTypeDirectDefinition

Class	***BaseType*** (abstract)			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This abstract meta-class represents the ability to specify a platform dependent base type.			
Base	*ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable*			
Subclasses	SwBaseType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseType Definition	BaseTypeDefinition	1	aggr	This is the actual definition of the base type. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 5.27: BaseType

The aggregation `BaseType.baseTypeDefinition` affects two abstract meta-classes and is therefore never used directly. However, in order to not break the semantics of the derived meta-classes `SwBaseType` and `BaseTypeDirectDefinition` a constraint is still required.

5.2.6.2 Category

[constr_1011] category of SwBaseType ⌈For the attribute SwBaseType.category only the values FIXED_LENGTH and VOID are supported **at the time when the contract phase generation is executed**.⌋*()*

[constr_1422] Value of category is VOID ⌈If the value of the attribute SwBaseType.category is set to VOID then the attribute baseTypeSize and baseTypeEncoding shall not exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1012] Value of category is FIXED_LENGTH ⌈If the value of the attribute SwBaseType.category is set to FIXED_LENGTH then the attribute baseTypeSize shall be filled with content **at the time when the contract phase generation is executed**.⌋*()*

5.2.6.3 Native Declaration

[constr_1010] If nativeDeclaration does not exist ⌈If nativeDeclaration does not exist in the SwBaseType, it is required that the shortName (e.g. "uint8") of the corresponding ImplementationDataType is equal to a name of one of the Platform or Standard Types predefined in AUTOSAR code.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The consequence of [constr_1010] is that if the nativeDeclaration does not exist the RTE generator will **not** consider the ImplementationDataType for the generation of data type definitions.

Still, the compiler will positively be able to resolve the data type because it can fall back to the data type definitions contained in the header file for platform and standard data types that has to be included by regulation of the AUTOSAR standard.

Please note that nativeDeclaration shall yield a valid C data type symbol, whether this is done by a typedef or a by using the symbol[11] of an integral data type is principally all the same.

Of course, using the symbol of an integral data type as the value of nativeDeclaration increases the odds that the enclosing SwBaseType can be used independently of the availability of the definition of a typedef that may or may not be available in a given context.

[TPS_SWCT_01563] Applicable values for nativeDeclaration ⌈For the purpose of avoiding portability issues the value nativeDeclaration should only consist of the symbol of an integral C data type.⌋*()*

[11] Note that the symbol does not necessarily have to consist of a single token, i.e. for all intents and purposes (for example) unsigned char is also considered the symbol of an integral C data type.

For more information on this refer to [25].

5.2.6.4 Base Type Size

[TPS_SWCT_01444] Size of SwBaseType is specified in bits ⌈The value of attribute BaseTypeDirectDefinition.baseTypeSize is specified in bits.⌋*()*

The value of attribute baseTypeSize in the context of the definition of an ImplementationDataType where attribute category is set to the value ARRAY is modeled as part of the ImplementationDataTypeElement, see Listing 5.9.

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>MyImplDataType</SHORT-NAME>
  <CATEGORY>ARRAY</CATEGORY>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>MyArrayElement</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <ARRAY-SIZE>6</ARRAY-SIZE>
      <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <BASE-TYPE-REF DEST="SW-BASE-TYPE">/SwBaseTypes/MyBaseType</
                BASE-TYPE-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 5.9: Definition of an ImplementationDataType that represents an array

The referenced SwBaseType is sketched in Listing 5.10.

```
<SW-BASE-TYPE>
  <SHORT-NAME>MyBaseType</SHORT-NAME>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
```

Listing 5.10: Definition of SwBaseType of an elements of an ImplementationDataType that represents an array

Because the value of baseTypeSize is defined in the context of the SwDataDefProps referenced by the ImplementationDataTypeElement, it should be clear that this value is **not supposed to determine the size of the entire array in bits**. Instead, it is clearly dedicated to specify the **size of a primitive element of the array in bits**.

In the case of the example presented in Listings 5.9 and 5.10, the size of an element of the array is 32 bits. Thus, the entire array occupies $32 * 6 = 192$ bits.

The attribute baseTypeEncoding specifies how the values of the base type are encoded.

[TPS_SWCT_01861] Specification of baseTypeSize in the context of an ImplementationDataType where attribute category is set to ARRAY ⌈In the context of the definition of an ImplementationDataType, where attribute category is set to ARRAY, the conditions for the definition of baseTypeSize depends on the nature of the ImplementationDataTypeElement:

- If the ImplementationDataTypeElement represents a primitive data type (i.e. category is set to VALUE or to TYPE_REFERENCE where eventually an ImplementationDataType of category VALUE is referenced): The value of attribute baseTypeSize that is defined in the context of the definition of an ImplementationDataType where attribute category is set to ARRAY applies to the **size of an element of the array data type**.
- If the ImplementationDataTypeElement represents a composite data object (i.e. the ImplementationDataTypeElement that represents the element of the array has attribute category set to STRUCTURE or ARRAY), then attribute baseTypeSize shall only be used on deeper levels of the hierarchy.
- If the ImplementationDataTypeElement is typed a composite data type (i.e. the ImplementationDataTypeElement that represents the element of the array has attribute category set to TYPE_REFERENCE, where eventually an ImplementationDataType of category STRUCTURE or ARRAY is referenced), then the definition of attribute baseTypeSize is delegated to deeper levels of the referenced ImplementationDataType.

⌋*()*

Please note that the understanding specified in [TPS_SWCT_01861] is also helpful in terms of model economics. If the value were supposed to define the size of the entire array, it would typically be necessary to create a lot of additional SwBaseTypes for this purpose.

The fact that the value is meant per element means that it is possible to re-use an SwBaseType that most likely already exists (because it is very likely that the code that accesses the array also needs to deal with the element data type anyway).

5.2.6.5 Base Type Encoding

The attribute baseTypeEncoding specifies how the values of the base type are encoded.

[TPS_SWCT_01845] Supported value encodings for SwBaseType ⌈The supported values for attribute BaseTypeDirectDefinition.baseTypeEncoding are:

- 1C: One's complement

- 2C: Two's complement
- BCD-P: Packed Binary Coded Decimals
- BCD-UP: Unpacked Binary Coded Decimals
- DSP-FRACTIONAL: Digital Signal Processor
- SM: Sign Magnitude
- IEEE754: floating-point numbers
- ISO-8859-1: single-byte coded character
- ISO-8859-2: single-byte coded character
- WINDOWS-1252: single-byte coded character
- UTF-8: UCS Transformation Format 8
- UTF-16: Character encoding for Unicode *code points* based on 16 bit *code units* [15]
- UCS-2: Universal Character Set 2
- NONE: Unsigned Integer
- VOID: corresponds to a void in C. The encoding is not formally specified here.
- BOOLEAN: This represents an unsigned integer to be interpreted as boolean. The value shall be interpreted as true if the value of the unsigned integer is 1 and it shall be interpreted as false if the value of the unsigned integer is 0.

  A CompuMethod shall be referenced by the corresponding AutosarDataType that implements the common sense behind the boolean concept, i.e. define a TEXTTABLE with two CompuScales: e.g. true –> 1, false –> 0.

This rule shall be imposed **at any time in the workflow**.⌋*()*

Please note that restrictions apply to the usage of some values of attribute BaseTypeDirectDefinition.baseTypeEncoding, as documented in [constr_10383].

At least for the usage of SwBaseType in the context of software, there is an obvious relation between the definition of the baseTypeSize and the baseTypeEncoding such that there is a minimum value that needs to be defined for the definition of the baseTypeSize for certain values of baseTypeEncoding, see Table 5.28.

Please note that it is still possible to use objects that are larger than required for storing information that occupies only a part of the data object.

For example, it would be possible to put UTF-8-encoded information into a 64-bit wide data object. But in this case, only 8 bit of the 64-bit object would actually be usable. The rest is wasted[12]

baseTypeEncoding	Minimum value of baseTypeSize
1C	8
2C	8
BCD-P	8
BCD-UP	8
DSP-FRACTIONAL	8
SM	8
IEEE754	32
ISO-8859-1	8
ISO-8859-2	8
WINDOWS-1252	8
UTF-8	8
UTF-16	16
UCS-2	16
NONE	8
VOID	n/a
BOOLEAN	8

Table 5.28: Minimum value of baseTypeSize for a given baseTypeEncoding

In other words, using a 64-bit data object to host information encoded as UTF-8 does not mean that the object is filled with UTF-8 information, thereby forming some sort of an array of UTF-8-encoded information.

In AUTOSAR, an array has to be defined as an array data type, i.e. ApplicationArrayDataType or ImplementationDataType where attribute category is set to the value ARRAY.

5.2.6.6 Byte Order and Memory Alignment

[TPS_SWCT_01262] memAlignment and byteOrder are platform-specific ⌈The value of attributes BaseTypeDirectDefinition.memAlignment and BaseTypeDirectDefinition.byteOrder is platform-specific and therefore should be set only in use cases where this is really needed.

These attributes shall be considered as optional.

If a SwBaseType is platform-specific then also the ImplementationDataType and software-component descriptions build on top of it become platform-specific.⌋*()*

[12]A motivation for such a choice may be that the underlying hardware is faster or maybe only capable of word-based addressing and therefore the storing of the UTF-8 information in just a single 8-bit-wide data object might not be possible.

However, there are use cases for SwBaseType where this does not matter: especially the calibration support format which is generated in ECU-specific scope (and also contains SwBaseType, see [6]) could well be platform-specific.

Further regulations apply for the case that the value UTF-16 is used for setting the attribute BaseTypeDirectDefinition.baseTypeEncoding:

[constr_1398] Existence of attributes of BaseTypeDirectDefinition ⌈If the value of attribute BaseTypeDirectDefinition.baseTypeEncoding is set to UTF-16 then the attribute BaseTypeDirectDefinition.byteOrder shall exist.

The only allowed values of BaseTypeDirectDefinition.byteOrder in this case are mostSignificantByteFirst and mostSignificantByteLast.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

There is already predefined terminology (see [15]) existing that describes the two possible cases of byte orientation in a UTF-16-encoded string. The connection to this terminology is defined by [TPS_SWCT_01651] and [TPS_SWCT_01652].

Enumeration	**ByteOrderEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	When more than one byte is stored in the memory the order of those bytes may differ depending on the architecture of the processing unit. If the least significant byte is stored at the lowest address, this architecture is called little endian and otherwise it is called big endian. ByteOrder is very important in case of communication between different PUs or ECUs.
Aggregated by	ApSomeipTransformationProps.byteOrder, BaseTypeDirectDefinition.byteOrder, DiagnosticCommonProps.defaultEndianness, ISignalToIPduMapping.packingByteOrder, MultiplexedIPdu.selectorFieldByteOrder, PduToFrameMapping.packingByteOrder, SegmentPosition.segmentByteOrder, SOMEIPTransformationDescription.byteOrder, System.containerIPduHeaderByteOrder
Literal	*Description*
mostSignificantByteFirst	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format) **Tags:**atp.EnumerationLiteralIndex=0
mostSignificantByteLast	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format) **Tags:**atp.EnumerationLiteralIndex=1
opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details. **Tags:**atp.EnumerationLiteralIndex=2

Table 5.29: ByteOrderEnum

[TPS_SWCT_01651] UTF-16BE ⌈If the value of attribute BaseTypeDirectDefinition.baseTypeEncoding is set to UTF-16 and the attribute BaseTypeDirectDefinition.byteOrder in this case are mostSignificantByteFirst then the SwBaseType corresponds to the definition of UTF-16BE according to the Unicode standard [15].⌋*()*

[TPS_SWCT_01652] UTF-16LE ⌈If the value of attribute BaseTypeDirectDefinition.baseTypeEncoding is set to UTF-16 and the attribute BaseTypeDirectDefinition.byteOrder in this case are mostSignificantByteLast then the

SwBaseType corresponds to the definition of UTF-16LE according to the Unicode standard [15].⌋()

A further question that needs clarification is the usage of the so-called Byte Order Mark which allows (at run-time) for determining the actual byte order directly from the payload of a Unicode string.

As AUTOSAR has means to formally and comprehensively define the byte-order of any given DataPrototype that can hold a string at run time, it is **not** necessary to support a further instrument that takes care of the same purpose.

[TPS_SWCT_01653] UTF-16-encoded strings are not allowed to start with a BOM ⌈If the value of attribute BaseTypeDirectDefinition.baseTypeEncoding is set to UTF-16 then the value of a DataPrototype (which is effectively representing a string) is not allowed to start with a Byte Order Mark (BOM).⌋()

Please note that [TPS_SWCT_01653] removes a possible redundancy in the definition and execution of UTF-16-encoded strings.

The redundancy is not only regarded unnecessary but also **potentially dangerous** because it is not possible to check whether the definition is consistent with the execution at configuration time.

From the formal point of view, [TPS_SWCT_01653] does not represent an actual constraint although it is formulated as such.

However, an AUTOSAR tool would not be able to properly check the condition at configuration time and therefore this rule is published as a specification item.

5.2.7 Data Type Terminology

There are uses of data types that on the one hand need a handy term (because this kind of data type is used a lot) but on the other hand cannot easily be expressed in simple terms of meta-model elements (like ApplicationDataType).

Therefore, it is not an option to fully describe the characteristics of these kinds of data types precisely every time one of these is used. A definition of terminology is supposed to associate the mentioned kinds of data types with the term under which their use shall be paraphrased.

5.2.7.1 Primitive Type

In some cases it is necessary to constrain that applicability of data types to primitive C data types. It would be possible to describe the characteristics of eligible AutosarDataTypes at every single place in an AUTOSAR specification where this specific limitation applies.

However, this may end up in lengthy and potentially inconsistent descriptions at different places within AUTOSAR specifications. Therefore, this chapter provides a canonical definition of a primitive data type that can be referred to from other places.

[TPS_SWCT_01564] Non-recursive definition of a primitive data type ⌈An AutosarDataType is considered a primitive data type if the following conditions apply:

- it is an ApplicationPrimitiveDataType of category VALUE or BOOLEAN
- it is an ImplementationDataType of category VALUE

⌋*(RS_SWCT_03216, RS_SWCT_03217)*

[TPS_SWCT_01565] Recursive definition of a primitive data type ⌈An AutosarDataType is considered a primitive data type if the following conditions apply:

- it is an AutosarDataType according to [TPS_SWCT_01564]
- it is an AutosarDataType of category TYPE_REFERENCE that, after all type-references have been resolved, boils down an AutosarDataType according to [TPS_SWCT_01564].

⌋*(RS_SWCT_03216, RS_SWCT_03217)*

5.2.7.2 Compound Primitive Data Type

[TPS_SWCT_01179] Compound Primitive Data Type ⌈For clarification, a "compound primitive data type" is an ApplicationPrimitiveDataType of category STRING, CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK.

This implies the existence of a swRecordLayout owned by the swDataDefProps of the ApplicationPrimitiveDataType that defines the mapping to a corresponding ImplementationDataType.

The main characteristic of the "compound primitive data type" is that with respect to the application data type layer its data type is considered a primitive data type but when it comes to the implementation data type layer the type is implemented as a composite data type according to the applicable SwRecordLayout.⌋*(RS_SWCT_03216)*

Properties of a Compound Primitive Data Type with respect to the definition of an invalidValue are described in section 5.4.2.

5.2.7.3 Integral Primitive Type

The SenderReceiverToSignalMapping (see [10]) allows for the integral mapping of a piece of data to a single SystemSignal. The specification of AUTOSAR COM [22] imposes certain requirements on the characteristics of data that apply for the integral mapping.

[TPS_SWCT_01477] **`Integral Primitive Types`** ⌈Data types that qualify for being used in the context of a `SenderReceiverToSignalMapping` shall be called `Integral Primitive Type`s.⌋*(RS_SWCT_03218)*

[constr_1229] **`category` of `ImplementationDataType` boils down to `VALUE`** ⌈An `ImplementationDataType` qualifies as an `Integral Primitive Type` if and only if either

- its `category` is `VALUE` or `TYPE_REFERENCE` that eventually boils down to `VALUE` **or**
- its `category` is `ARRAY` **and** it has only one `subElement` **and** one of the following conditions applies:
  - `subElement.category` is set to `VALUE` or `TYPE_REFERENCE` that eventually boils down to `VALUE` **and** the `subElement` refers to a `SwBaseType` where `baseTypeSize` is set to the value 8 **and** the `baseTypeEncoding` is set to `NONE`.
  - `subElement.category` is set to `TYPE_REFERENCE` **and** the `swDataDefProps.implementationDataType` literally represents the `Platform Data Type` named “uint8”.
  - `subElement.category` is set to `TYPE_REFERENCE` **and** the attribute `swDataDefProps.implementationDataType.shortName` is set to “uint8” **and** `swDataDefProps.baseType.baseTypeDefinition.nativeDeclaration` does not exist.

This rule shall be imposed **`at the time when the contract phase generation is executed`**.⌋*()*

[constr_1230] **`ApplicationDataType` that qualifies for `Integral Primitive Type`** ⌈An `ApplicationDataType` qualifies as an `Integral Primitive Type` if and only if **all** the following conditions apply:

- `ApplicationDataType.category` is set to `BOOLEAN`, `VALUE`, `STRING`, or `ARRAY`
- in the applicable scope a `DataTypeMap` is available that refers to the given `ApplicationDataType`
- the found `DataTypeMap` refers to an `ImplementationDataType` that fulfills the requirements of [constr_1229]

This rule shall be imposed **`at the time when the contract phase generation is executed`**.⌋*()*

5.2.7.4 Variable-Size Array Data Type

The definition of and further explanation regarding the term Variable-Size Array Data Type can be found in chapter 2.7.

5.2.7.5 Wrapped Union Data Type

There are use cases for sending a DataPrototype that is effectively typed by a **union** data type over a communication network. In this case, however, it is necessary to not only send the DataPrototype itself but add an information about the applicable member of the union as a form of "meta-data" to the transmission.

By this means the sender can identify the applicable member of the union and the receiver can accordingly access the proper union element.

It is the nature of union data types that executable code shall **symmetrically** access the union, i.e. the member that was written needs to be read, the usage of a union as a "type converter" is heavily frowned upon (because it causes unspecified behavior from ISO-C:99 [26] point of view) and shall be discouraged by AUTOSAR.

Thus, AUTOSAR needs to take this condition into account and define a specific modeling for the handling of union data types.

[TPS_SWCT_01700] Definition of unions that can be transmitted over a communication network ⌈If it is intended to send a data object typed by a union data type over a communication bus then a specific modeling is required for this purpose.

- The union data type shall never be used as such, it shall always be enclosed in an ImplementationDataType of category STRUCTURE that aggregates exactly two ImplementationDataTypeElements:
  - The **first** ImplementationDataTypeElement shall be used to identify the applicable element of the actual union data type.

    The shortName of this element shall be set to "**memberSelector**", it shall be of category VALUE, or of category TYPE_REFERENCE that finally boils down to category VALUE.

    Furthermore, it shall refer to a SwBaseType with attribute baseTypeEncoding set to NONE and attribute baseTypeSize set to the value 8, 16, or 32.

    This ImplementationDataTypeElement shall be called the Member Selector.
  - The **second** ImplementationDataTypeElement shall be of category UNION, it represents the actual union "payload".
- The purpose of the Member Selector is to identify the element of the union data type that applies for a given access to the union.

If the value of the Member Selector is set to 1 then the first subElement of the ImplementationDataType of category UNION is applicable.

If the value of the Member Selector is set to 2 then the second subElement is applicable and so on.

- The value of the Member Selector shall range between **the value 1** and the number of subElements of the ImplementationDataTypeElement of category UNION. Once again, the index counting is 1-based!
- Obviously, the actual data type used to hold the Member Selector shall be capable of storing a value that corresponds to the number of subElements of the ImplementationDataTypeElement of category UNION.
- Constraint [constr_1441] applies.

⌋*(RS_SWCT_03217)*

[TPS_SWCT_01701] Wrapped Union Data Type ⌈Data types that fulfill the requirements of [TPS_SWCT_01700] shall be called Wrapped Union Data Types.⌋*(RS_SWCT_03217)*

[constr_1442] category TYPE_REFERENCE shall not be used for modeling the "payload" of a Wrapped Union Data Type ⌈For the modeling of the "payload" part of a Wrapped Union Data Type it shall not be possible **at any time in the workflow** to use an ImplementationDataTypeElement of category TYPE_REFERENCE that finally (i.e. after all possible indirections are resolved) boils down to category UNION.⌋*()*

The definition of the Wrapped Union Data Type represents the **canonical way** of how union data types shall be used in AUTOSAR on the application and communication level. Consequentially, the usage of the category value UNION is effectively limited to an ImplementationDataTypeElement.

[constr_1444] Limited applicability of Wrapped Union Data Type ⌈There is no support **at any time in the workflow** for the usage of Wrapped Union Data Type in PortInterfaceMappings, and Diagnostics.⌋*()*

For the time being, AUTOSAR restricts the initialization of a union data type to the first member of the union data type, see [constr_1445].

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>UnionExample</SHORT-NAME>
  <CATEGORY>STRUCTURE</CATEGORY>
  <SUB-ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>memberSelector</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
    <IMPLEMENTATION-DATA-TYPE-ELEMENT>
      <SHORT-NAME>payload</SHORT-NAME>
      <CATEGORY>UNION</CATEGORY>
      <SUB-ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
```

```
          <SHORT-NAME>primitive</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
          <SHORT-NAME>array</SHORT-NAME>
          <CATEGORY>ARRAY</CATEGORY>
          <SUB-ELEMENTS>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
              <SHORT-NAME>arraySub</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
              <ARRAY-SIZE>4</ARRAY-SIZE>
              <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
          </SUB-ELEMENTS>
        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
      </SUB-ELEMENTS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 5.11: Simplified example of a Wrapped Union Data Type

One obvious consequence of this restriction is that for any given ValueSpecification taken to initialize a Wrapped Union Data Type the value of the Member Selector is **strictly** locked to 1.

[constr_1445] Initialization of the Member Selector of a Wrapped Union Data Type ⌈The initValue for the Member Selector shall **never be set to any value other than 1**.

This rule shall be imposed **at any time in the workflow**.⌋*()*

Another aspect of the initialization of a Wrapped Union Data Type is that the “payload” part cannot be treated as a composite data type unless the first element of the “payload” part is typed by a composite data type.

In other words, it is not possible to initialize the first subElement of an ImplementationDataTypeElement of category UNION. It is only possible to assign an initial value to the “payload” part itself.

[TPS_SWCT_01702] Initialization of the “payload” of a Wrapped Union Data Type ⌈The initValue for the ImplementationDataTypeElement of category UNION shall be assigned to the ImplementationDataTypeElement of category UNION but it shall reflect the structure of the first subElement of the ImplementationDataTypeElement of category UNION.⌋*(RS_SWCT_03217)*

In other words, if the first subElement of the ImplementationDataTypeElement of category UNION is of a primitive type then a NumericalValueSpecification shall be used to initialize the ImplementationDataTypeElement of category UNION.

If the subElement is typed by a composite data type then a CompositeValueSpecification shall be used to initialize the ImplementationDataTypeElement of category UNION.

To summarize the initialization issue, a Wrapped Union Data Type is modeled as a structure of two elements and requires a RecordValueSpecification that in turn aggregates two ValueSpecifications, one for the Member Selector that shall have no other value than 1, and one for the “payload”.

The structure of the second ValueSpecification depends on the data type used for the first element of the “payload”.

The following example shows a simplified and stripped-down (e.g. without the SwDataDefProps required to make the model complete) model of a Wrapped Union Data Type.

5.2.7.6 Optional Element Structure

As already mentioned in section 2.8, there are use cases for structured data types that contain optional elements that may or may not exist at a given time.

These data types require a specific modeling on both the level of ApplicationDataType and the level of ImplementationDataType.

[TPS_SWCT_01775] Structured data types with optional elements ⌈A structured data type that contains at least one optional element shall be called an Optional Element Structure.⌋*(RS_SWCT_03320)*

On the level of ApplicationDataType, the existence of optional elements is signaled by setting the attribute ApplicationRecordElement.isOptional to true. For more details, please refer to section 5.2.4.2.2.

The description of how an Optional Element Structure shall be modeled using ImplementationDataType can be found in section 5.2.5.3.1.

5.3 Data Prototypes

5.3.1 Overview

[TPS_SWCT_01264] Data prototypes implement a role of a data type ⌈Generally speaking, a data prototype represents the implementation of a role of a data type within the definition of another data type, e.g. a “typed” data object declared within a software component or a port interface.

This means formally that it has an is-of-type relation to a data type and is usually aggregated by another element, e.g. the internal behavior or a port interface.⌋*()*

In the meta-model, various kinds of data prototypes are derived from the abstract DataPrototype as shown in figure 5.19.

The reason for the introduction of this hierarchy was the distinction between AutosarDataPrototype (which can be used for the application and implementation types as well) and ApplicationCompositeElementDataPrototype (which is restricted to be used within the application types).

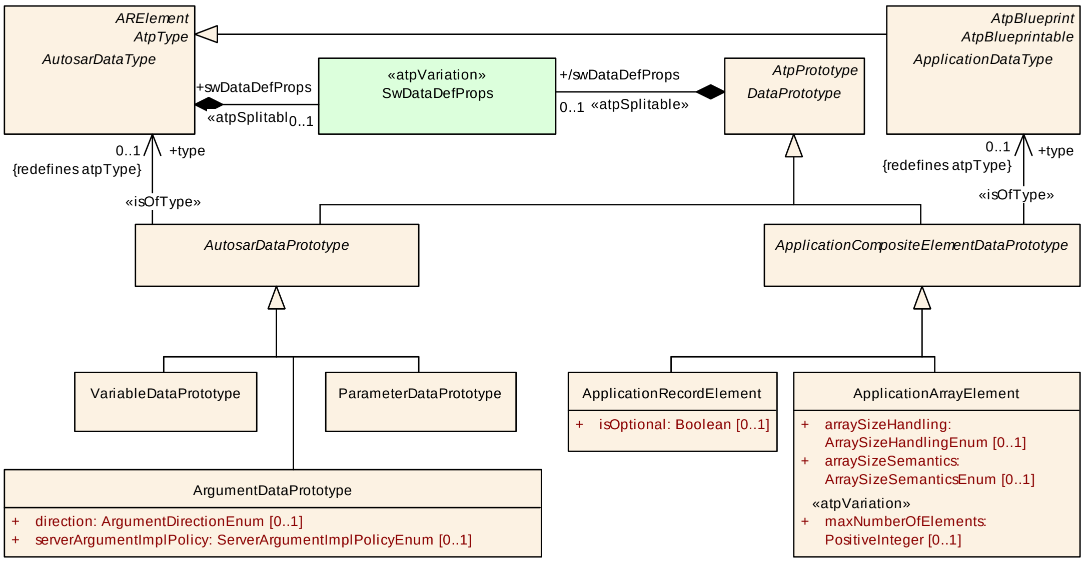


Figure 5.19: Data Prototypes Overview

Because these DataPrototypes are modeled as own meta-classes it is possible to define own attributes for them (on M2) which (in the M1 model) could extend or constrain the attribute values already set via the corresponding data type.

[TPS_SWCT_01265] DataPrototype aggregates an own set of SwDataDefProps
⌈This mechanism is used here in the way that DataPrototype aggregates an own set of SwDataDefProps. Thus, each kind of DataPrototype has the ability to extend or even overwrite the SwDataDefProps already defined by its ApplicationDataType or ImplementationDataType.

This mechanism, if carefully applied, allows for a better reuse of data types because they can be kept free of the properties which vary according to the context or are defined in later project phases.⌋*()*

Class	***DataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	DataPrototype (abstract)			
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps

Table 5.30: DataPrototype

Class	***AutosarDataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ArgumentDataPrototype, ParameterDataPrototype, VariableDataPrototype			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	AutosarDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table 5.31: AutosarDataPrototype

Class	***ApplicationCompositeElementDataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	This class represents a data prototype which is aggregated within a composite application data type (record or array). It is introduced to provide a better distinction between target and context in instance Refs.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ApplicationArrayElement, ApplicationRecordElement			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	ApplicationDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table 5.32: ApplicationCompositeElementDataPrototype

Chapter 5.4 describes more details about this aspect of the meta-model.

[TPS_SWCT_01445] Applicability of `SwDataDefProps` for `DataPrototype`s ⌈The applicability of `SwDataDefProps` for `DataPrototype`s shall follow the same rules as for the `category`s of the corresponding `AutosarDataType`s.⌋*()*

The applicability of `SwDataDefProps` for `DataPrototype`s is documented in [constr_1007].

Further information can be found in [constr_1289] and [constr_1288].

Please note that [constr_1289] does not include the `ApplicationRecordElement` and `ApplicationArrayElement` because these specializations of `ApplicationCompositeElementDataPrototype` are already part of [constr_1007]. The same applies for [constr_1288] which does not include the `ImplementationDataTypeElement`.

[constr_1289] Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes ⌈

Attributes of SwDataDefProps	Root El.			Attribute Existence per Category												
	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP	CUBOID	CUBE_4	CUBE_5
additionalNativeTypeQualifier																
annotation	x	x	x	*	*	*	*	*	*	*	*	*	*	*	*	*
baseType																
compuMethod																
dataConstr.dataConstrRule.physConstrs	x	x		0..1	0..1		0..1		0..1			0..1	0..1	0..1	0..1	0..1
dataConstr.dataConstrRule.internalConstrs	x	x		d/c[13]	d/c		d/c		d/c			d/c	d/c	d/c	d/c	d/c
displayFormat	x	x		0..1	0..1		0..1	0..1	0..1			0..1	0..1	0..1	0..1	0..1
displayPresentation	x	x		0..1	0..1		0..1			0..1	0..1	0..1	0..1	0..1	0..1	0..1
implementationDataType																
invalidValue																
stepSize	x	x	x	0..1	0..1		0..1			0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAddrMethod	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swBitRepresentation																
swCalibrationAccess	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swCalprmAxisSet																
swCalprmAxisSet.swCalprmAxis/SwAxisGrouped.swCalprmRef		x	x				0..1					0..1	0..1	0..1	0..1	0..1
swCalprmAxisSet.swCalprmAxis/SwAxisIndividual.swVariableRef		x	x				0..1			0..1	0..1	0..1	0..1	0..1	0..1	0..1
swCalprmAxisSet.swCalprmAxis/SwAxisGrouped.sharedAxisType																
swCalprmAxisSet.swCalprmAxis/SwAxisIndividual.inputVariableType																
swCalprmAxisSet.swCalprmAxis/SwAxisIndividual.unit																
swComparisonVariable			x									0..1	0..1	0..1	0..1	0..1
swDataDependency	x	x		0..1								0..1	0..1	0..1	0..1	0..1
swHostVariable																
swImplPolicy	x			0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution																
swInterpolationMethod	x	x	x	0..1						0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIsVirtual	x	x		0..1					0..1			0..1	0..1	0..1	0..1	0..1
swPointerTargetProps																

▽

[13] don't care

△

Attributes of SwDataDefProps	Root El.			Attribute Existence per Category												
	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP	CUBOID	CUBE_4	CUBE_5
swRecordLayout																
swRefreshTiming	x	x		0..1	0..1			0..1	0..1							
swTextProps																
swValueBlockSize																
swValueBlockSizeMult																
unit																
valueAxisDataType																

This rule shall be imposed at any time in the workflow.

⌋()

[constr_1288] Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes ⌈

Attributes of SwDataDefProps	Root Element			Attribute Existence per Category						
	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
additionalNativeTypeQualifier										
annotation	x	x	*	*	*	*	*	*	*	*
baseType										
compuMethod										
dataConstr.dataConstrRule.physConstrs	x	x		d/c[14]			d/c			d/c
dataConstr.dataConstrRule.internalConstrs	x	x		0..1			0..1			0..1
displayFormat	x	x		0..1			0..1	0..1	0..1	0..1
displayPresentation	x	x		0..1			0..1			0..1
implementationDataType										
invalidValue										

▽

[14] don't care

△

Attributes of SwDataDefProps	Root Element			Attribute Existence per Category						
stepSize	x	x		0..1						0..1
swAddrMethod	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swBitRepresentation										
swCalibrationAccess	x	x		0..1			0..1	0..1	0..1	0..1
swCalprmAxisSet										
swComparisonVariable										
swDataDependency										
swHostVariable										
swImplPolicy	x			0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution										
swInterpolationMethod										
swIsVirtual										
swPointerTargetProps										
swPointerTargetProps.swDataDefProps										
swPointerTargetProps.functionPointerSignature										
swRecordLayout										
swRefreshTiming	x	x		0..1			0..1	0..1	0..1	0..1
swTextProps										
swValueBlockSize										
swValueBlockSizeMult										
unit										
valueAxisDataType										

This rule shall be imposed **at any time in the workflow**.

⌋()

[TPS_SWCT_01266] Three non-abstract classes derived from AutosarDataPrototype ⌈There are three non-abstract classes derived from AutosarDataPrototype which reflect the main use cases in the SWC-Template:

- Operation arguments (ArgumentDataPrototype) in a client-server interface.
- Variables (VariableDataPrototype) which are changed by the application software at runtime.
- Parameters (ParameterDataPrototype) which are constant (except for calibration access) from the application point of view.

⌋()

Class	**VariableDataPrototype**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes
Note	A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.

▽

△

Class	VariableDataPrototype			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstanceMemory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvDataDescriptor.bulkNvBlock, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface.nvData, SenderReceiverInterface.dataElement, ServiceInterface.event, SwcInternalBehavior.arTypedPerInstanceMemory, SwcInternalBehavior.explicitInterRunnableVariable, SwcInternalBehavior.implicitInterRunnableVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table 5.33: VariableDataPrototype

Class	ParameterDataPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A ParameterDataPrototype represents a formalized generic piece of information that is typically immutable by the application software layer, but mutable by measurement and calibration tools. ParameterDataPrototype is used in various contexts and the specific context gives the otherwise generic ParameterDataPrototype a dedicated semantics.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswInternalBehavior.perInstanceParameter, *InternalBehavior*.constantMemory, NvBlockDescriptor.romBlock, ParameterInterface.parameter, SwcInternalBehavior.perInstanceParameter, SwcInternalBehavior.sharedParameter			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype

Table 5.34: ParameterDataPrototype

[TPS_SWCT_01267] `DataPrototype` can be aggregated in different roles ⌈Note that even though the meta-classes `VariableDataPrototype` and `ParameterDataPrototype` already express specific use cases of the underlying data type the same `DataPrototype` can still be aggregated in different roles, e.g. in the `SwcInternalBehavior` to express different methods how to access it.⌋*()*

An example is the aggregation of `VariableDataPrototype` by `SwcInternalBehavior` in the roles of either `implicitInterRunnableVariable` or `explicitInterRunnableVariable`. Find more information concerning these use cases in chapter 7.

[TPS_SWCT_01268] Definition of `initValue` for a `VariableDataPrototype` or a `ParameterDataPrototype` ⌈It is possible to assign an `initValue` for both a `VariableDataPrototype` and a `ParameterDataPrototype`.⌋*()*

This aspect is sketched in Figure 5.20.

[TPS_SWCT_01269] In `PortInterface`s, initial values defined for `DataPrototype`s are ignored ⌈These `initValue`s have no meaning for `DataPrototype`s within `PortInterface`s because in this case a more specific definition of initial values via the so-called `ComSpec` is required.⌋*()*

For more information, please refer to chapter 4.5.

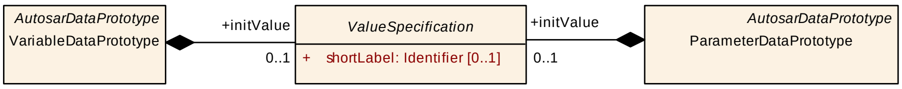


Figure 5.20: Initial value for AutosarDataPrototypes

Find more information about the interpretation of initValue in section 5.7.

[constr_1416] Existence of ApplicationArrayElement.maxNumberOfElements ⌈The attribute ApplicationArrayElement.maxNumberOfElements shall exist **at the time when the contract phase generation is executed** for all ApplicationArrayElements defined in the scope of an ApplicationArrayDataType where the attribute ApplicationArrayDataType.dynamicArraySizeProfile does not exist.⌋*()*

This means that for fixed-size array data types the attribute ApplicationArrayElement.maxNumberOfElements shall be defined for every dimension of the fixed-size array data.

5.3.2 Data Constraints for DataPrototypes typed by Array DataTypes

There are cases where it should be possible to reference different DataConstrs from DataPrototypes of category ARRAY typed by either an ApplicationArrayDataType or an ImplementationDataType of category ARRAY.

For example, consider a predefined AutosarDataType of category ARRAY with uint8 elements, but without the definition of DataConstrs.

This AutosarDataType is instantiated several times by means of the existence of DataPrototypes of category ARRAY.

Each of the derived DataPrototypes of category ARRAY may have individual upper and lower limits which apply to all elements, e.g. one DataPrototype has a range of 0..3 while another has a range of 0..7 for all elements.

This use case occurs, for example, when communicating array-data with different ranges over the network.

The motivation to specify the DataConstr at the DataPrototype of category ARRAY is:

- Make explicit that all derived DataPrototypes are based on the same AutosarDataType.
- Reduce the number of AutosarDataTypes in the model and in the code.
- Specify the DataConstrs only once where needed.
- Make explicit that all array elements have the same DataConstrs.

The same motivation also applies to the SwDataDefProps.displayFormat and SwDataDefProps.stepSize attributes of DataPrototypes of category ARRAY.

AUTOSAR supports this use case under the following conditions:

[constr_1407] Definition of SwDataDefProps.dataConstr depending on the capabilities of the data type ⌈The definition of a SwDataDefProps.dataConstr according to [constr_1288] and [constr_1289] is only supported for a DataPrototype of category ARRAY if the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY also supports the specification of a SwDataDefProps.dataConstr.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The prioritization of SwDataDefProps.dataConstr for a DataPrototype of category ARRAY follows the spirit given in [constr_1015] for the inheritance of DataPrototypes from AutosarDataTypes.

[TPS_SWCT_01796] Prioritization of SwDataDefProps.dataConstr for a DataPrototype of category ARRAY ⌈A SwDataDefProps.dataConstr specified for a DataPrototype of category ARRAY refines the SwDataDefProps.dataConstr specified at the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY.⌋*()*

[constr_1408] Definition of SwDataDefProps.displayFormat depending on the capabilities of the data type ⌈The definition of a SwDataDefProps.displayFormat according to [constr_1288] and [constr_1289] is only supported for a DataPrototype of category ARRAY if the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY also supports the specification of a SwDataDefProps.displayFormat.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The prioritization of SwDataDefProps.displayFormat for a DataPrototype of category ARRAY follows the spirit given in [constr_1015] for the inheritance of DataPrototypes from AutosarDataTypes.

[TPS_SWCT_01797] Prioritization of SwDataDefProps.displayFormat for a DataPrototype of category ARRAY ⌈A SwDataDefProps.displayFormat specified for a DataPrototype of category ARRAY refines the SwDataDefProps.displayFormat specified at the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY.⌋*()*

[constr_1413] Definition of SwDataDefProps.stepSize depending on the capabilities of the data type ⌈The definition of a SwDataDefProps.stepSize according to [constr_1288] and [constr_1289] is only supported for a DataPrototype of category ARRAY if the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY also supports the specification of a SwDataDefProps.stepSize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The prioritization of SwDataDefProps.stepSize for a DataPrototype of category ARRAY follows the spirit given in [constr_1015] for the inheritance of DataPrototypes from AutosarDataTypes.

[TPS_SWCT_01798] Prioritization of SwDataDefProps.stepSize for a DataPrototype of category ARRAY ⌈A SwDataDefProps.stepSize specified for a DataPrototype of category ARRAY refines the SwDataDefProps.stepSize specified at the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY.⌋*()*

[constr_1409] Definition of SwDataDefProps.dataConstr depending on the capabilities of the element data type ⌈The definition of a SwDataDefProps.dataConstr according to [constr_1007] and [constr_1009] is only supported for an ApplicationArrayDataType or an ImplementationDataType of category ARRAY if the aggregated ApplicationArrayDataType.element or ImplementationDataType.subElement also supports the specification of a SwDataDefProps.dataConstr.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1718] Inheritance of SwDataDefProps.dataConstr from an array data type to the array elements ⌈A SwDataDefProps.dataConstr specified for an ApplicationArrayDataType or ImplementationDataType of category ARRAY applies to all array leaf elements represented by (potentially multiple levels of) ApplicationArrayDataType.element or ImplementationDataType.subElement.

In this case, the ApplicationArrayDataType.element or ImplementationDataType.subElement shall not have an own SwDataDefProps.dataConstr. This also applies for multi-dimensional array data types.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1410] Definition of SwDataDefProps.displayFormat depending on the capabilities of the element data type ⌈The definition of a SwDataDefProps.displayFormat according to [constr_1007] and [constr_1009] is only supported for an ApplicationArrayDataType or an ImplementationDataType of category ARRAY if the aggregated ApplicationArrayDataType.element or ImplementationDataType.subElement also supports the specification of a SwDataDefProps.displayFormat.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1719] Inheritance of SwDataDefProps.displayFormat from an array data type to the array elements ⌈A SwDataDefProps.displayFormat specified for an ApplicationArrayDataType or ImplementationDataType of category ARRAY applies to all array leaf elements represented by (potentially multiple levels of)

ApplicationArrayDataType.element or ImplementationDataType.subElement.

In this case, the ApplicationArrayDataType.element or ImplementationDataType.subElement shall not have an own SwDataDefProps.displayFormat. This also applies for multi-dimensional array data types.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1414] Definition of SwDataDefProps.stepSize depending on the capabilities of the element data type ⌈The definition of a SwDataDefProps.stepSize according to [constr_1007] and [constr_1009] is only supported for an ApplicationArrayDataType or an ImplementationDataType of category ARRAY if the aggregated ApplicationArrayDataType.element or ImplementationDataType.subElement also supports the specification of a SwDataDefProps.stepSize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1720] Inheritance of SwDataDefProps.stepSize from an array data type to the array elements ⌈A SwDataDefProps.stepSize specified for an ApplicationArrayDataType or ImplementationDataType of category ARRAY applies to all array leaf elements represented by (potentially multiple levels of) ApplicationArrayDataType.element or ImplementationDataType.subElement.

In this case, the ApplicationArrayDataType.element or ImplementationDataType.subElement shall not have an own SwDataDefProps.stepSize. This also applies for multi-dimensional array data types.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

5.3.3 Reference to Data Prototypes

This chapter explains the various patterns for referencing DataPrototypes.

[TPS_SWCT_01446] References to a DataPrototype may or may not imply the necessity for using an instanceRef ⌈As references to a DataPrototype may or may not imply the necessity for using an instanceRef this would mean that in some places the meta-model would have to implement both variants depending on the use case. To avoid this, AUTOSAR defines a unified reference implementation for VariableDataPrototypes and ParameterDataPrototypes.⌋()

5.3.3.1 AUTOSAR Variable Ref

[TPS_SWCT_01270] **AutosarVariableRef** ⌈With the advent of AutosarVariableRef it is possible to implement a uniform reference to a VariableDataPrototype that covers all foreseen use cases:

- Reference to a localVariable, no AtpInstanceRef required.
- Reference to an autosarVariable (which involves an AtpInstanceRef).
- Reference to the internal structure of a VariableDataPrototype implemented using a composite ImplementationDataType.

⌋*()*

Class	**AutosarVariableRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents a reference to a variable within AUTOSAR which can be one of the following use cases: localVariable: • localVariable which is used as whole (e.g. InterRunnableVariable, inputValue for curve) autosarVariable: • a variable provided via Port which is used as whole (e.g. dataAccesspoints) • an element inside of a composite local variable typed by ApplicationDatatype (e.g. inputValue for a curve) • an element inside of a composite variable provided via Port and typed by ApplicationDatatype (e.g. inputValue for a curve) autosarVariableInImplDatatype: • an element inside of a composite local variable typed by ImplementationDatatype (e.g. nvram Data mapping) • an element inside of a composite variable provided via Port and typed by Implementation Datatype (e.g. inputValue for a curve)			
Base	*ARObject*			
Aggregated by	InstantiationDataDefProps.variableInstance, NvBlockDataMapping.nvRamBlockElement, NvBlockData Mapping.readNvData, NvBlockDataMapping.writtenNvData, NvBlockDataMapping.writtenReadNvData, RoleBasedDataAssignment.usedDataElement, SwVariableRefProxy.autosarVariable, VariableAccess. accessedVariable			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
autosarVariable	DataPrototype	0..1	iref	This references a variable which is provided by a port and/or which is part of a CompositeDataType. **InstanceRef implemented by:**VariableInAtomicSWC TypeInstanceRef
autosarVariable InImplDatatype	ArVariableIn ImplementationData InstanceRef	0..1	aggr	This is used if the target variable is inside of variableData Prototype typed by an ImplementationDataType.
localVariable	VariableDataPrototype	0..1	ref	This reference is used if the variable is local to the current component. It would also be possible to use the instance refence here. Such an instance ref would not have a contextElement, since the current instance is the context. But the local instance is a special case which may provide further optimization. Therefore an explicit reference is provided for this case.

Table 5.35: AutosarVariableRef

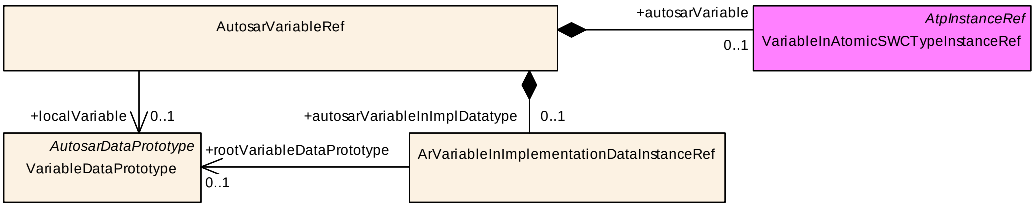


Figure 5.21: Implementation of AutosarVariableRef

Rules for the modeling and semantics of an AtpInstanceRef are defined in [11].

[constr_2536] Target of an autosarVariable in AutosarVariableRef shall refer to a variable ⌈The target of autosarVariable (which in fact is an instance ref) in AutosarVariableRef shall either be or be nested in VariableDataPrototype. This means that the target shall either be a VariableDataPrototype or an ApplicationCompositeElementDataPrototype that in turn is owned by a VariableDataPrototype.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

5.3.3.2 AUTOSAR Parameter Ref

[TPS_SWCT_01271] AutosarParameterRef ⌈With the advent of AutosarParameterRef, it is possible to implement a uniform reference to a ParameterDataPrototype that covers all foreseen use cases:

- Reference to a localParameter, no AtpInstanceRef required.
- Reference to an autosarParameter (which involves an AtpInstanceRef).

⌋*()*

Please note that there is a very limited amount of use-cases available where the AutosarParameterRef can (with the active consent of the AUTOSAR standard) reference a VariableDataPrototype.

[constr_1173] Applicability of AutosarParameterRef referencing a VariableDataPrototype ⌈A reference from AutosarParameterRef to VariableDataPrototype is **only** applicable if the AutosarParameterRef is used in the context of SwAxisGrouped **at the time when the contract phase generation is executed**.⌋*()*

For example, the use case referenced in [constr_1173] applies if it is required to store a grouped axis in a variable in order to adapt the axis during run-time of the ECU by a dedicated algorithm. Note that in all cases where [constr_1173] does not apply [constr_2535] shall be fulfilled.

Class	AutosarParameterRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents a reference to a parameter within AUTOSAR which can be one of the following use cases: localParameter: • localParameter which is used as whole (e.g. sharedAxis for curve) autosarVariable: • a parameter provided via PortPrototype which is used as whole (e.g. parameterAccess) • an element inside of a composite local parameter typed by ApplicationDatatype (e.g. sharedAxis for a curve) • an element inside of a composite parameter provided via Port and typed by ApplicationDatatype (e.g. sharedAxis for a curve) autosarParameterInImplDatatype: • an element inside of a composite local parameter typed by ImplementationDatatype • an element inside of a composite parameter provided via PortPrototype and typed by ImplementationDatatype			
Base	*ARObject*			
Aggregated by	InstantiationDataDefProps.parameterInstance, ParameterAccess.accessedParameter, RoleBasedDataAssignment.usedParameterElement, SwCalprmRefProxy.arParameter			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
autosarParameter	DataPrototype	0..1	iref	This instance reference is used if the calibration parameter is either imported via a port or is part of a composite data structure. **InstanceRef implemented by:**ParameterInAtomicSWCTypeInstanceRef
localParameter	DataPrototype	0..1	ref	In the majority of cases this reference goes to ParameterDataPrototypes rather than VariableDataPrototypes. Pointing the reference to a VariableDataPrototype is limited to special use cases, e.g. if the AutosarParameterRef is used in the context of an SwAxisGrouped. This reference is used if the arParameter is local to the current component. Of course, it would technically also be feasible to use an InstanceRef for this case. However, the InstanceRef would not have a contextElement (because the current instance is the context). Hence, the local instance is a special case which may provide further optimization. Therefore an explicit reference is provided for this case.

Table 5.36: AutosarParameterRef

[constr_2535] Target of an `autosarParameter` in `AutosarParameterRef` shall refer to a parameter ⌈Except for the specifically described cases where [constr_1173], applies the target of `autosarParameter` (which in fact is an instance ref) in `AutosarParameterRef` shall either be or be nested in `ParameterDataPrototype`. This means that the target shall either be a `ParameterDataPrototype` or an `ApplicationCompositeElementDataPrototype` that in turn is owned by a `ParameterDataPrototype`.

This rule shall be imposed **`at the time when the contract phase generation is executed`**.⌋*()*

5.3.3.3 Modeling Approach

The attribute Ref.index shall be used whenever a model element that represents an array is referenced in a scalar context, i.e. when the reference is supposed to identify a specific array element.

Primitive	**Ref**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This primitive denotes a name based reference. For detailed syntax see the xsd.pattern. • first slash (relative or absolute reference) [optional] • Identifier [required] • a sequence of slashes and Identifiers [optional] This primitive is used by the meta-model tools to create the references. **Tags:** xml.xsd.customType=REF xml.xsd.pattern=/?[a-zA-Z][a-zA-Z0-9_]{0,127}(/[a-zA-Z][a-zA-Z0-9_]{0,127})* xml.xsd.type=string			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
base	Identifier	0..1	attr	This attribute reflects the base to be used for this reference. **Tags:**xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:** atp.Status=draft xml.attribute=true
index	PositiveInteger	0..1	attr	This attribute supports the use case to point on specific elements in an array. This is in particular required if arrays are used to implement particular data objects. The counting of array indices starts with the value 0, i.e. the index of the first array element is 0. **Tags:**xml.attribute=true

Table 5.37: Ref

A very typical example for such a situation is the access to an element of an ApplicationArrayDataType by means of a reference to ApplicationArrayElement in which the index attribute is used.

The usage of the index attribute does not make sense if the context of the access is already scalar, e.g. accessing an ApplicationRecordElement in the context of an ApplicationRecordDataType, the usage of the attribute index in a context that is already scalar by nature would also be misleading and is therefore prohibited.

[constr_1161] Applicability of the attribute Ref.index ⌈The usage of attribute Ref.index is limited to references to the following meta-classes:

- ApplicationArrayElement
- Sub-classes of AbstractImplementationDataTypeElement.

This rule shall be imposed **at any time in the workflow**.⌋*()*

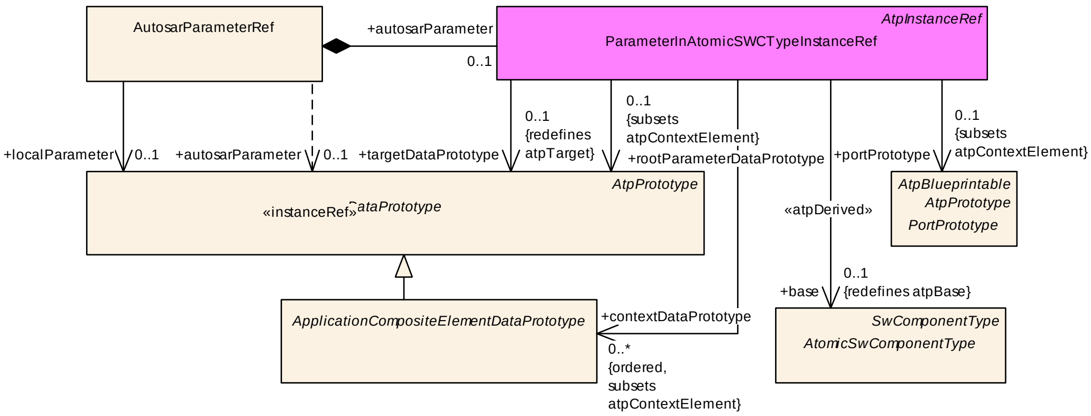


Figure 5.22: Implementation of the InstanceRef for AutosarParameterRef

Note: If the target of the AtpInstanceRef is an AutosarDataPrototype then the rootVariableDataPrototype shall not exist.

The implementation of the AtpInstanceRefs for AutosarVariableRef and AutosarParameterRef probably needs some clarification regarding the references to DataPrototypes.

[TPS_SWCT_01374] Implementation of AutosarParameterRef ⌈The reference to rootParameterDataPrototype is **not** redundant. It is required for identifying the autosarParameter itself in a ParameterInterface for the case that the targetDataPrototype is an ApplicationCompositeElementDataPrototype.⌋*()*

As explained before, the implementation of AutosarParameterRef in a specific case is subject to [constr_1173].

[constr_1608] Existence of rootParameterDataPrototype ⌈The reference rootParameterDataPrototype shall exist **at any time in the workflow** if and only if

- AutosarDataType of the autosarParameter is a composite data type and
- targetDataPrototype refers to a DataPrototype inside the rootParameterDataPrototype.

⌋*()*

Note: If the target of the AtpInstanceRef is an AutosarDataPrototype then the rootParameterDataPrototype shall not exist.

[TPS_SWCT_01375] Implementation of AutosarVariableRef ⌈The reference to rootVariableDataPrototype is **not** redundant. It is required for identifying the autosarVariable itself in a SenderReceiverInterface or NvDataInterface for the case that the targetDataPrototype is an ApplicationCompositeElementDataPrototype.⌋*()*

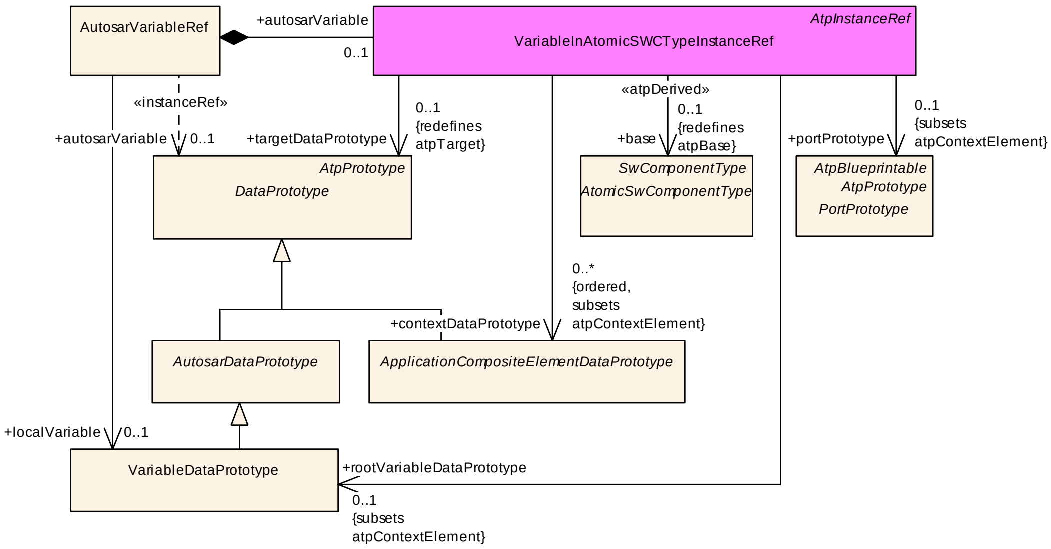


Figure 5.23: Implementation of the InstanceRef for AutosarVariableRef

[constr_1609] Existence of rootVariableDataPrototype ⌈The reference rootVariableDataPrototype shall exist **at any time in the workflow** if and only if

- the AutosarDataType of the autosarVariable is a composite data type and
- the targetDataPrototype refers to a DataPrototype inside the rootVariableDataPrototype.

⌋*()*

5.3.3.4 Access into VariableDataPrototype typed by an ImplementationDataType

The meta-class ArVariableInImplementationDataInstanceRef, despite the name, has formally no relationship to AtpInstanceRef. Therefore, the following definition applies:

[TPS_SWCT_01681] Context path in ArVariableInImplementationDataInstanceRef ⌈The references in the roles

- portPrototype
- rootVariableDataPrototype
- ordered collection of contextDataPrototype
- targetDataPrototype

constitute the path leading from the root to the specified inner instance of a dataElement inside a VariableDataPrototype typed by an ImplementationDataType.⌋*()*

This relation is also depicted in Figure 5.24.

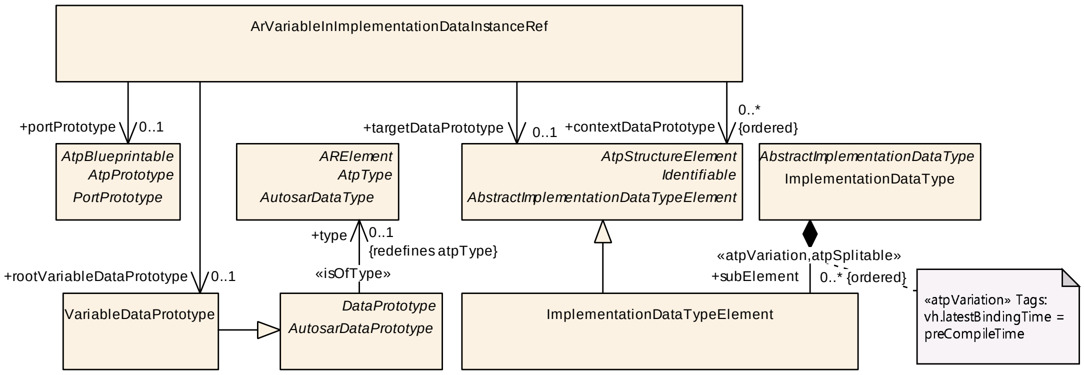


Figure 5.24: Implementation of ArVariableInImplementationDataInstanceRef

[constr_1423] Completeness of references ArVariableInImplementationDataInstanceRef.contextDataPrototype ⌈The reference ArVariableInImplementationDataInstanceRef.contextDataPrototype shall be defined for

- each *leaf* (i.e. the end of a chain of aggregating elements) ImplementationDataTypeElement of category TYPE_REFERENCE in a chain of referencing ImplementationDataTypes which is not the targetDataPrototype
- and each ImplementationDataTypeElement owned by an ImplementationDataType or ImplementationDataTypeElement of category ARRAY in a chain of referencing ImplementationDataTypes

starting from the ImplementationDataTypes of the rootVariableDataPrototype down to the leaf ImplementationDataTypeElement which is typed (directly or indirectly via ImplementationDataType of category TYPE_REFERENCE) by the ImplementationDataType of the targetDataPrototype.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

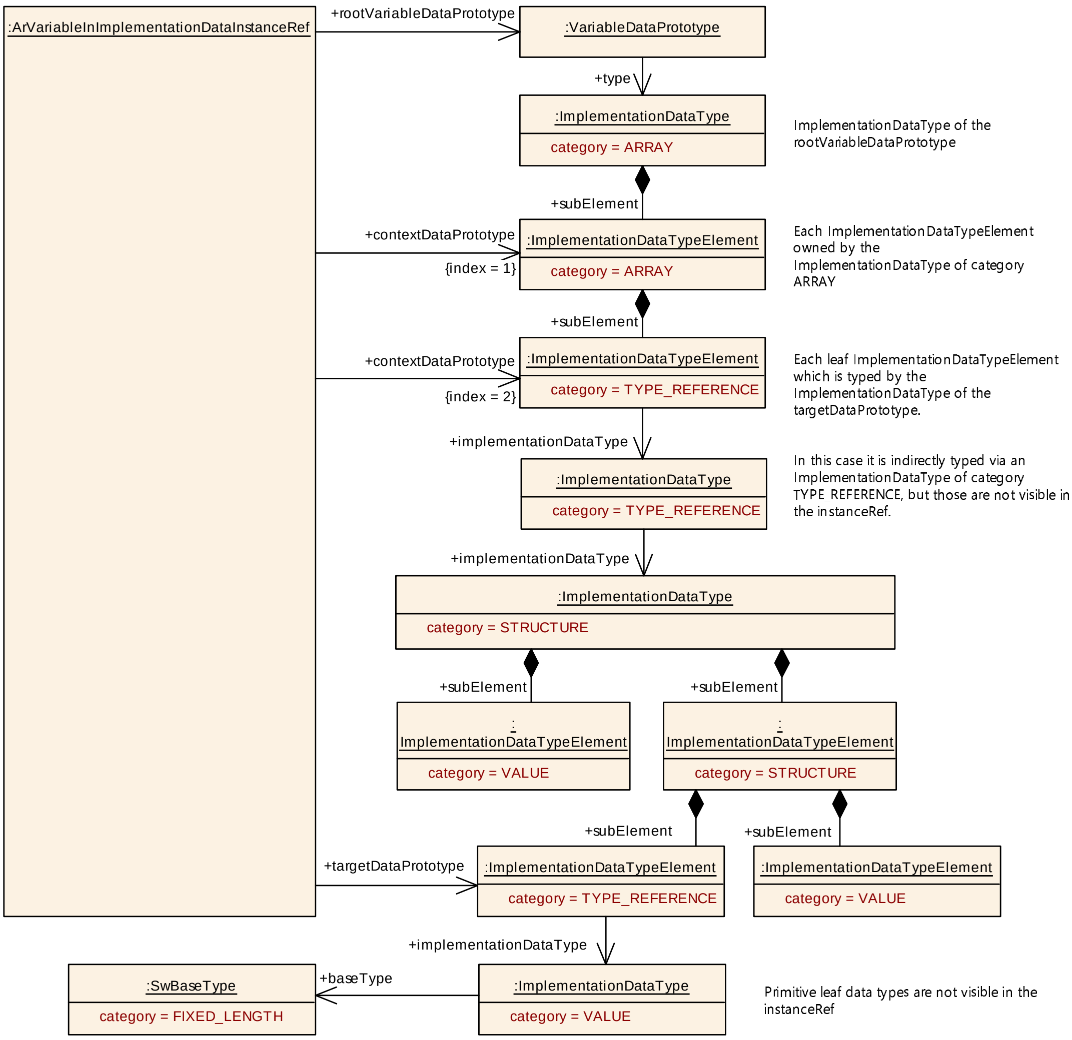


Figure 5.25: Example for the usage of ArVariableInImplementationDataInstanceRef

[constr_1424] Existence of ArVariableInImplementationDataInstanceRef.contextDataPrototype ⌈The attribute ArVariableInImplementationDataInstanceRef.contextDataPrototype shall only exist **at any time in the workflow** for an ImplementationDataTypeElement category TYPE_REFERENCE or ARRAY.⌋*()*

Technically, it would be possible to avoid the context for a one-dimensional array in the hierarchy. The context is still required because then the rule for the existence of contexts becomes much simpler.

Figure 5.25 contains an example of a nested ImplementationDataType along with the application of ArVariableInImplementationDataInstanceRef. The example contains both cases for the definition of a contextDataPrototype mentioned in [constr_1423].

Class	ArVariableInImplementationDataInstanceRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents the ability to navigate into a data element inside of an VariableDataPrototype which is typed by an ImplementationDatatype. Note that it shall not be used if the target is the VariableDataPrototype itself (e.g. if its a primitive). Note that this class follows the pattern of an InstanceRef but is not implemented based on the abstract classes because the ImplementationDataType isn't either, especially because ImplementationDataType Element isn't derived from AtpPrototype.			
Base	*ARObject*			
Aggregated by	AutosarVariableRef.autosarVariableInImplDatatype, ImplementationDataTypeSubElementRef.implementationDataTypeElement			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
contextDataPrototype (ordered)	AbstractImplementationDataTypeElement	*	ref	This is a context in case there are subelements with explicit types. The reference has to be ordered to properly reflect the nested structure. **Tags:**xml.sequenceOffset=30
portPrototype	PortPrototype	0..1	ref	This is the port providing/receiving the root of the variable **Tags:**xml.sequenceOffset=10
rootVariableDataPrototype	VariableDataPrototype	0..1	ref	This refers to the VariableDataPrototype typed by the ImplementationDatatype in which the target can be found. **Tags:**xml.sequenceOffset=20
targetDataPrototype	AbstractImplementationDataTypeElement	0..1	ref	This reference points to the target ImplementationDataTypeElement. **Tags:**xml.sequenceOffset=40

Table 5.38: ArVariableInImplementationDataInstanceRef

[constr_1911] Existence of ArVariableInImplementationDataInstanceRef.targetDataPrototype ⌈For each ArVariableInImplementationDataInstanceRef, the reference targetDataPrototype shall exist **at the time when the contract phase generation is executed**.⌋*()*

5.3.3.5 Access into ParameterDataPrototype typed by an ImplementationDataType

Please note that it is also possible to access the inside of a nested ParameterDataPrototype typed by an ImplementationDataType in pretty much the same way as this is possible for a VariableDataPrototype typed by an ImplementationDataType.

[TPS_SWCT_01738] Context path in ArParameterInImplementationDataInstanceRef ⌈The references in the roles

- portPrototype
- rootParameterDataPrototype
- ordered collection of contextDataPrototype
- targetDataPrototype

constitute the path leading from the root to the specified inner instance of a parameter inside a ParameterDataPrototype typed by an ImplementationDataType.⌋()

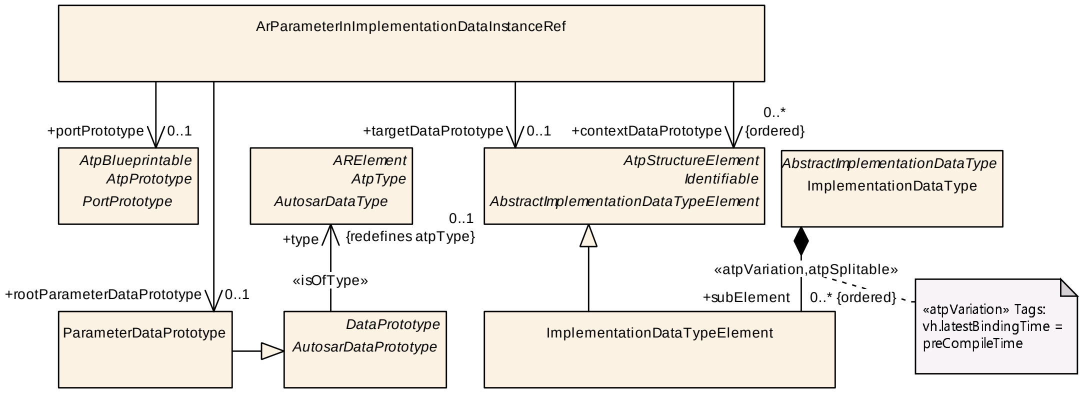


Figure 5.26: Implementation of ArParameterInImplementationDataInstanceRef

This relation is also depicted in Figure 5.26.

[constr_1516] Completeness of references ArParameterInImplementationDataInstanceRef.contextDataPrototype ⌈The reference ArParameterInImplementationDataInstanceRef.contextDataPrototype shall be defined for

- each *leaf* (i.e. the end of a chain of aggregating elements) ImplementationDataTypeElement of category TYPE_REFERENCE in a chain of referencing ImplementationDataTypes which is not the targetDataPrototype
- and each ImplementationDataTypeElement owned by an ImplementationDataType or ImplementationDataTypeElement of category ARRAY in a chain of referencing ImplementationDataTypes

starting from the ImplementationDataTypes of the rootParameterDataPrototype down to the leaf ImplementationDataTypeElement which is typed (directly or indirectly via ImplementationDataType of category TYPE_REFERENCE) by the ImplementationDataType of the targetDataPrototype.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1517] Existence of ArParameterInImplementationDataInstanceRef.contextDataPrototype ⌈The attribute ArParameterInImplementationDataInstanceRef.contextDataPrototype shall only exist **at any time in the workflow** for an ImplementationDataTypeElement category TYPE_REFERENCE or ARRAY.⌋()

Technically, it would be possible to avoid the context for a one-dimensional array in the hierarchy. The context is still required because then the rule for the existence of contexts becomes much simpler.

Class	ArParameterInImplementationDataInstanceRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents the ability to navigate into an element inside of an ParameterDataPrototype typed by an ImplementationDatatype. Note that it shall not be used if the target is the ParameterDataPrototype itself (e.g. if the target is a primitive data type). Note that this class follows the pattern of an InstanceRef but is not implemented based on the abstract classes because the ImplementationDataType isn't either, especially because ImplementationDataType Element (intentionally) isn't derived from AtpPrototype.			
Base	*ARObject*			
Aggregated by	ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
contextData Prototype (ordered)	AbstractImplementation DataTypeElement	*	ref	This is a context in case there are subelements with explicit types. The reference has to be ordered to properly reflect the nested structure.
portPrototype	PortPrototype	0..1	ref	This reference points to the PortPrototype providing/ receiving the root of the parameter.
rootParameter DataPrototype	ParameterData Prototype	0..1	ref	This refers to the ParameterDataPrototype typed by the implementationDataType in which the target can be found.
targetData Prototype	AbstractImplementation DataTypeElement	0..1	ref	This reference points to the target ImplementationData TypeElement.

Table 5.39: ArParameterInImplementationDataInstanceRef

[constr_1912] Existence of reference ArParameterInImplementationDataInstanceRef.targetDataPrototype ⌈For each ArParameterInImplementationDataInstanceRef, the reference targetDataPrototype shall exist **at the time when the contract phase generation is executed**.⌋*()*

5.4 Properties of Data Definitions

5.4.1 Overview

As it has already been shown in the previous chapters, various properties and associations can be attached to the definition of data types as well as prototypes. These are described by the meta-class SwDataDefProps which covers all properties of a particular data object under various aspects.

In general, the properties specified within SwDataDefProps may apply to all kind of data declared within the software-component template and within the basic software module description template as well, e.g. component local data, data used for communication, data used for measurement as well as for calibration.

However, there are constraints for the attributes depending on the role of the data:

The usage of SwDataDefProps in the context of the ComSpec networkRepresentation (which is configurable in several roles and different contexts) implements a "fork" in the dependency graph that is coded into [constr_1015] and this "fork" should not have an impact on elements on the left of the `ComSpec` `networkRepresentation` column.

[constr_1015] Prioritization of `SwDataDefProps` ⌈

Attributes of SwDataDefProps	Usage For			Place of Setting										
	RTE	A2L	Other Usage	ApplicationDataType	ImplementationDataType	DataPrototype	InstantiationDataDefProps	ParameterAccess	SwServiceArg	FlatInstanceDescriptor	McDataInstance	SwSystemconst	PerInstanceMemory	ComSpec networkRepresentation
additionalNativeTypeQualifier	x		x	NA	D	I	NA	NA	D	NA	S	NA	NA	NA
annotation			x	D	A	A	A	A	D	NA	A	D	NA	A
baseType	x	x	x	NA	D	I	I	I	D	NA	S	M	NA	R
compuMethod	x	x	x	D	AI	I	I	NA	I	AI	S	D	NA	R
dataConstr	x	x	x	D	C	R	R	I	R	NA	S	D	NA	NA
displayFormat		x		D	A	R	R	I	R	NA	S	D	NA	NA
displayPresentation	x	x	x	D	A	R	R	NA	NA	NA	S	NA	NA	NA
implementationDataType	x		x	NA	D	I	I	I	D	NA	NA	NA	NA	NA
invalidValue	x	x		D	A	I	I	NA	NA	NA	S	NA	NA	D
stepSize		x		D	A	A	A	A	NA	A	S	NA	NA	NA
swAddrMethod	x	x	x	D	R	R	R	NA	NA	R	NA	NA	D	NA
swAlignment	x		x	NA	D	R	R	NA	NA	NA	NA	NA	NA	NA
swBitRepresentation		x	x	NA	NA	NA	NA	NA	NA	NA	D	NA	NA	NA
swCalibrationAccess	x	x		D	R	R	R	NA	R	R	S	D	NA	NA
swCalprmAxisSet	x	x		D	NA	I	I	I	NA	NA	S	NA	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisGrouped.swCalprmRef		x		NA	NA	NA	D	R	NA	NA	S	NA	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisIndividual.swVariableRef		x		NA	NA	NA	D	R	NA	NA	S	NA	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisGrouped.sharedAxisType		x		D	NA	NA	NA	NA	NA	NA	S	NA	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisIndividual.inputVariableType		x		D	NA	NA	NA	NA	NA	NA	S	NA	NA	NA
swCalprmAxisSet/SwAxisIndividual.unit		opt.		D	NA	I	I	I	I	NA	S	NA	NA	NA
swComparisonVariable		x		NA	NA	NA	NA	D	NA	NA	S	NA	NA	NA
swDataDependency		x	x	NA	NA	D	R	NA	NA	NA	S	NA	NA	NA
swHostVariable		x	x	NA	NA	NA	NA	NA	NA	NA	D	NA	NA	NA
swImplPolicy	x		x	D	A	A	NA	NA	D	NA	NA	NA	NA	NA

▽

△

Attributes of SwDataDefProps	Usage For			Place of Setting										
	RTE	A2L	Other Usage	ApplicationDataType	ImplementationDataType	DataPrototype	InstantiationDataDefProps	ParameterAccess	SwServiceArg	FlatInstanceDescriptor	McDataInstance	SwSystemconst	PerInstanceMemory	ComSpec networkRepresentation
swIntendedResolution			x	D[15]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
swInterpolationMethod			x	D	I	R	R	R	NA	NA	S	NA	NA	NA
swIsVirtual		x		NA	NA	D	R	NA	NA	NA	S	NA	NA	NA
swPointerTargetProps			x	NA	D	I	NA	NA	D	NA	NA	NA	NA	NA
swRecordLayout	x	x	x	D	NA	I	I	I	NA	NA	S	NA	NA	NA
swRefreshTiming		x		D	R	R	R	NA	R	R	R	NA	NA	NA
swTextProps		x	x	D	I	I	I	I	NA	NA	S	NA	NA	NA
swValueBlockSize		x	x	D	I	I	I	I	NA	NA	S	NA	NA	NA
swValueBlockSizeMult		x	x	D	I	I	I	I	NA	NA	S	NA	NA	NA
unit		x	x	D	I	I	I	NA	I	NA	S	D	NA	NA
valueAxisDataType		x	x	D	I	I	I	I	NA	NA	S	NA	NA	NA

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋()

The following settings apply in [constr_1015]:

D **Define** the attribute independent of settings to the left.

R Use or **re-define** definition from the left in the scope of this element.

A **Add** attribute if not defined on the left, or as additional information.

If the attribute has an upper multiplicity > 1 and the attribute is defined on the left then the attribute is added to the attribute defined on the left.

If the attribute has an upper multiplicity of 1 and the attribute is not defined on the left then the attribute is defined.

If the attribute has an upper multiplicity of 1 and the attribute is already defined on the left then the attribute is not redefined but this is considered as invalid configuration.

[15] swIntendedResolution is used only in an early phase of the definition of data types, namely in the context of the definition of so-called blueprints. To that extent, swIntendedResolution represents a non-binding requirement that shall later be considered for the definition of an appropriate CompuMethod.

I **Inherit** the definition from the left for usage in the scope of this element.

NA Attribute is **not applicable** for usage in the scope of this element.

M Attribute is **meaningless** in the scope of this element. As it was allowed in previous versions, declaring it as Not Applicable (NA) would break compatibility. Tools shall ignore such an attribute without a warning.

C This means that the left element constrains right element.

AI If the attribute is already defined on the left then the attribute is not redefined but adds implementation-related information.

Example: an ApplicationDataType of category BOOLEAN supports the definition of an own CompuMethod to define the semantics of e.g. (ON, OFF) or (HIGH, LOW) or (PASSED, FAILED) **as long as the number of values match and matching pairs of values on application level and implementation level exist**. In contrast, the corresponding ImplementationDataType uses (true, false) as the applicable literals in any of the above mentioned cases.

S Create a “Self-contained” artifact based on the left.

Example: A CompuMethod defined in the context of a System of category ECU_EXTRACT is copied into the separate artifact for the McSupportData and references need to be updated to the copy.

Use case: Provide a McDataGenerator with a single, self-contained file to do its job.

Some property names contain the term “variable” or “calprm”, this comes from historical[16] reasons and can be taken as some hint where the property most likely applies to.

The usage of the "/" in the table rows mentioning the content of swCalprmAxisSet.swCalprmAxis, in particular SwAxisGrouped.swCalprmRef and SwAxisGrouped.sharedAxisType resp. SwAxisIndividual.swVariableRef, SwAxisIndividual.unit, and SwAxisIndividual.inputVariableType represents a "shortcut" that glosses over a specific aspect of the modeling of SwCalprmAxis that is visible in the model but does not appear in the AUTOSAR XML schema, see Figure 5.27 (which contains all the meta-classes and roles mentioned in the table entries).

In particular, the "shortcut" affects the existence of meta-class SwCalprmAxisTypeProps and its aggregation in the role SwCalprmAxis.swCalprmAxisTypeProps. As depicted in Figure 5.27, SwCalprmAxisTypeProps acts as an abstract base class to both SwAxisGrouped and SwAxisIndividual.

In ARXML files that conform to the AUTOSAR XML schema, however, both SwAxisGrouped and SwAxisIndividual appear as direct child elements of SwCalprmAxis.

[16]In the beginning of ASAM and MSR, measurements and calibration parameters (characteristics) were separated and the properties were merged over time.

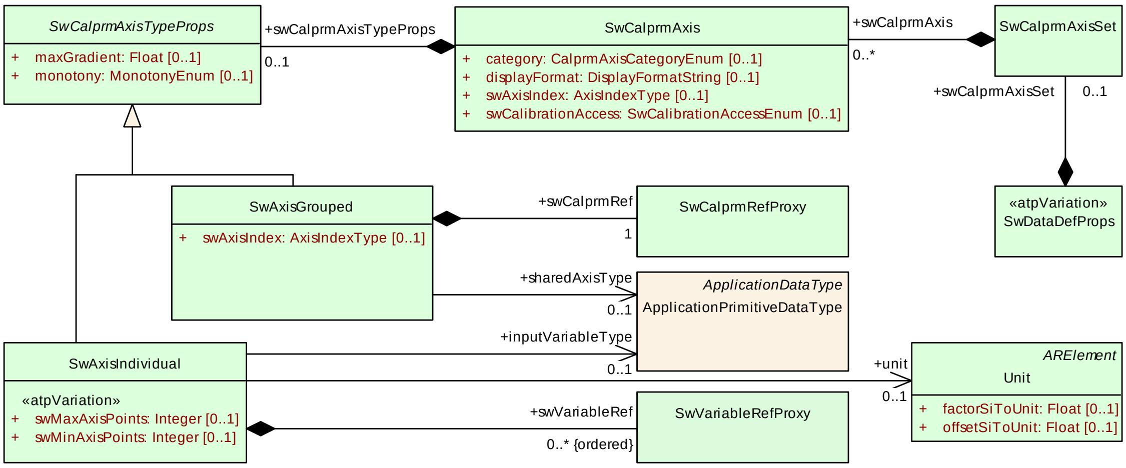


Figure 5.27: Modeling of SwAxisGrouped and SwAxisIndividual

This difference between meta-model and AUTOSAR XML schema is explained by the existence of a set of tags at the aggregation SwCalprmAxis.swCalprmAxisTypeProps. The details of how these tags impact the schema generation are explained in the *TPS XML Schema Production Rules* [27].

To summarize, the "shortcut" in the table rows simply approximates the situation in ARXML instead of reflecting the actual modeling in the AUTOSAR meta-model.

Class	<<atpVariation>> **SwDataDefProps**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:**vh.latestBindingTime=codeGenerationTime
Base	*ARObject*

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
Aggregated by	*AutosarDataType*.swDataDefProps, CompositeNetworkRepresentation.networkRepresentation, *Data Prototype*.swDataDefProps, DataPrototypeTransformationProps.networkRepresentationProps, DiagnosticDataElement.swDataDefProps, DiagnosticEnvDataElementCondition.swDataDefProps, Dlt Argument.networkRepresentation, FlatInstanceDescriptor.swDataDefProps, ImplementationDataType Element.swDataDefProps, InstantiationDataDefProps.swDataDefProps, ISignal.networkRepresentation Props, McDataInstance.resultingProperties, ParameterAccess.swDataDefProps, PerInstanceMemory.sw DataDefProps, *ReceiverComSpec*.networkRepresentation, *SenderComSpec*.networkRepresentation, SomeipDataPrototypeTransformationProps.networkRepresentation, SwPointerTargetProps.swDataDef Props, SwServiceArg.swDataDefProps, SwSystemconst.swDataDefProps, SystemSignal.physicalProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:**xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:**xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:**xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:**xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:**xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:**xml.sequenceOffset=215

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:**xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:**xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod. **Tags:**xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:**xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:**xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:**xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:**xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:**xml.sequenceOffset=230

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:**xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:**xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:**xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:**xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:**xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:**xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:**xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swValueBlockSizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValueBlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:**xml.sequenceOffset=350
valueAxisDataType	ApplicationPrimitiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:**xml.sequenceOffset=355

Table 5.40: SwDataDefProps

Primitive	**NativeDeclarationString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This string contains a native data declaration of a data type in a programming language. It is basically a string, but white-space shall be preserved. **Tags:** xml.xsd.customType=NATIVE-DECLARATION-STRING xml.xsd.type=string xml.xsd.whiteSpace=preserve

Table 5.41: NativeDeclarationString

Class	**SwBitRepresentation**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	Description of the structure of a bit variable: Comprises of the bitPosition in a memory object (e.g. swHostVariable, which stands parallel to swBitRepresentation) and the numberOfBits . In this way, interrelated memory areas can be described. Non-related memory areas are not supported.			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swBitRepresentation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bitPosition	Integer	0..1	attr	If the "bit data object" is hosted within another data object (e.g. if the memory can be accessed via byte as well as bit address), this attribute specifies the position of the data object. The count starts at zero (0). **Tags:**xml.sequenceOffset=20

▽

△

Class	SwBitRepresentation			
numberOfBits	Integer	0..1	attr	Number of bits allocated by a "bit data object" within its host data object. **Tags:**xml.sequenceOffset=30

Table 5.42: SwBitRepresentation

Primitive	**DisplayFormatString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This is a display format specifier for the display of values e.g. in documents or in measurement and calibration systems. The display format specifier is a subset of the ANSI C printf specifiers with the following form: % [flags] [width] [.prec] type character Due to the numerical nature of value settings, only the following type characters are allowed: • d: Signed decimal integer • i: Signed decimal integer • o: Unsigned octal integer • u: Unsigned decimal integer • x: Unsigned hexadecimal integer, using "abcdef" • X: Unsigned hexadecimal integer, using "ABCDEF" • e: Signed value having the form [-]d.dddd e [sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or - • E: Identical to the e format except that E rather than e introduces the exponent • f: Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits; the number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision • g: Signed value printed in f or e format, whichever is more compact for the given value and precision; trailing zeros are truncated, and the decimal point appears only if one or more digits follow it • G: Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate) • b: binary number without prefix or suffix, e.g. 110 • B: binary number with suffix capital B, e.g. 110B **Tags:** xml.xsd.customType=DISPLAY-FORMAT-STRING xml.xsd.pattern=%[\-+#]?[0-9]*(\.[0-9]+)?[bBdiouxXfeEgGcs] xml.xsd.type=string

Table 5.43: DisplayFormatString

Class	**Annotation**			
Package	M2::MSR::Documentation::Annotation			
Note	This is a plain annotation which does not have further formal data.			
Base	*ARObject*, *GeneralAnnotation*			
Aggregated by	*EcucAbstractReferenceValue*.annotation, *EcucParameterValue*.annotation, HwAttributeValue.annotation, *Identifiable*.annotation, PostBuildVariantCriterionValue.annotation, SwDataDefProps.annotation, SwSystemconstValue.annotation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 5.44: Annotation

[constr_1244] DataPrototypes used in application software shall not be typed by C enums ⌈A ImplementationDataType that is used to type a DataPrototype owned by an AtomicSwComponentType shall not set swDataDefProps.additionalNativeTypeQualifier to enum **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01272] Semantics of swComparisonVariable ⌈Please note that swComparisonVariables shall be displayed in the MCD system on the ordinate in a curve.

By showing the input value and the comparison value the calibration engineer can see if the current working point is above or below a curve provident thresholds. For example, in a curve specifying a temperature-depending gear shift threshold engine speed the engine speed can be shown as "comparisonVariable".

These variables can be used to display the value of a variable on the value axis of a calibration parameter (characteristic), that is currently displayed in the MCD-System.

The purpose is to compare the appropriate result from the calibration parameter in question, with a value being calculated or taken from a sensor (the comparison variable).

The sole purpose of this comparison-variable is therefore to serve the calibration process.⌋*()*

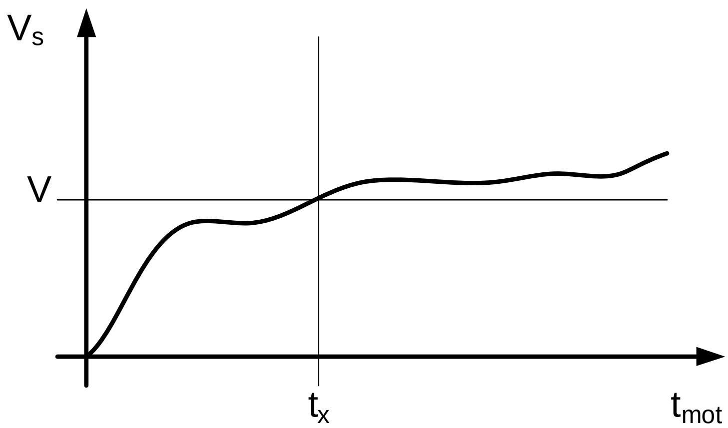


Figure 5.28: Explanation of swComparisonVariable

The meaning behind swComparisonVariable is depicted in Figure 5.28. Legend: t_x represents the current temperature and t_{mot} represents the motor temperature.

V represents the current speed as shown in the MCD system for comparison: this is the swComparisonVariable.

Likewise, V_s represents the speed characteristic over the temperature.

Enumeration	**SwCalibrationAccessEnum**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Aggregated by	ModeDeclarationGroupPrototype.swCalibrationAccess, SwCalprmAxis.swCalibrationAccess, SwDataDefProps.swCalibrationAccess
Literal	*Description*
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file. **Tags:**atp.EnumerationLiteralIndex=0
readOnly	The element will only appear as read-only in an ASAP file. **Tags:**atp.EnumerationLiteralIndex=1
readWrite	The element will appear in the ASAP file with both read and write access. **Tags:**atp.EnumerationLiteralIndex=2

Table 5.45: SwCalibrationAccessEnum

[TPS_SWCT_01273] Precedence rules for the application of SwDataDefProps ⌈ SwDataDefProps can be specified on various levels, from type over prototype to instantiation, finally data access and calibration support after RTE generation. In general, properties specified on prototype level override the ones specified on type level.

More formally, the precedence of such properties is:

1. attributes of SwDataDefProps defined on ApplicationDataType which may be overwritten by
2. attributes of SwDataDefProps defined on ImplementationDataType which may be overwritten by
3. attributes of SwDataDefProps defined on DataPrototype which may be overwritten by
4. attributes of SwDataDefProps defined on InstantiationDataDefProps which may be overwritten by
5. attributes of SwDataDefProps defined on ParameterAccess respectively Argument which may be overwritten by
6. attributes of SwDataDefProps defined on FlatInstanceDescriptor which may be overwritten by
7. attributes of SwDataDefProps defined on McDataInstance

⌋*()*

Note that details about applicable attributes of SwDataDefProps can be found in [constr_1015].

[TPS_SWCT_01274] SwDataDefProps used to support calibration and measurement ⌈The last item in the list of use cases contained in [TPS_SWCT_01273] denotes that SwDataDefProps are also used as part of McSupportData which is a direct input to the generation of measurement and calibration configuration formats (so-called A2L-files).

This use case is further explained in [6]. Since these data are generated by the RTE, they will use a copy of the properties according to the precedence given above.

However, even in this use case which comes after RTE generation it is possible that properties relevant for the MCD system are added which had been undefined so far.

This for example, applies to the attribute swRefreshTiming which denotes a timing information relevant for the measurement system; this information may be set rather late in the process chain.⌋*()*

Obviously such an override is not applicable in all cases. In particular, the properties covering the structure shall not be redefined on DataPrototype. Implementation policy, semantics and code generation policy may be changed under consideration of compatibility rules.

Access policy for the MCD system is the most likely subject to be redefined on the DataPrototype of even on an instantiation level.

Section 5.4.3 describes how SwDataDefProps are used for measuring purposes while Section 5.4.4 describes the construction of characteristics based on the combination of SwDataDefProps with DataPrototypes.

Section 2.2.2 describes in which context calibration parameters can be defined. Finally, sections 2.2.3, 7.5.4, and 5.5.4 show how calibration parameters are used in RunnableEntitys and show the link to an actual ECU implementation.

Enumeration	**SwImplPolicyEnum**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	Specifies the implementation strategy with respect to consistency mechanisms of variables.
Aggregated by	BswInternalTriggeringPoint.swImplPolicy, InternalTriggeringPoint.swImplPolicy, SwDataDefProps.swImplPolicy, Trigger.swImplPolicy
Literal	***Description***
const	forced implementation such that the running software within the ECU shall not modify it. For example implemented with the "const" modifier in C. This can be applied for parameters (not for those in NVRAM) as well as argument data prototypes. **Tags:**atp.EnumerationLiteralIndex=0
fixed	This data element is fixed. In particular this indicates, that it might also be implemented e.g. as in place data, (#DEFINE). **Tags:**atp.EnumerationLiteralIndex=1
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 measurementPoint data elements. **Tags:**atp.EnumerationLiteralIndex=2
queued	The content of the data element is queued and the data element has 'event' semantics, i.e. data elements are stored in a queue and all data elements are processed in 'first in first out' order. The queuing is intended to be implemented by RTE Generator. This value is not applicable for parameters. **Tags:**atp.EnumerationLiteralIndex=3
standard	This is applicable for all kinds of data elements. For variable data prototypes the 'last is best' semantics applies. For parameter there is no specific implementation directive. **Tags:**atp.EnumerationLiteralIndex=4

Table 5.46: SwImplPolicyEnum

[constr_10099] Allowed values of the attribute SwDataDefProps.swImplPolicy vs. DataPrototypes and their roles ⌈

Attribute of SwImplPolicyEnum	VariableDataPrototype							ParameterDataPrototype					Misc.
	VariableDataPrototype in SenderReceiverInterface	VariableDataPrototype in NvDataInterface	VariableDataPrototype in role ramBlock	VariableDataPrototype in role implicitInterRunnableVariable	VariableDataPrototype in role explicitInterRunnableVariable	VariableDataPrototype in role arTypedPerInstanceMemory	VariableDataPrototype in role staticMemory	ParameterDataPrototype in ParameterInterface	ParameterDataPrototype in role romBlock	ParameterDataPrototype in role sharedParameter	ParameterDataPrototype in role perInstanceParameter	ParameterDataPrototype in role constantMemory	ArgumentDataPrototype
const	NA	NA	NA	NA	NA	NA	NA	x	NA	x	x	x	NA
fixed	NA	NA	NA	NA	NA	NA	NA	x	NA	NA	NA	x	NA
measurementPoint	x	NA	NA	NA	NA	x	x	NA	NA	NA	NA	NA	NA
queued	x	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
standard	x	x	x	x	x	x	x	x	x	x	x	x	x

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

The restrictions implemented in [constr_10099] reflect the fact that not all possible implementation strategies are useful or supported for all kinds of DataPrototypes.

Please note that the usage of swImplPolicy is further constraint in the combination with the attribute value swCalibrationAccess as described in [constr_1017].

The following settings apply to the table contained in [constr_10099]:

x Attribute is applicable for usage in the scope of this element.

NA Attribute is **not** applicable for usage in the scope of this element.

[constr_2035] swImplPolicy for VariableDataPrototype in SenderReceiverInterface ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype owned by a SenderReceiverInterface shall be either standard, queued, or measurementPoint.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_2036] swImplPolicy for VariableDataPrototype in NvDataInterface ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype owned by a NvDataInterface shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2037] swImplPolicy for VariableDataPrototype in the role ramBlock ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype aggregated in the role NvBlockDescriptor.ramBlock shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2038] swImplPolicy for VariableDataPrototype in the role implicitInterRunnableVariable ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype aggregated in the role SwcInternalBehavior.implicitInterRunnableVariable shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2039] swImplPolicy for VariableDataPrototype in the role explicitInterRunnableVariable ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype aggregated in the role SwcInternalBehavior.explicitInterRunnableVariable shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2040] swImplPolicy for VariableDataPrototype in the role arTypedPerInstanceMemory ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype aggregated in the role SwcInternalBehavior.arTypedPerInstanceMemory shall be standard or measurementPoint.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2041] swImplPolicy for VariableDataPrototype in the role staticMemory ⌈The overriding value of attribute swImplPolicy of a VariableDataPrototype aggregated in the role InternalBehavior.staticMemory shall be standard or measurementPoint.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2042] swImplPolicy for ParameterDataPrototype in ParameterInterface ⌈The overriding value of attribute swImplPolicy of a ParameterDataPrototype owned by a ParameterInterface shall be either standard, const, or fixed.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_2043] swImplPolicy for ParameterDataPrototype in the role romBlock ⌈The overriding value of attribute swImplPolicy a ParameterDataPrototype aggregated in the role NvBlockDescriptor.romBlock shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋*()*

[constr_2044] swImplPolicy for ParameterDataPrototype in the role sharedParameter ⌈The overriding value of attribute swImplPolicy of a ParameterDataPrototype aggregated in the role SwcInternalBehavior.sharedParameter shall be standard or const.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋*()*

[constr_2045] swImplPolicy for ParameterDataPrototype in the role perInstanceParameter ⌈The overriding value of attribute swImplPolicy of a ParameterDataPrototype in the role SwcInternalBehavior.perInstanceParameter shall be standard or const.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋*()*

[constr_2046] swImplPolicy for ParameterDataPrototype in the role constantMemory ⌈The overriding value of attribute swImplPolicy of a ParameterDataPrototype aggregated in the role InternalBehavior.constantMemory shall be standard, const, or fixed.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋*()*

[constr_2047] swImplPolicy for ArgumentDataPrototype ⌈The overriding value of attribute swImplPolicy of an ArgumentDataPrototype shall be standard.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋*()*

[TPS_SWCT_02000] Default value for attribute swImplPolicy ⌈If the attribute swImplPolicy is not explicitly set at any of the locations listed in "Place of Setting" for SwDataDefProps, the default value standard applies.⌋*()*

Please note that the locations listed in "Place of Setting" for SwDataDefProps are described in [constr_1015].

5.4.2 Invalid Value

The diagram 5.5 shows that in addition to the semantics defined through the compuMethod (explained below in chapter 5.5.1), also an invalidValue can be specified. This is a requirement of the VFB [3], allowing expressing which specific value is used to indicate invalidation.

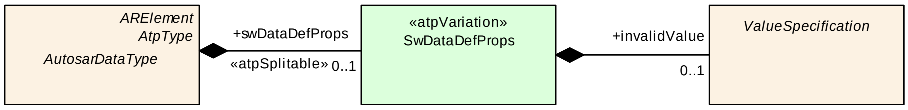


Figure 5.29: Invalid value

The relation of the definition of an invalidValue to a CompuMethod (see section 5.5.1) is illustrated in Figures 5.30, 5.31, and 5.32.

Figure 5.30 depicts the possibility that the invalidValue can be defined (with different consequences, see [TPS_SWCT_01834] and [TPS_SWCT_01835]) **inside or outside** of the boundaries of the corresponding CompuMethod.

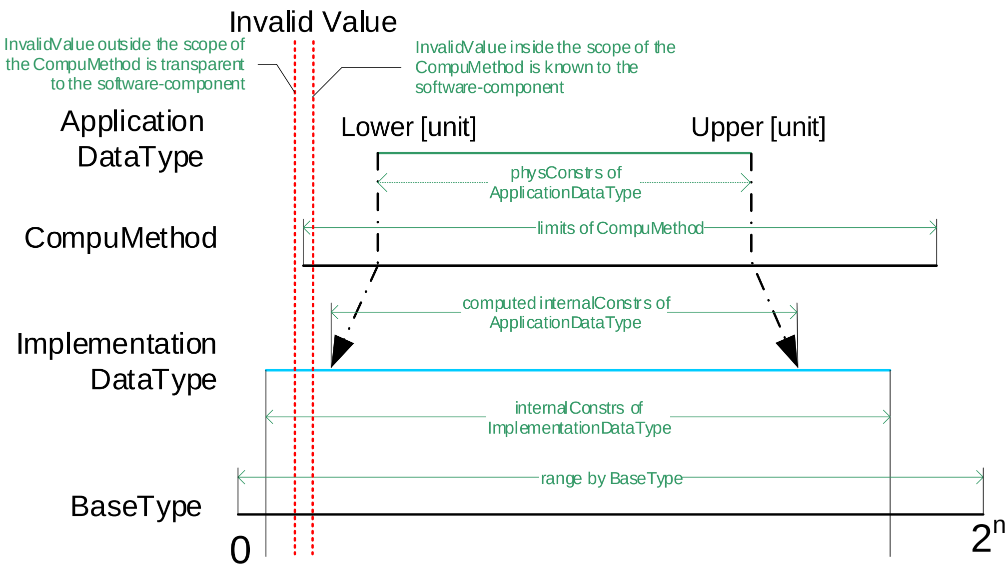


Figure 5.30: Value ranges and invalid values for linear and rational CompuMethod

Figure 5.31 and Figure 5.32 depict a similar situation for the case of mixed CompuMethods where the invalidValue is defined in the discrete part of a CompuMethod.

Figure 5.31 sketches a case where a CompuMethod has a linear and a discrete part and the invalidValue is defined by means of one value that is defined in the discrete part of the CompuMethod.

As mentioned by [TPS_SWCT_01834], the invalidValue shall be defined in the physical domain in this case. In other words, the invalidValue shall be defined by a symbol according to [TPS_SWCT_01432].

As a consequence of the definition of an invalidValue **inside**, the scope of a mixed CompuMethod the invalidValue is visible to the software-component.

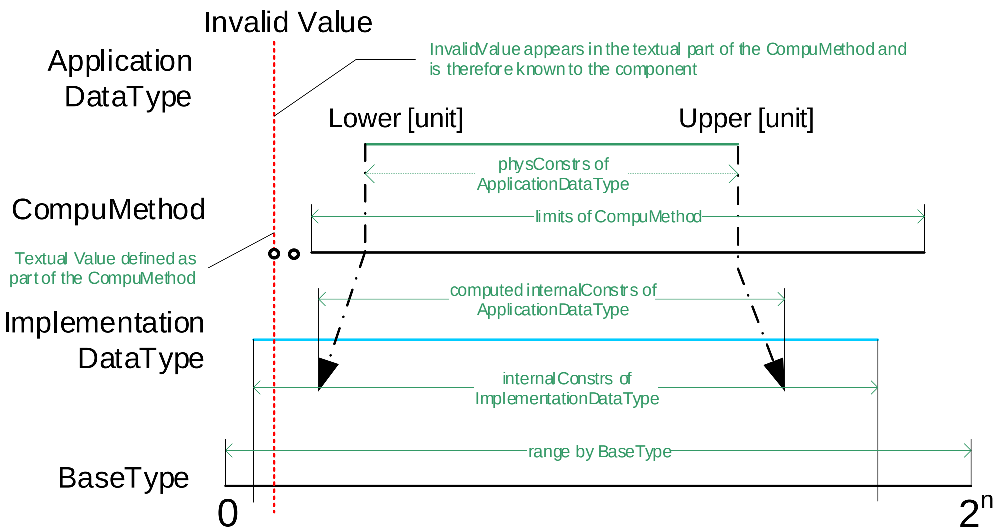


Figure 5.31: Value ranges and invalid values with discrete invalidValue defined inside the scope of the CompuMethod

Figure 5.32, on the other hand, sketches a case where a CompuMethod has a linear and a discrete part and the invalidValue is not within the defined linear interval and not defined by means of one value out of the discrete part of the CompuMethod.

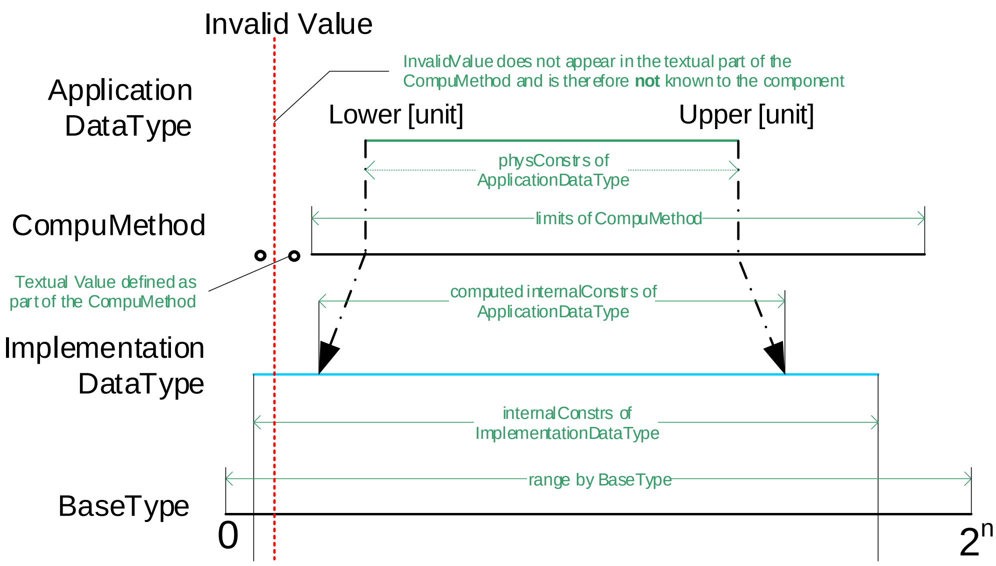


Figure 5.32: Value ranges and invalid values with discrete invalidValue defined outside the scope of the CompuMethod

As mentioned by [TPS_SWCT_01835], the invalidValue shall be defined in the internal domain in this case. In other words, the invalidValue shall be defined by a NumericalValueSpecification.

As a consequence of the definition of an invalidValue **outside** the scope of a mixed CompuMethod, the invalidValue is invisible (and therefore not accessible) to the software-component.

The conclusions of the illustrations in Figures 5.30, 5.31, and 5.32 are summarized in the following list:

- **[TPS_SWCT_01432] Keep the invalidValue transparent to the sending and receiving software components** ⌈On the one hand it is possible to keep the invalidValue transparent to the sending and receiving software components. In this case the invalidation API of the RTE on the sender side has to be used.

  The receiving software component can either use the data receive status or the DataReceiveErrorEvent respectively DataReceivedEvent to decide about the validity of the received data or the receiving software component can rely on the reception of an initValue as a default value in case of data invalidation.

  In this case the invalid value should (and usually will) be **outside the range limits** defined by the compuMethod.⌋*()*

  [TPS_SWCT_01835] invalidValue is outside the scope of the compuMethod ⌈If the value of the invalidValue of an ApplicationPrimitiveDataType of category VALUE is supposed to be **outside** the scope of the applicable CompuMethod, a NumericalValueSpecification (that provides a value in the internal representation) shall be used to describe the invalidValue of the ApplicationPrimitiveDataType.⌋*(RS_SWCT_03216)*

  Because of the existence of [TPS_SWCT_01834] and [TPS_SWCT_01835], the definition of the invalidValue is fully specified and therefore [constr_1221] does not apply to this case.

- **[TPS_SWCT_01434] Sender and receiver have knowledge of invalid value** ⌈On the other hand it is possible that the communicating software components do have knowledge about the invalidValue and the invalidValue is visible for them.

  This is in particular the case if the sender and receiver are calculating a checksum over a larger data structure to implement an end to end communication protection. To ensure the integrity of the checksums it is required to set invalid values by the sending component directly and to receive invalid values unchanged.

  In this case the invalid value should (and usually will) be **inside the range limits** defined by the compuMethod.⌋*()*

  [TPS_SWCT_01834] invalidValue is inside the scope of the compuMethod ⌈If the value of the invalidValue of an ApplicationPrimitiveDataType of category VALUE is supposed to be **inside** the scope of the applicable CompuMethod, an ApplicationValueSpecification shall be used to describe the invalidValue of the ApplicationPrimitiveDataType.⌋*(RS_SWCT_03216)*

[TPS_SWCT_01834] means that the value of the ApplicationValueSpecification shall be within the bounds defined by swDataDefProps.compuMethod.compuPhysToInternal.compuContent.compuScale.lowerLimit or upperLimit or the inverse case that is based on the bounds defined by swDataDefProps.compuMethod.compuInternalToPhys.compuContent.compuScale.lowerLimit or upperLimit.

[TPS_SWCT_01436] Different receivers require different handling of data invalidation ⌈It is possible that in case of 1:n communication different receivers requiring a different handling of data invalidation depending on the criticality of its functionality. For instance, one receiver applies the checksum based end to end communication protection and another receiver relies on the substitution of invalid values by invalidValues.⌋*()*

The handling of invalidValue for ApplicationPrimitiveDataType of category STRING is defined by [constr_1242].

A typical use case for putting the invalidValue inside the boundaries of the applicable CompuMethod is a composite data type that contains the values of all individual wheel speeds. If one of the sensors fails and starts to send invalidValue it would probably not make sense to consider the whole composite data element invalid.

It may very likely still be possible to make sense of the remaining intact wheel speed values and carry on with whatever business the receiving software-component has with that data.

From this perspective, it would obviously be OK for the sending software-component to actively send the invalidValue that is then processed as a “regular” value without applying additional semantics by the RTE/Com.

[TPS_SWCT_01646] Sending invalidValue without invalidation applied by RTE/Com ⌈For intentionally sending invalidValue without invalidation applied by RTE/Com the SenderReceiverInterface.invalidationPolicy.handleInvalid shall be set to the value HandleInvalidEnum.dontInvalidate.⌋*()*

[constr_1390] Restriction to the value of SenderReceiverInterface.invalidationPolicy.handleInvalid ⌈If the value of SenderReceiverInterface.invalidationPolicy.handleInvalid is set to any value other than HandleInvalidEnum.dontInvalidate then the invalidValue shall not be within the interval defined by the CompuMethod of the applicable dataElement **at the time when the contract phase generation is executed**.⌋*()*

Please note that ApplicationPrimitiveDataTypes of category VALUE in principle can have an invalidValue provided by a NumericalValueSpecification because the value of the attribute invalidValue can be **outside the range** of the applicable CompuMethod (see [TPS_SWCT_01432]).

[TPS_SWCT_01437] invalidValue can also be specified without setting a compuMethod ⌈An invalidValue can also be specified without setting a compuMethod.⌋*()*

Figure 5.30 illustrates the relationship between ApplicationDataType, CompuMethod, ImplementationDataType, invalidValue, BaseType.

[constr_2545] invalidValue shall fit in the specified ranges ⌈The invalidValue shall be in the range of the ImplementationDataType **at the time when the contract phase generation is executed**.⌋*()*

Please note that the invalidValue is a ValueSpecification. Of course, it would technically be possible to use any subclass of ValueSpecification at this place.

[constr_1016] Restriction of invalidValue for ImplementationDataType and ImplementationDataTypeElement ⌈invalidValue for ImplementationDataType and ImplementationDataTypeElement is restricted to be either a compatible NumericalValueSpecification, TextValueSpecification (caution, [constr_1284] applies) or a ConstantReference that in turn points to a compatible ValueSpecification.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1384] Definition of invalidValue for DataPrototype typed by ApplicationPrimitiveDataType of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK ⌈An invalidValue **shall not** be specified for a DataPrototype typed by ApplicationPrimitiveDataType of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK **at any time in the workflow**.⌋*()*

Rationale for [constr_1384]: there is no use case for sending a DataPrototype typed by ApplicationPrimitiveDataType of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK over a communication bus.

[TPS_SWCT_01486] ApplicationPrimitiveDataType of category STRING may have invalidValue ⌈The only kind of Compound Primitive Data Type that is allowed to define an invalidValue is an ApplicationPrimitiveDataType of category STRING.⌋*(RS_SWCT_03216)*

[constr_1241] Compound Primitive Data Types and invalidValue ⌈Compound Primitive Data Types that have set the value of category other than STRING shall **not** define invalidValue **at the time when the contract phase generation is executed**.⌋*()*

[constr_1242] Restriction of invalidValue for ApplicationPrimitiveDataType of category STRING ⌈invalidValue for ApplicationPrimitiveDataType of category STRING ([constr_1241] applies) is restricted to be either a compatible ApplicationValueSpecification or a ConstantReference that in turn points to a compatible ApplicationValueSpecification **at the time when the contract phase generation is executed**.⌋()

[TPS_SWCT_01487] Correspondence of invalidValue for ApplicationPrimitiveDataType and ImplementationDataType ⌈The invalidValue specified on the level of an ApplicationPrimitiveDataType shall correspond to the invalidValue specified on the level of a compatible ImplementationDataType. The terms “corresponds” boils down to:

- category VALUE or BOOLEAN: application of CompuMethod
- category STRING: the definition of a constant value for a string data type is explained in section "Specification of values for Strings". There is no formal support defined to check that the values of invalidValue really correspond to each other.

⌋()

[constr_10196] Definition of invalidValue for DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE ⌈If an invalidValue is defined for a DataPrototype that is typed by an ImplementationDataType that references or inherits (see [constr_1015]) a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE, the applicable ValueSpecification shall be a TextValueSpecification **if** the value fits into the intervals defined by the CompuMethod.

In this case the value provided shall match to one of the applicable text values (vt, shortLabel, symbol) defined by the applicable CompuScales.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Please note that several attributes of meta-class CompuScale can be taken to describe the actual value. It is therefore necessary to clarify what happens if several of these attributes exist within the context of one CompuScale. This clarification can be found in [TPS_SWCT_01696].

The consequence of the existence of [constr_10196] is that a NumericalValueSpecification can be used for the definition of an invalidValue for a DataPrototype that is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE if the invalidValue is **not** supposed to be represented by the CompuMethod.

[constr_1302] Restriction of data invalidation ⌈Data invalidation is only applicable for one of the following cases applicable on the **receiving** side:

1. VariableDataPrototypes typed by either an ApplicationPrimitiveDataType or an ImplementationDataType of category VALUE or TYPE_REFERENCE that boils down to category VALUE that have defined an invalidValue.

2. VariableDataPrototypes typed by either an ApplicationComposite-DataType or an ImplementationDataType of category STRUCTURE, or ARRAY or of category TYPE_REFERENCE that boils down to category STRUCTURE, or ARRAY that have **at least one** primitive element with an invalidValue.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Please note that [constr_1302], in general, leaves room for the definition of an invalid value for a DataPrototype typed by a Wrapped Union Data Type because it demands the existence of a primitive element that has an invalidValue. In the case of a Wrapped Union Data Type, the primitive element could be the Member Selector, and thus [constr_1302] would technically be fulfilled.

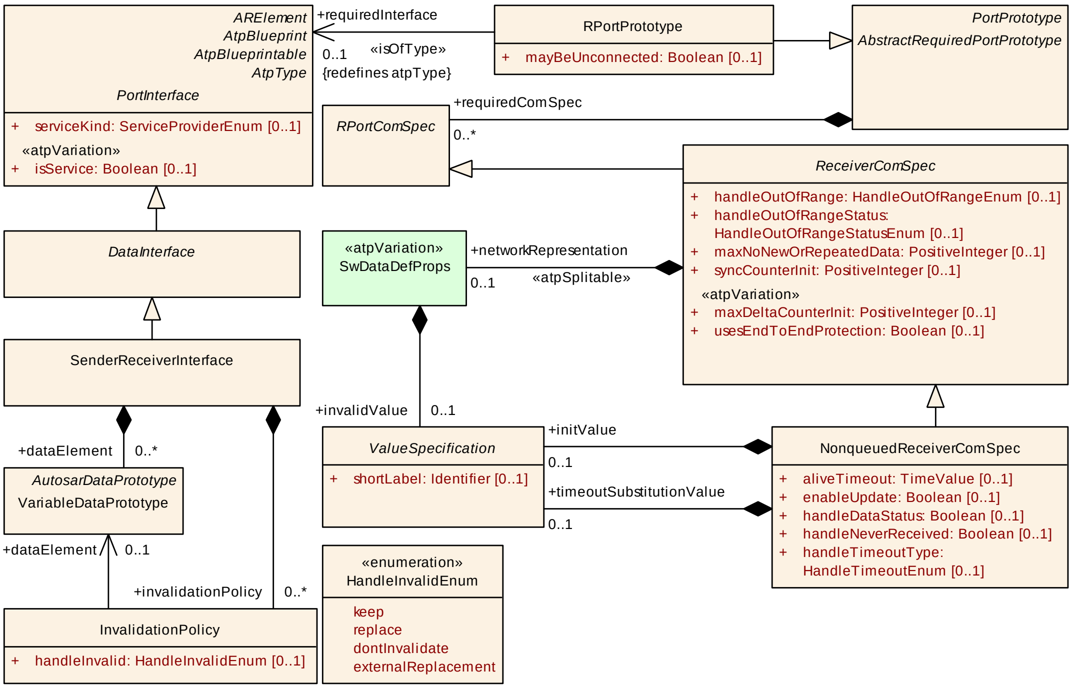


Figure 5.33: Relationships required to consider the invalidValue

On the one hand, it does not make sense to just define an invalid value for the Member Selector from the semantic point of view. On the other hand, the actual payload may not even have an invalid value according to [constr_1009] or [constr_1288], respectively.

In order to simplify the situation and make a clear statement, [constr_1446] has been defined.

[constr_1446] No definition of invalidValue for a Wrapped Union Data Type
⌈The definition of an invalidValue for a DataPrototype typed by a Wrapped Union Data Type is not supported **at any time in the workflow**.⌋()

[constr_1140] Combination of invalidValue with the attribute handleInvalid
⌈The combination of setting the attribute handleInvalid of the meta-class InvalidationPolicy owned by SenderReceiverInterface to value replace **and** of

setting the value of the attribute initValue owned by a corresponding NonqueuedReceiverComSpec effectively to the value of the invalidValue (owned by a corresponding SwDataDefProps) is not supported **at the time when the contract phase generation is executed**.⌋()

The term "corresponding" (as utilized in [constr_1140]) refers to the fact that information regarding the fulfillment of [constr_1140] is factually distributed over different areas of the meta-model. For clarification, the following relationship should be considered:

The SenderReceiverInterface defines how to deal with an invalid value by means of the attribute handleInvalid on the basis of individual dataElements. The SenderReceiverInterface is taken for typing a RPortPrototype that in turn owns a ReceiverComSpec. [constr_1140] applies if the particular ReceiverComSpec is actually a NonqueuedReceiverComSpec that refers to the same dataElement.

In this case the invalidValue owned by the SwDataDefProps that in turn is owned by the respective dataElement is relevant for the fulfillment of [constr_1140]. The "big picture" of this relationship is sketched in Figure 5.33.

[constr_1219] Invalidation depends on the value of swImplPolicy ⌈If the value of swImplPolicy of a SenderReceiverInterface.dataElement is set to the value SwImplPolicyEnum.queued, then the enclosing SenderReceiverInterface shall not aggregate in the role invalidationPolicy an InvalidationPolicy

- that references the dataElement and
- where the value of InvalidationPolicy.handleInvalid is set to anything else than HandleInvalidEnum.dontInvalidate

at the time when the contract phase generation is executed.⌋()

[constr_1282] Restriction concerning the usage of RuleBasedValueSpecification or a ReferenceValueSpecification for the specification of an invalidValue ⌈The aggregation of a RuleBasedValueSpecification or a ReferenceValueSpecification for the definition of a ApplicationPrimitiveDataType.swDataDefProps.invalidValue is not supported **at the time when the contract phase generation is executed**.⌋()

5.4.3 Properties for Measurement

In embedded automotive software design, measurement means access to memory locations in an ECU and transferring its contents to the measurement & calibration system.

While in classical software design, variables abstract the memory locations in the code, AUTOSAR provides for this purpose the DataPrototype with its various specializations:

- VariableDataPrototype of a SenderReceiverInterface or NvDataInterface used in a PortPrototype (of a SwComponentPrototype), to capture sender-receiver and non-volatile data communication between SwComponentPrototypes
- ArgumentDataPrototype of a ClientServerOperation in a ClientServerInterface to capture client-server communication between SwComponentPrototypes.
- VariableDataPrototype in the context of an SwcInternalBehavior to
  - capture communication between RunnableEntitys within a SwComponentPrototype
  - handle data in a non-volatile memory block
  - provide pure software component internal memory which has to be accessible for an MCD system

[TPS_SWCT_01440] Measurement is not limited to primitive objects ⌈The ability of being measured is not restricted to primitive data (category VALUE) but can also be applied to composite data (category STRUCTURE or ARRAY).⌋*()*

The following semantical and structural features from SwDataDefProps are relevant (among other purposes) for the measurement system:

- swCalibrationAccess
- swImplPolicy
- compuMethod
- unit (if not specified by compuMethod)
- baseType

[TPS_SWCT_01130] Measurement and calibration access to model elements is defined by swCalibrationAccess ⌈The ability to be accessed by e.g. a calibration tool is given by setting the swCalibrationAccess attribute.⌋*(RS_SWCT_03152)*

The valid settings of swCalibrationAccess are regulated by [constr_1017].

[constr_1017] Supported combinations of swImplPolicy and swCalibrationAccess ⌈

swImplPolicy	swCalibrationAccess		
	notAccessible	readOnly	readWrite
fixed	**yes**	not supported	not supported
const	**yes**	**yes**	not supported
standard	**yes**	**yes**	**yes**

▽

△

queued	**yes**	not supported	not supported
measurementPoint	not supported	**yes**	not supported

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋()

[TPS_SWCT_01559] Default value for attribute SwDataDefProps.swCalibrationAccess ⌈The default value for the attribute SwDataDefProps.swCalibrationAccess is SwCalibrationAccessEnum.notAccessible.⌋()

[constr_1018] dataElement with swImplPolicy set to measurementPoint shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess ⌈Due to the nature of dataElements characterized by setting the swImplPolicy to measurementPoint, such dataElements shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess **at the time when the contract phase generation is executed**.⌋()

5.4.4 Properties of Curves and Maps

A characteristic table is defined by setting the category of the corresponding AutosarDataType or DataPrototype to CURVE respectively MAP, CUBOID, CUBE_4, and CUBE_5.

Its SwDataDefProps determine an axis description. The type of the functional values is given by the attached SwBaseType and the CompuMethod.

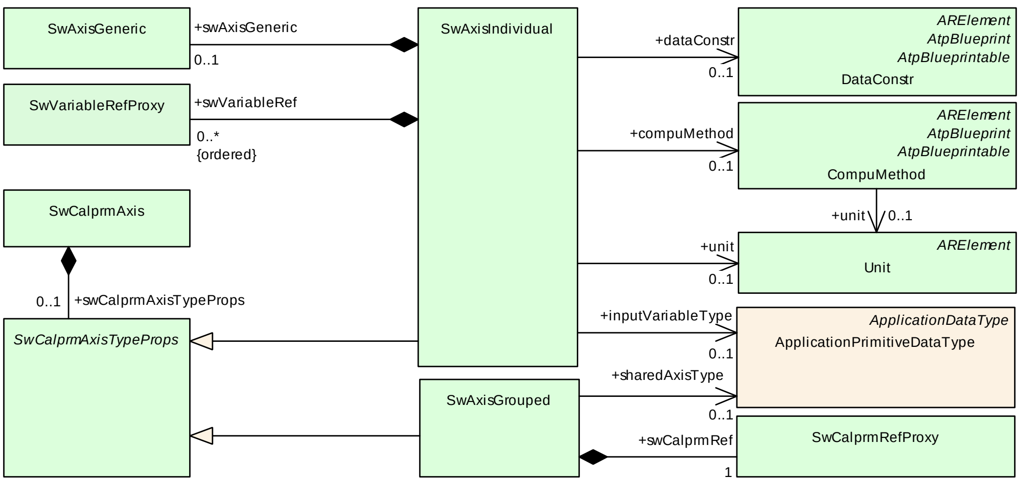


Figure 5.34: Overview on the Meta-Model for Axis Description

The axis description itself is defined by the meta-model element SwCalprmAxisSet aggregating the appropriate number of SwCalprmAxisTypeProps.

This is the base class for a so called “individual axis” (formalized by meta-class SwAxisIndividual) or a “grouped axis” (formalized by meta-class SwAxisGrouped).

The latter is used to share axis points by several characteristic tables. Figure 5.34 shows an overview on the relevant meta-model elements.

The type of the functional values is given by the attached SwBaseType and the CompuMethod or by the referenced ApplicationDataType.

If an ApplicationDataType is referenced (via valueAxisDataType) this supersedes CompuMethod, Unit, and BaseType if these are defined in parallel.

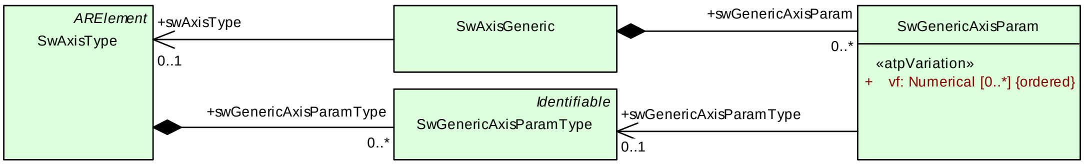


Figure 5.35: Overview on a Generic Axis

Figure 5.36 shows how an individual axis is represented by the meta-model. The corresponding M1 Model is illustrated in Figure 5.37.

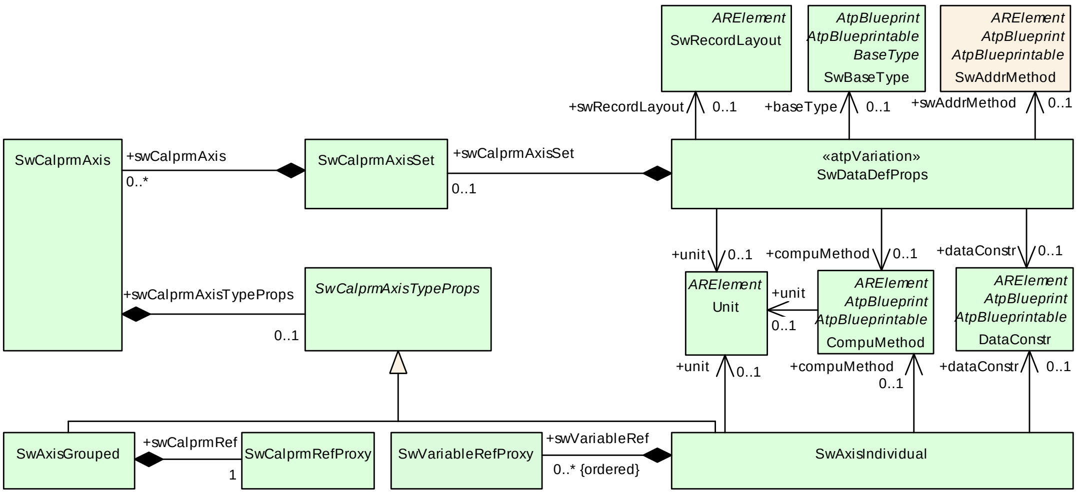


Figure 5.36: Meta-Model Elements used for a Curve

The SwAxisIndividual references value-models to account the minimum and the maximum number of axis values as well as the number of axis points.

Hence, the size of the structure to hold the functional values is determined by the number of axis values for all axes. The type of the axis values is determined when the type of the referenced input value (swVariableRef) has been set. For further details see section 5.4.5.

[TPS_SWCT_01107] swMinAxisPoints and swMaxAxisPoints represent variation points ⌈The value of attributes swMinAxisPoints and swMaxAxisPoints is subject to variant handling.⌋*(RS_SWCT_03148)*

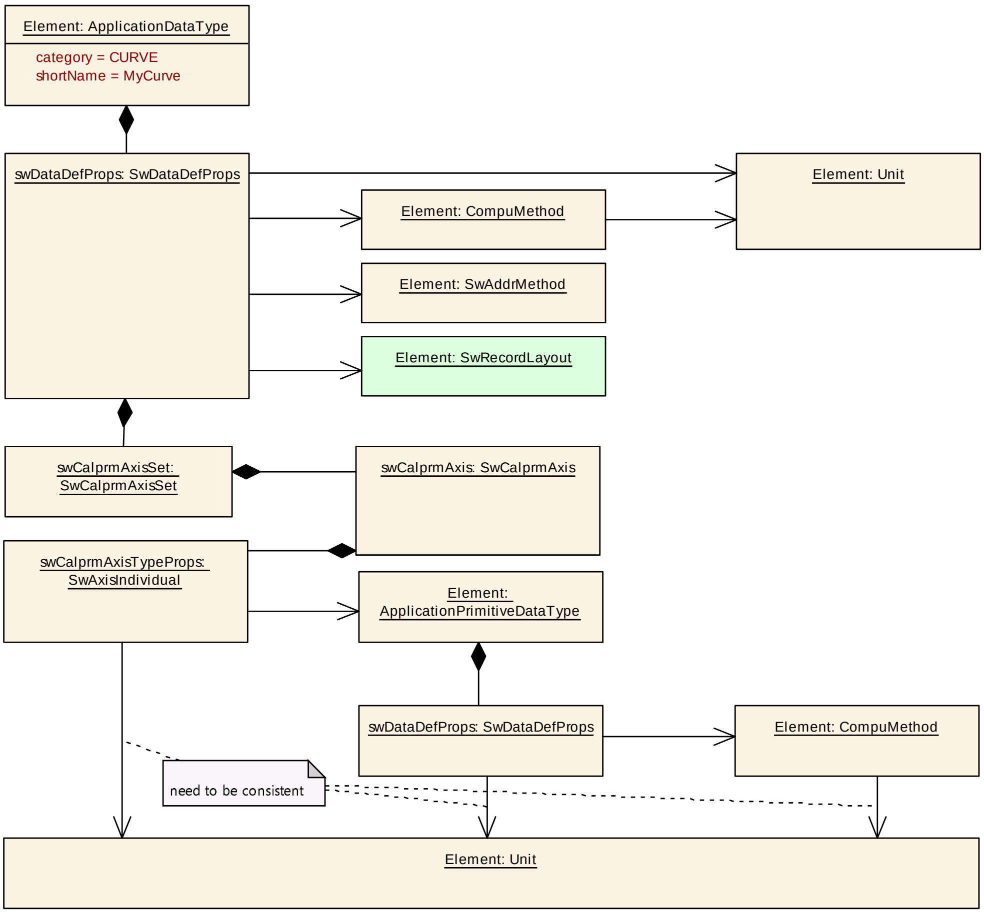


Figure 5.37: Illustration of a Curve in M1

Class	SwCalprmAxisSet			
Package	M2::MSR::DataDictionary::CalibrationParameter			
Note	This element specifies the input parameter axes (abscissas) of parameters (and variables, if these are used adaptively).			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swCalprmAxisSet			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swCalprmAxis	SwCalprmAxis	*	aggr	One axis belonging to this SwCalprmAxisSet **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 5.47: SwCalprmAxisSet

Class	SwCalprmAxis			
Package	M2::MSR::DataDictionary::CalibrationParameter			
Note	This element specifies an individual input parameter axis (abscissa).			
Base	*ARObject*			
Aggregated by	SwCalprmAxisSet.swCalprmAxis			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	CalprmAxisCategoryEnum	0..1	attr	This property specifies the category of a particular axis. **Tags:**xml.sequenceOffset=30
displayFormat	DisplayFormatString	0..1	attr	This property specifies how the axis values shall be displayed e.g. in documents or in measurement and calibration tools. **Tags:**xml.sequenceOffset=100
swAxisIndex	AxisIndexType	0..1	attr	This attribute specifies which axis is specified by the containing SwCalprmAxis. For example in a curve this is usually "1". In a map this is "1" or "2". **Tags:**xml.sequenceOffset=20
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Describes the applicability of parameters and variables. **Tags:**xml.sequenceOffset=90
swCalprmAxisTypeProps	SwCalprmAxisTypeProps	0..1	aggr	specific properties depending on the type of the axis. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=40 xml.typeElement=true xml.typeWrapperElement=false

Table 5.48: SwCalprmAxis

Enumeration	CalprmAxisCategoryEnum
Package	M2::MSR::DataDictionary::CalibrationParameter
Note	This enum specifies the possible values of the category property within SwCalprmAxis.
Aggregated by	RuleBasedAxisCont.category, SwAxisCont.category, SwCalprmAxis.category
Literal	***Description***
comAxis	COM_AXIS is equal to an STD_AXIS, the difference is, that a COM_AXIS is an shared axis, that means this axis can be used multiple times by different CURVEs, MAPs, CUBOIDs, CUBE_4s, and CUBE_5s. **Tags:** atp.EnumerationLiteralIndex=0 xml.name=COM_AXIS
fixAXIS	FIX_AXIS means that the input axis is not stored. The axis is calculated using parameters and so on it is also not possible to modify the axis points. **Tags:** atp.EnumerationLiteralIndex=4 xml.name=FIX_AXIS
resAxis	RES_AXIS is also an shared axis like COM_AXIS, the difference is that this kind of axis can be used for rescaling. **Tags:** atp.EnumerationLiteralIndex=6 xml.name=RES_AXIS

▽

△

Enumeration	**CalprmAxisCategoryEnum**
stdAxis	STD_AXIS means that input and output axis definition are stored within this CURVE, MAP, CUBOID, CUBE_4, and CUBE_5. There is no shared or calculated axis. **Tags:** atp.EnumerationLiteralIndex=8 xml.name=STD_AXIS

Table 5.49: CalprmAxisCategoryEnum

Class	***SwCalprmAxisTypeProps*** (abstract)			
Package	M2::MSR::DataDictionary::CalibrationParameter			
Note	Base class for the type of the calibration axis. This provides the particular model of the specialization. If the specialization would be the directly from SwCalPrmAxis, the sequence of common properties and the specializes ones would be different.			
Base	*ARObject*			
Subclasses	SwAxisGrouped, SwAxisIndividual			
Aggregated by	SwCalprmAxis.swCalprmAxisTypeProps			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
maxGradient	Float	0..1	attr	This attribute defines the maximum permissible gradient for an adjustable object (curve, map or cuboid) with respect to a specific axis. MaxGrad = maximum(absolute((Value i,k - Value i-1,k)/(Axis Point i - Axis Point i-1)))
monotony	MonotonyEnum	0..1	attr	This attribute specifies the monotony constraint for an adjustable object (curve, map or cuboid) with respect to a specific axis. This information can be used by MCD system to verify whether the monotony constraint is fulfilled and to prevent from changes violating the constraint.

Table 5.50: SwCalprmAxisTypeProps

Class	**SwAxisIndividual**			
Package	M2::MSR::DataDictionary::Axis			
Note	This meta-class describes an axis integrated into a parameter (field etc.). The integration makes this individual to each parameter. The so-called grouped axis represents the counterpart to this. It is conceived as an independent parameter (see class SwAxisGrouped).			
Base	*ARObject*, *SwCalprmAxisTypeProps*			
Aggregated by	SwCalprmAxis.swCalprmAxisTypeProps			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
compuMethod	CompuMethod	0..1	ref	This is the compuMethod which is expected for the axis. It is used in early stages if the particular input-value is not yet available. **Tags:**xml.sequenceOffset=30
dataConstr	DataConstr	0..1	ref	Refers to constraints, e.g. for plausibility checks. **Tags:**xml.sequenceOffset=80
inputVariable Type	ApplicationPrimitive DataType	0..1	ref	This is the datatype of the input value for the axis. This allows to define e.g. a type of curve, where the input value is finalized at the access point.

▽

△

Class	SwAxisIndividual			
swAxisGeneric	SwAxisGeneric	0..1	aggr	this specifies the properties of a generic axis if applicable. **Tags:**xml.sequenceOffset=90
swMaxAxisPoints	Integer	0..1	attr	Maximum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
swMinAxisPoints	Integer	0..1	attr	Minimum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=70
swVariableRef (ordered)	SwVariableRefProxy	*	aggr	Refers to input variables of the axis. It is possible to specify more than one variable. Here the following is valid: • The variable with the highest priority shall be given first. It is used in the generation of the code and is also displayed first in the application system. • All variables referenced shall be of the same physical nature. This is usually detected in that the conversion formulae affected refer back to the same SI-units. In AUTOSAR this ensured by the constraint, that the referenced input variables shall use a type compatible to "inputVariableType". • This multiple referencing allows a base point distribution for more than one input variable to be used. One example of this are the temperature curves which can depend both on the induction air temperature and the engine temperature. These variables can be displayed simultaneously by MCD systems (adjustment systems), enabling operating points to be shown in the curves. **Tags:** xml.roleElement=false xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
unit	Unit	0..1	ref	This represents the physical unit of the input value of the axis. It is provided to support the case that the particular input variable is not yet known. **Tags:**xml.sequenceOffset=40

Table 5.51: SwAxisIndividual

Class	**SwAxisGeneric**			
Package	M2::MSR::DataDictionary::Axis			
Note	This meta-class defines a generic axis. In a generic axis the axispoints points are calculated in the ECU. The ECU is equipped with a fixed calculation algorithm. Parameters for the algorithm can be stored in the data component of the ECU. Therefore these parameters are specified in the data declaration, not in the calibration data.			
Base	*ARObject*			
Aggregated by	SwAxisIndividual.swAxisGeneric			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swAxisType	SwAxisType	0..1	ref	Associated axis calculation strategy. **Tags:**xml.sequenceOffset=20
swGenericAxisParam	SwGenericAxisParam	*	aggr	Specific parameter of a generic axis. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false

Table 5.52: SwAxisGeneric

Class	**SwAxisType**			
Package	M2::MSR::DataDictionary::Axis			
Note	This meta-class represents a specific axis calculation strategy. No formal specification is given, due to the fact that it is possible to use arbitrary algorithms for calculating axis-points. Instead, the algorithm is described verbally but the parameters are specified formally with respect to their names and constraints. As a result, SwAxisType mainly reserves appropriate keywords. **Tags:**atp.recommendedPackage=SwAxisTypes			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swGenericAxisDesc	DocumentationBlock	0..1	aggr	Associated axis description in textual form. **Tags:**xml.sequenceOffset=20
swGenericAxisParamType	SwGenericAxisParamType	*	aggr	Parameters for this calculation algorithm. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 5.53: SwAxisType

Class	**SwGenericAxisParam**
Package	M2::MSR::DataDictionary::Axis
Note	This meta-class describes a specific parameter of a generic axis. The name of the parameter is defined through a reference to a parameter type defined on a corresponding axis type. The value of the parameter is given here in case that it is not changeable during calibration. Example is shift / offset in a fixed axis.
Base	*ARObject*
Aggregated by	SwAxisGeneric.swGenericAxisParam

▽

△

Class	SwGenericAxisParam			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swGenericAxisParamType	SwGenericAxisParamType	0..1	ref	Parameter type defined on a corresponding axis type. References can only be made to axis parameters types which are defined within the referenced axis type. **Tags:**xml.sequenceOffset=20
vf (ordered)	Numerical	*	attr	This attribute represents the value of the generic axis parameter. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false

Table 5.54: SwGenericAxisParam

Class	SwGenericAxisParamType			
Package	M2::MSR::DataDictionary::Axis			
Note	This meta-class describes a generic axis parameter type, namely: • Plausibility checks can be specified via dataConstr. • Textual description (desc), as a formal description is not of any use, due to the large variety of possibilities. • If this parameter contains structures, these can be simulated through the recursive use of SwGenericAxisParamTypes.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	SwAxisType.swGenericAxisParamType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataConstr	DataConstr	0..1	ref	This reference denoted data constraints applicable to the generic axis parameter. **Tags:**xml.sequenceOffset=20

Table 5.55: SwGenericAxisParamType

Class	SwAxisGrouped			
Package	M2::MSR::DataDictionary::Axis			
Note	An SwAxisGrouped is an axis which is shared between multiple calibration parameters.			
Base	*ARObject*, *SwCalprmAxisTypeProps*			
Aggregated by	SwCalprmAxis.swCalprmAxisTypeProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
sharedAxisType	ApplicationPrimitiveDataType	0..1	ref	This is the datatype of the calibration parameter providing the shared axis.

▽

△

Class	**SwAxisGrouped**			
swAxisIndex	AxisIndexType	0..1	attr	Describes which axis of the referenced calibration parameter provides the values for the group axis. The index satisfies the following convention: • 0 = value axis. in this case, the interpolation result of the referenced parameter is used as a base point index. • The index should only be specified if the parameter under swCalprm contains more than one axis. It is standard practice for the axis index of parameters with more than one axis, to be set to 1, if data has not been assigned to swAxis Index. **Tags:**xml.sequenceOffset=20
swCalprmRef	SwCalprmRefProxy	1	aggr	This property specifes the calibration parameter which serves as the input axis. In AUTOSAR, the type of the referenced Calibration parameter shall be compatible to the type specified by sharedAxisType. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 5.56: SwAxisGrouped

5.4.4.1 Specification of grouped Axes

Please note that SwAxisGrouped has a dual nature in that it is used to

- define the data type of the shared axis (see Figure 5.38)
- identify the specific DataPrototype that implements a shared axis in the context of a specific access to the enclosing curve (see Figure 5.39)

Figure 5.38 depicts the usage of SwAxisGrouped for the definition of an ApplicationPrimitiveDataType that implements a curve.

It is worth noticing that ApplicationPrimitiveDataType appears in two different roles in the diagram:

- The definition of the ApplicationPrimitiveDataType of the curve itself is depicted in the upper part of the diagram.
- The selection of the ApplicationPrimitiveDataType that represents the grouped axis is depicted in the lower part of the diagram.

The modeling depicted in Figure 5.38 can be further clarified by an example model that is reduced to the minimum content to exemplify the use case. Here, the role of the usage of SwAxisGrouped is to define the data type of the curve itself and of a shared axis:

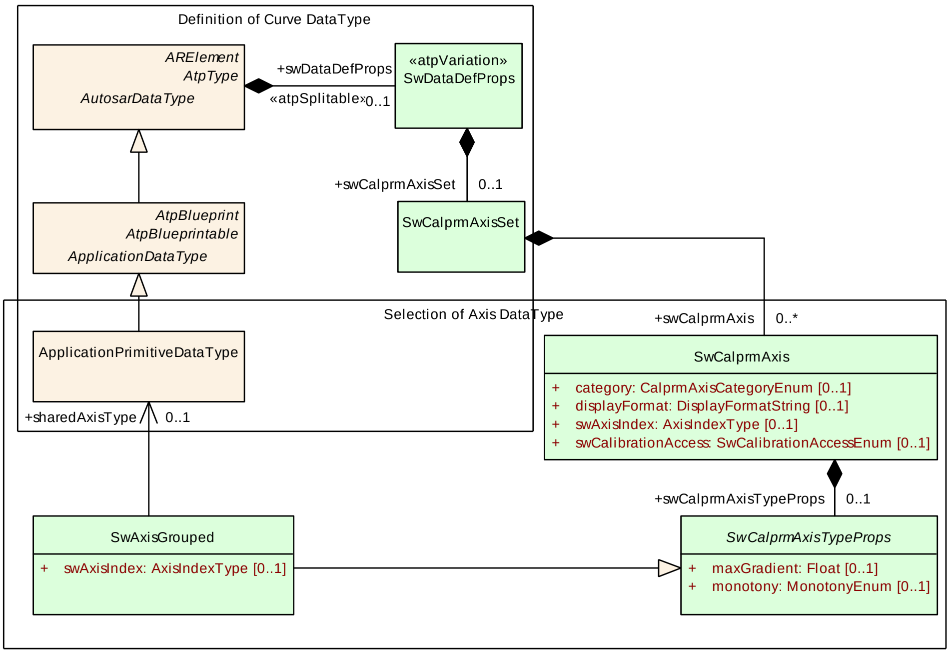


Figure 5.38: Definition of an ApplicationPrimitiveDataType that implements a curve

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyAppType</SHORT-NAME>
  <CATEGORY>CURVE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALPRM-AXIS-SET>
          <SW-CALPRM-AXIS>
            <SW-AXIS-GROUPED>
              <SHARED-AXIS-TYPE-REF
              DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/ApplicationDataTypes/
                  MyAxisDataType</SHARED-AXIS-TYPE-REF>
            </SW-AXIS-GROUPED>
          </SW-CALPRM-AXIS>
        </SW-CALPRM-AXIS-SET>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyAxisDataType</SHORT-NAME>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.12: Usage of SwAxisGrouped to define the data type of a shared axis

Figure 5.39 depicts the usage of SwAxisGrouped in the context of a ParameterAccess (which, in the case of a curve, typically results in the call of an interpolation method for the respective curve and the specific axis that is supposed to be used in the interpolation).

Also in this case, the diagram depicts two distinct (yet related) aspects:

- In the upper part of the diagram, the definition of the access to the DataPrototype that represents the specific curve inside the enclosing SwComponentPrototype is modeled by means of the ParameterAccess.accessedParameter.
- In the lower part of the diagram, the identification of the DataPrototype that represents the grouped axis inside the enclosing SwComponentPrototype is modeled by means of the SwAxisGrouped.swCalprmRef.arParameter.

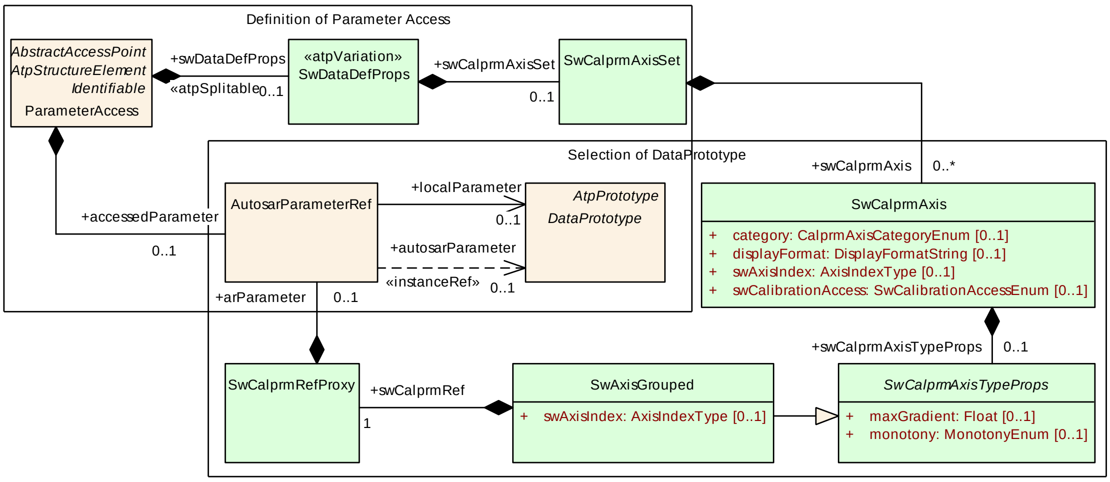


Figure 5.39: Usage of SwAxisGrouped in the context of a ParameterAccess

The following bare-bones model listing exemplifies the usage of SwAxisGrouped in the context of a ParameterAccess where a specific local parameter is identified to take the role of the shared axis in the specific access to the curve.

```
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>A</SHORT-NAME>
  <INTERNAL-BEHAVIORS>
    <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME>B</SHORT-NAME>
      <PER-INSTANCE-PARAMETERS>
        <PARAMETER-DATA-PROTOTYPE>
          <SHORT-NAME>Axis</SHORT-NAME>
        </PARAMETER-DATA-PROTOTYPE>
        <PARAMETER-DATA-PROTOTYPE>
          <SHORT-NAME>Curve</SHORT-NAME>
        </PARAMETER-DATA-PROTOTYPE>
      </PER-INSTANCE-PARAMETERS>
      <RUNNABLES>
        <RUNNABLE-ENTITY>
          <SHORT-NAME>MyRunnable</SHORT-NAME>
          <PARAMETER-ACCESSS>
            <PARAMETER-ACCESS>
              <SHORT-NAME>MyParamAccess</SHORT-NAME>
              <ACCESSED-PARAMETER>
                <LOCAL-PARAMETER-REF DEST="AUTOSAR-DATA-PROTOTYPE">/P/A/B/
                   Curve</LOCAL-PARAMETER-REF>
              </ACCESSED-PARAMETER>
              <SW-DATA-DEF-PROPS>
```

```
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-CALPRM-AXIS-SET>
                      <SW-CALPRM-AXIS>
                        <SW-AXIS-GROUPED>
                          <AR-PARAMETER>
                            <LOCAL-PARAMETER-REF DEST="AUTOSAR-DATA-
                                PROTOTYPE">/P/A/B/Axis</LOCAL-PARAMETER-REF>
                          </AR-PARAMETER>
                        </SW-AXIS-GROUPED>
                      </SW-CALPRM-AXIS>
                    </SW-CALPRM-AXIS-SET>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </PARAMETER-ACCESS>
          </PARAMETER-ACCESSS>
        </RUNNABLE-ENTITY>
      </RUNNABLES>
    </SWC-INTERNAL-BEHAVIOR>
  </INTERNAL-BEHAVIORS>
</APPLICATION-SW-COMPONENT-TYPE>
```

Listing 5.13: Usage of SwAxisGrouped in the context of a ParameterAccess

[constr_10096] Shared axis shall not be a fixed axis ⌈An ApplicationPrimitiveDataType of category COM_AXIS shall not contain the definition of an SwCalprmAxis of category FIX_AXIS.

This rule shall be imposed **at the time when the contract phase generation is executed**⌋*()*

Rationale for the existence of [constr_10096]: the definition of a grouped axis that is implemented by a fixed axis is not supported by the A2L [28] standard.

5.4.4.2 Specification of fix Axes

In most cases the axes of a curve or map are accessible to a calibration software and it is possible to calibrate axes points and their corresponding values.

There are cases, however, where axes are intentionally declared as fix and where no intention exists to change the properties of the axis ever[17].

These axes are also known as fix axes. The support for the creation of fix axes in the meta-model is based upon the usage of SwAxisGeneric as depicted in Figure 5.35.

[TPS_SWCT_01747] Value of category for fix axis ⌈A fix axis shall be modeled as an SwCalprmAxis with attribute category set to the value FIX_AXIS.⌋*()*

[17]Typically, a calibration software does not have the ability to manipulate (or even inspect) the axis' properties by inspecting the ECU's memory.

[TPS_SWCT_01748] Sub-categories of fix axes ⌈There are different sub-categories of fix axes:

- Fix axis where the distance between axis points can be computed according to a standardized algorithm.

  In this case, fix axes of arbitrary length can be described by feeding three arguments defined in the context of the axis description into the axis algorithm.

  Consequently, the memory footprint of different fix axis of this category is literally identical, independently of the number of axis points.

  The following variations exist:

  – **Subcategory PAR**, i.e. category = FIX_AXIS_PAR: the axis is created out of a *starting value* and a *shift* that creates further axis points as using a power-of-two algorithm. The details can be found in [28].

  – **Subcategory PAR_DIST**, i.e. category = FIX_AXIS_PAR_DIST: the axis is created out of a *starting value* and an *offset* that adds further axis points with the distance given by offset. The details can be found in [28].

- Fix axis where the axis points are defined as a list of values directly in the axis definition. This variety boils down to

  – **Subcategory PAR_LIST**, i.e. category = FIX_AXIS_PAR_LIST: the axis is created out of a list of numerical values that represent the axis points. The details can be found in [28].

These values of category shall be used for SwAxisType.⌋*()*

As mentioned before, the modeling of a fix axis is based upon the definition of the SwAxisGeneric. But this statement by itself is not yet sufficient to unambiguously clarify the details of the modeling.

For this purpose, it is necessary to provide further information about the specifics of the roles SwAxisGeneric.swAxisType and SwAxisGeneric.swGenericAxisParam.

[TPS_SWCT_01749] Semantics of SwAxisGeneric.swAxisType in the definition of a fix axis ⌈The role SwAxisGeneric.swAxisType specifies the category of the fix axis according to [TPS_SWCT_01748].⌋*()*

[TPS_SWCT_01750] Semantics of SwAxisGeneric.swGenericAxisParam in the definition of a fix axis ⌈The role SwAxisGeneric.swGenericAxisParam provides the actual numeric values for the definition of the axis.

The semantics of a provided numerical value is clarified by the attribute SwGenericAxisParamType.category where meta-class SwGenericAxisParamType is referenced in the role swGenericAxisParamType.⌋*()*

Please note that the standardized values and multiplicities within the model of an SwAxisGeneric are defined in accordance with [TPS_SWCT_01479] and [TPS_SWCT_01480].

[constr_1544] Standardized values and multiplicities for the modeling of SwAxisGeneric for the definition of a fix axis ⌈

category of swAxisType	category of SwGenericAxisParamType	Multiplicity of swGenericAxisParam	Multiplicity of vf
FIX_AXIS_PAR	OFFSET	1	1
	SHIFT	1	1
FIX_AXIS_PAR_DIST	OFFSET	1	1
	DISTANCE	1	1
FIX_AXIS_PAR_LIST	LIST	1	1..*

This rule shall be imposed at the time when the contract phase generation is executed.

⌋()

The modeling of an axis of category FIX_AXIS_PAR is sketched in the following example model (Figure 5.40).

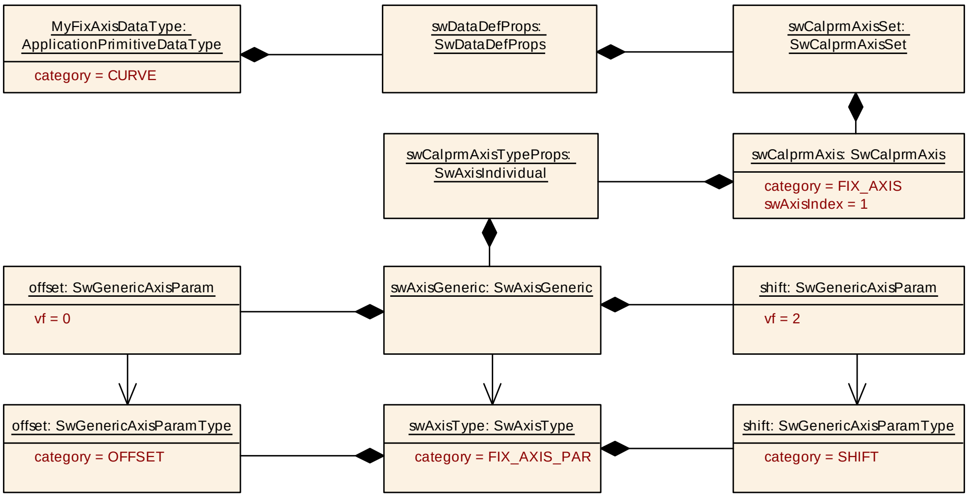


Figure 5.40: Modeling of a fix axis of category FIX_AXIS_PAR

The modeling of an axis of category FIX_AXIS_PAR_DIST is sketched in the following example model (Figure 5.41).

Please note that the axis points and values of a fix axis are defined in the definition of the fix axis itself and therefore any initial value assigned to a fix axis would be ignored anyway.

This might lead to confusion such that the initial value does not make it into the software. In order to avoid such confusion AUTOSAR **does not support the definition of an initial value for a fix axis**.

This regulation is reflected in the existence of [constr_1545].

[constr_1545] No initialization for fix axis ⌈An ApplicationValueSpecification taken to initialize an ApplicationPrimitiveDataType that contains a fix axis shall not contain initial values for the axis index of the fix axis inside the ApplicationPrimitiveDataType **at any time in the workflow**.⌋()

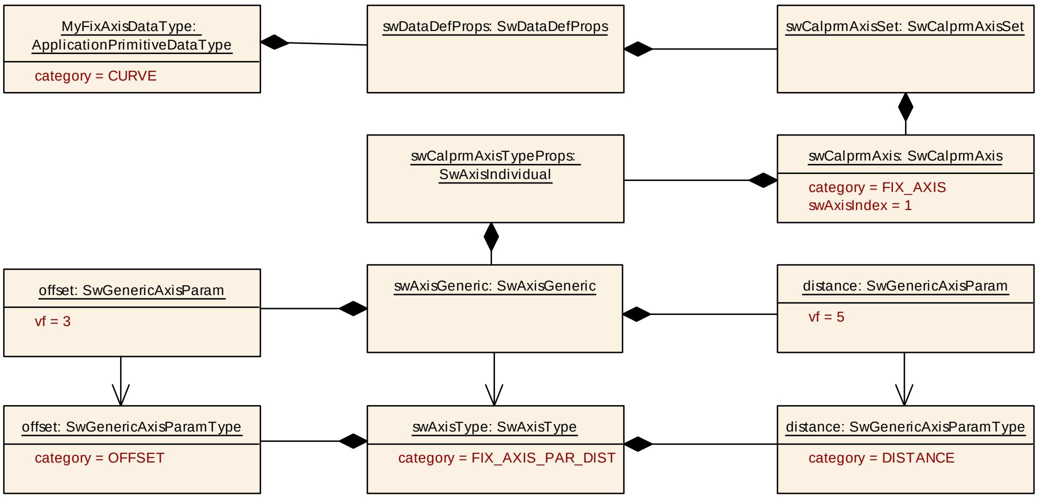


Figure 5.41: Modeling of a fix axis of category FIX_AXIS_PAR_DIST

The modeling of an axis of category FIX_AXIS_PAR_LIST is sketched in the following example model (Figure 5.42).

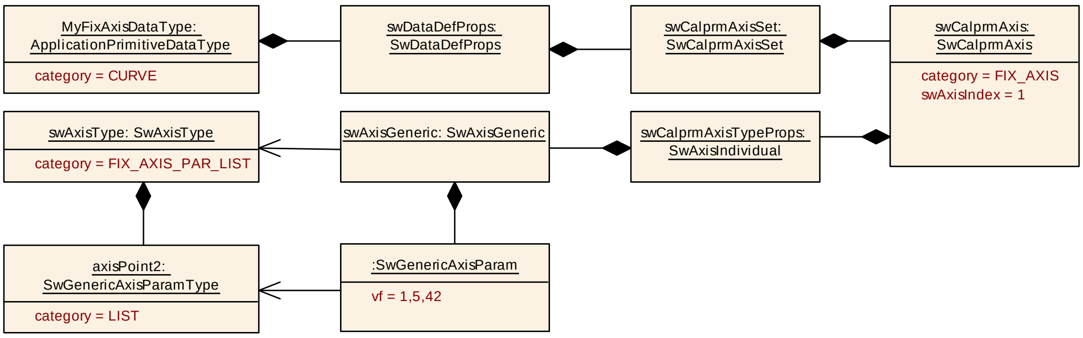


Figure 5.42: Modeling of a fix axis of category FIX_AXIS_PAR_LIST

Please note that the calibration software may still have access to axis points and values of the fix axis if these properties are specified in an A2L file.

For this purpose McDataInstance needs to be set up properly. The details are explained in [6].

5.4.5 Setting an Axis Input Value

When an interpolation routine is called, an input value has to be provided to find the appropriate axis entry in the implementation of a RunnableEntity. However, this input value cannot be arbitrarily chosen but only be selected from available VariableDataPrototype assigned to it.

Attributes (e.g. displayFormat, swCalibrationAccess) of a shared axis can be defined in the context of the definition of the ApplicationPrimitiveDataType of category COM_AXIS for the shared axis.

The same ability to define attributes[18] is also required for the definition of an "embedded" stdAxis.

There are several possible ways of how the required attributes can be defined, e.g. by means of the existence of the reference in the role inputVariableType if SwCalprmAxis.swCalprmAxisTypeProps yields an SwAxisIndividual, see [TPS_SWCT_01865].

Please note that the modeling of SwCalprmAxis and SwAxisIndividual, as mentioned in [TPS_SWCT_01865] and [TPS_SWCT_01866], is depicted in Figure 5.27.

[TPS_SWCT_01865] Definition of attributes for an "embedded" axis if reference inputVariableType exists ⌈If SwCalprmAxis.swCalprmAxisTypeProps yields an SwAxisIndividual and if the reference in the role SwAxisIndividual.inputVariableType exists, then the definition of attributes (e.g. displayFormat, swCalibrationAccess) of an SwCalprmAxis of category stdAxis can be done via the SwDataDefProps attached to the ApplicationPrimitiveDataType referenced in the role SwCalprmAxis.swCalprmAxisTypeProps.inputVariableType.⌋*()*

[TPS_SWCT_01866] Definition of attributes for an "embedded" axis if reference inputVariableType does not exist ⌈If SwCalprmAxis.swCalprmAxisTypeProps yields an SwAxisIndividual and if the reference in the role SwAxisIndividual.inputVariableType does not exist, then the attributes can be taken from the AutosarDataTypes of DataPrototypes referenced from a swVariableRef that is aggregated by an SwAxisIndividual, that in turn is obtained in the role swCalprmAxisTypeProps from an enclosing SwCalprmAxis.

In this case, the respective attributes have to be compatible in all AutosarDataTypes used to type DataPrototypes referenced from within the role SwAxisIndividual.swVariableRef.⌋*()*

This leaves the consideration of compatibility between the DataPrototype(s) referenced by means of SwAxisIndividual.swVariableRef and the actual axis specification to the following attributes:

- SwAxisIndividual.dataConstr
- SwAxisIndividual.compuMethod
- SwAxisIndividual.unit

[TPS_SWCT_01676] Preferred approach to checking the compatibility of input value and axis ⌈The compatibility in terms of data type between the description of an SwAxisIndividual and the DataPrototype(s) used as an input variable to the respective interpolation routine shall preferably be checked alternatively between

[18] Some attributes, like CompuMethod or DataConstr of an "embedded" axis can be directly defined on the basis of the SwAxisIndividual, but those are just individual shortcuts.

- the ApplicationPrimitiveDataType(s) of DataPrototype(s) referenced by means of SwAxisIndividual.swVariableRef (the provider in terms of compatibility)
- the ApplicationPrimitiveDataType referenced by means of SwAxisIndividual.inputVariableType (The requester in terms of compatibility).

For compatibility, the compuMethod of SwAxisIndividual.swVariableRef and the ApplicationPrimitiveDataType referenced by means of SwAxisIndividual.inputVariableType shall not be considered.⌋*()*

Rationale: in many cases the input variable is defined by a float data type to take benefit from the precision in computations. But the axis data type is an integer data type to save memory. In this situation, a requirement for compatible compuMethods would exclude the described scenario.

The implementation of the software-component shall make sure that the float value is properly converted and rescaled to an integer data type compatible to the axis data type.

[TPS_SWCT_01677] Fall-back approach to checking the compatibility of input value and axis ⌈If the reference SwAxisIndividual.inputVariableType does not exist then the compatibility in terms of data type between the description of an SwAxisIndividual and the DataPrototype(s) used as an input variable to the respective interpolation routine shall be checked on the basis of the following references:

- SwAxisIndividual.dataConstr
- SwAxisIndividual.unit

respectively

- SwAxisIndividual.dataConstr
- SwAxisIndividual.compuMethod.unit

against their respective counterparts in the ApplicationPrimitiveDataTypes of the DataPrototype(s) referenced by means of SwAxisIndividual.swVariableRef.⌋*()*

[constr_1420] Existence of SwAxisIndividual.inputVariableType ⌈If the reference SwAxisIndividual.inputVariableType does not exist then either:

- SwAxisIndividual.dataConstr
- SwAxisIndividual.unit

or

- SwAxisIndividual.dataConstr
- SwAxisIndividual.compuMethod.unit

shall exist **at the time when the contract phase generation is executed.**⌋()

The constraint is necessary for the generation of the respective specification of the axis in A2L.

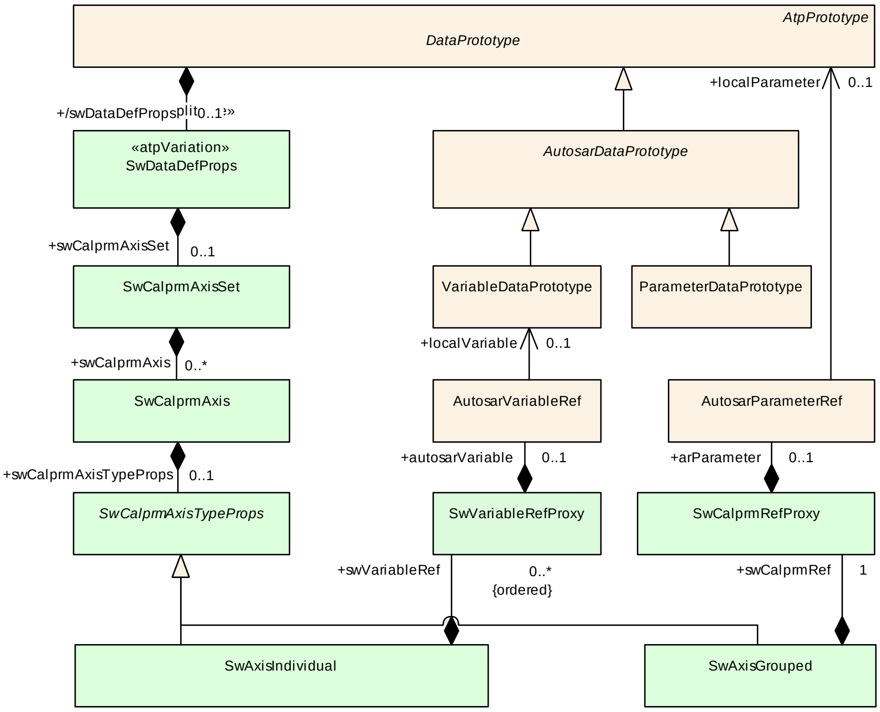


Figure 5.43: Extended Axis Elements and Input Variable Reference

Every `ParameterDataPrototype` then allows to specify zero or more input values (being type compatible to `inputVariableType`) in its axis description.

This means that at the specification time of an `SwcInternalBehavior`, a list of input values has to be specified where the implementer of a `RunnableEntity` can choose of. The input values are `DataPrototype` entities either being

- a `VariableDataPrototype` in a `SenderReceiverInterface` or `NvDataInterface` of a `PortPrototype`, of the `AtomicSwComponentType` where the `SwcInternalBehavior` is associated to, or an `ArgumentDataPrototype` in a `ClientServerOperation` of a `ClientServerInterface` in a `PortPrototype` of the `AtomicSwComponentType` where the `InternalBehavior` is associated to, or
- a `VariableDataPrototype` within the `SwcInternalBehavior`.

To achieve this, `SwAxisIndividual` is aggregating a `SwVariableRefProxy`.

Originally, MSRSW uses a `AutosarVariableRef` to set the input value of an axis appropriately. In AUTOSAR, this has been extended by first introducing a `SwVariableRefProxy`.

Note that this is a specific use case for the role SwVariableRefProxy.autosarVariable.

Note further that the use cases for the existence of the attributes SwVariableRefProxy.autosarVariable and SwVariableRefProxy.mcDataInstanceVar are entirely disjoint and therefore the simultaneous existence of these two attributes would not make any sense at all.

Therefore, [constr_1382] has been introduced to clarify this aspect.

[constr_1382] Mutually exclusive existence of attributes SwVariableRefProxy.autosarVariable vs. SwVariableRefProxy.mcDataInstanceVar ⌈In any given AUTOSAR model, the aggregations SwVariableRefProxy.autosarVariable and SwVariableRefProxy.mcDataInstanceVar shall never exist at the same time **at any time in the workflow**.⌋*()*

As shown in Figure 5.43, this approach is also used to represent a AutosarVariableRef in all roles, e.g. the result of an interpolation routine applied to an axis, the input value determination, a list of dependent parameters, and swDataDependency.

With the means of ApplicationArrayDataTypes it's possible to define DataPrototypes holding an n-dimensional array of Compound Primitive Data Types of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, or RES_AXIS.

For those DataPrototypes input values for the axes should be described to enable a display of the working point in the MCD system.

Thereby, typically the whole array of the contained axes is either associated with an array of variables or with a single value. In the case of arrays typically the n-th axis is combined with the n-th input value.

[constr_1425] Definition of swCalprmAxisSet.swCalprmAxis / SwAxisIndividual.swVariableRef depending on the capabilities of the data type ⌈The definition of a swCalprmAxisSet.swCalprmAxis / SwAxisIndividual.swVariableRef in the context of an InstantiationDataDefProps or a ParameterAccess is only supported for a DataPrototype of category ARRAY if the data type of the ApplicationArrayElement also supports the specification of a swCalprmAxisSet.swCalprmAxis / SwAxisIndividual.swVariableRef according to [constr_1289].

Thereby, multiple ApplicationArrayDataTypes might be nested to express multiple array dimensions. This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01683] Specification of an array of input variable for an array of axes ⌈For DataPrototypes typed by an array of elements of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, or RES_AXIS the applied InstantiationDataDefProps or ParameterAccess may reference a VariableDataPrototype typed by an ApplicationArrayDataType with the means of SwAxisIndividual.swVariableRef.autosarVariable.

This expresses the semantic that the n^th element in the axis array uses the n^th value in the input variable array for the specific SwAxisGrouped.swAxisIndex.⌋()

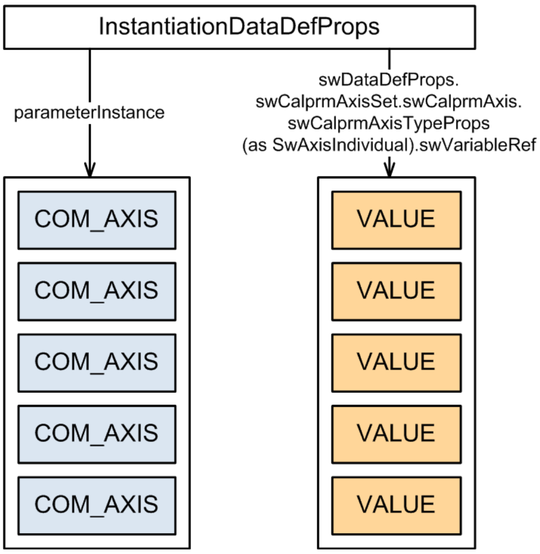


Figure 5.44: The n^th COM_AXIS in the array of COM_AXISs uses the n^th VALUE in the array of VALUEs as working point.

Please note that in this case the two associated arrays needs to have same number of dimensions and sizes of the dimensions.

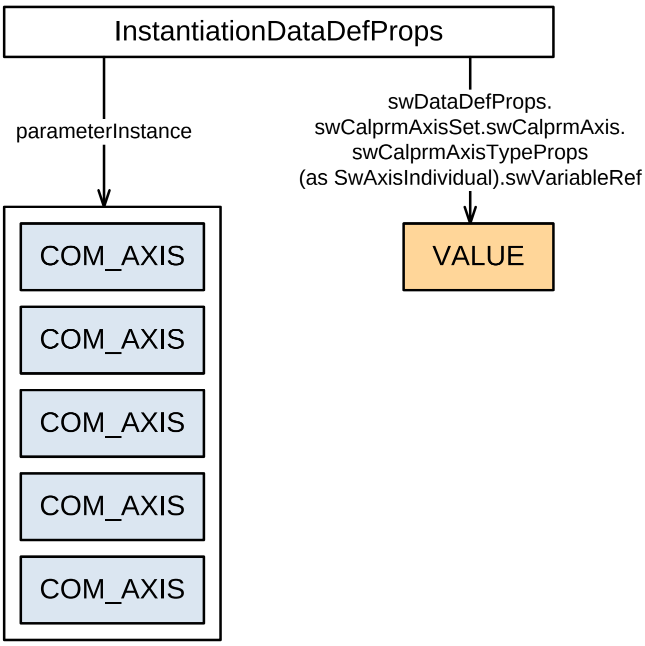


Figure 5.45: Each COM_AXIS in the array of COM_AXISs uses the identical VALUE as working point.

[constr_1426] Consistency of array sizes for axes and input variable array ⌈The number of array dimension defined by ApplicationArrayDataTypes and the values of the maxNumberOfElements attributes for the array of elements of category

CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, or RES_AXIS shall be **identical** to the number of array dimension and according value of the maxNumberOfElements of the VariableDataPrototype referenced by SwAxisIndividual.swVariableRef.autosarVariable.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01684] Specification of a single input variable for an array of axes ⌈For DataPrototypes typed by an array of elements of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, or RES_AXIS the applied InstantiationDataDefProps or ParameterAccess may reference a VariableDataPrototype typed by an ApplicationPrimitiveDataType with the means of SwAxisIndividual.swVariableRef.autosarVariable.

This expresses the semantic that each element in the axis array uses the identical input variable for the specific SwAxisGrouped.swAxisIndex.⌋*()*

5.4.6 Setting a Group Axis

Grouped curves share the same axis definition. In MSRSW, this is shown by referencing the SwCalprm, representing an individual curve, from a SwAxisGrouped.

Note that this does not describe which axis shall be taken from a reference swCalprmRef acting as a shared axis. This would be done in SwAxisGrouped.swAxisIndex.

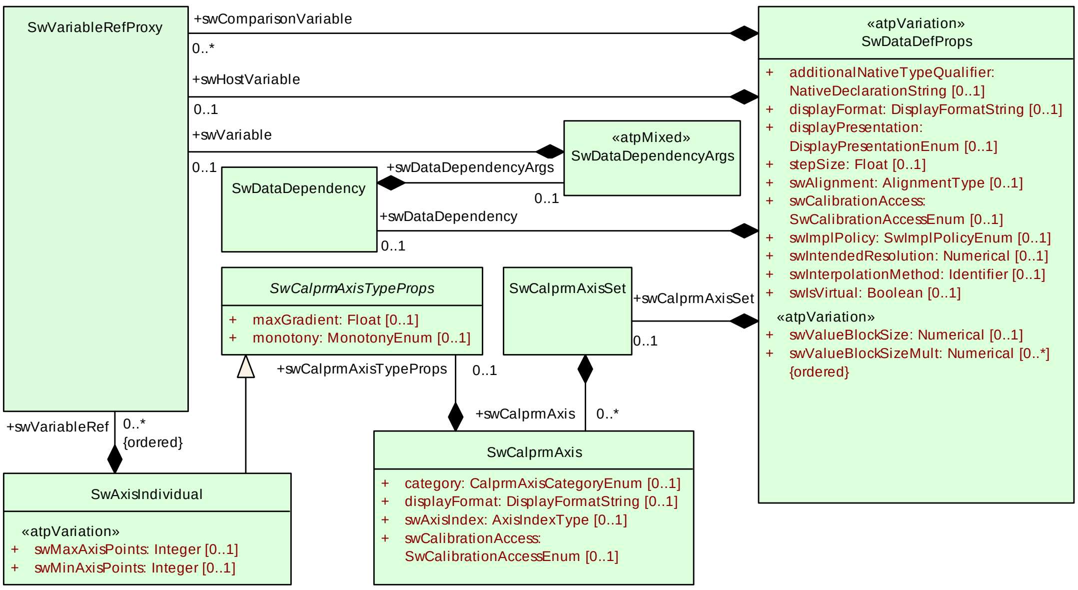


Figure 5.46: Applying Proxy Variable Reference Mechanism

AUTOSAR applies a similar proxy approach for parameters as for the variables. Therefore, an SwCalprmRefProxy has been introduced in MSRSW, and is aggregated by the SwAxisGrouped element.

The SwCalprmRefProxy aggregates an AutosarParameterRef providing an association to a ParameterDataPrototype, representing a curve with an axis. When defining the data-type of a parameter, the type of the shared axis is defined in sharedAxisType.

[constr_1020] ParameterDataPrototype needs to be of compatible data type as referenced in sharedAxisType ⌈Finally, the ParameterDataPrototype assigned in swCalprmRef shall be typed by data type compatible to sharedAxisType **at the time when the contract phase generation is executed.**⌋()

The AUTOSAR-style is shown in the upper left part of Figure 5.43, while in the upper middle the MSRSW style is shown, referencing the SwCalprm.

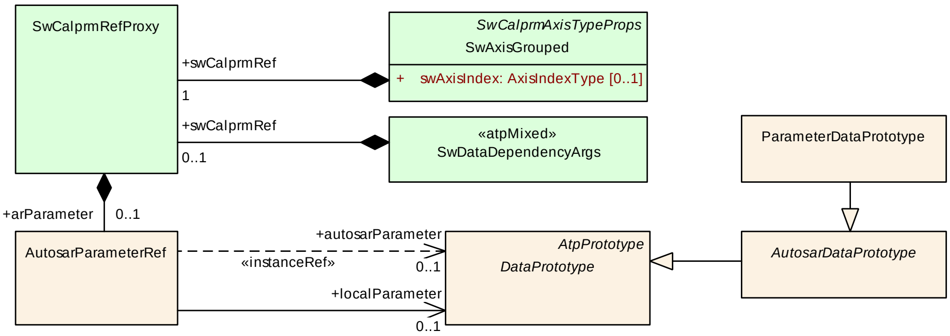


Figure 5.47: Applying Proxy Parameter Reference Mechanism

Class	**SwCalprmRefProxy**			
Package	M2::MSR::DataDictionary::DatadictionaryProxies			
Note	Wrapper class for different kinds of references to a calibration parameter.			
Base	*ARObject*			
Aggregated by	SwAxisGrouped.swCalprmRef, SwDataDependencyArgs.swCalprmRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arParameter	AutosarParameterRef	0..1	aggr	This represents a Parameter within AUTOSAR. Note that the Datatype of the referenced ParameterDataPrototype shall be an ApplicationDataType of category VALUE.
mcDataInstance	McDataInstance	0..1	ref	This reference is used in the McSupport file to express the final instance of group axis etc. It is not allowed to use this outside of an McDataInstance. The referenced mcDataInstance shall be originated from a ParameterDataPrototype.

Table 5.57: SwCalprmRefProxy

Class	**SwVariableRefProxy**			
Package	M2::MSR::DataDictionary::DatadictionaryProxies			
Note	Proxy class for several kinds of references to a variable.			
Base	*ARObject*			
Aggregated by	SwAxisIndividual.swVariableRef, SwDataDefProps.swComparisonVariable, SwDataDefProps.swHostVariable, SwDataDependencyArgs.swVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	SwVariableRefProxy			
autosarVariable	AutosarVariableRef	0..1	aggr	This represents the reference to a Variable in an Autosar system. Note that the target of the reference within AutosarVariableRef shall be typed by a primitive data type
mcDataInstanceVar	McDataInstance	0..1	ref	This reference is used in the McSupport file to express the final instance of input values etc. It is not allowed to use this outside of an McDataInstance. The referenced mcDataInstance shall be originated from a VariableDataPrototype.

Table 5.58: SwVariableRefProxy

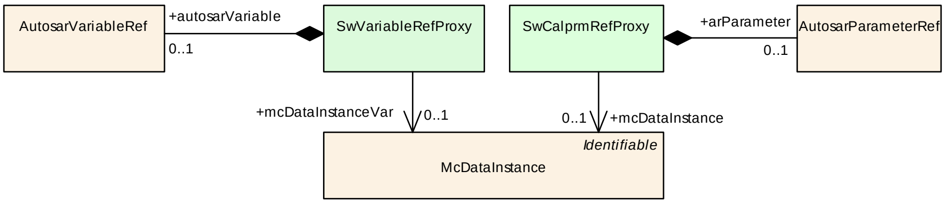


Figure 5.48: Proxy reference classes

The basic patterns for referencing DataPrototypes are explained in section 5.3.3. In the context of this chapter it is worth to remark that the definition of access to calibration parameters is implemented in the context of a RunnableEntity (see Figure 7.4).

[TPS_SWCT_01846] Use InstantiationDataDefProps to facilitate the definition of access to calibration parameters ⌈As the definition of a calibration parameter may involve the definition of several axes the necessity to provide this amount of information might become cumbersome and (to some extent) redundant and difficult to maintain if the same calibration parameter is accessed from within several RunnableEntitys. In other words: in this case it would be necessary to repeat the more or less complex set of information for each RunnableEntity.

To avoid this unnecessary level of complexity for the definition of access to calibration parameters, it is possible to define the access to the calibration parameter on the level of InstantiationDataDefProps which have been defined to facilitate this kind of re-use.⌋*()*

For more information please refer to section 7.5.4. This ability is also documented in [constr_1015].

With the means of ApplicationArrayDataTypes its possible to define DataPrototypes holding an n-dimensional array of Compound Primitive Data Types of category CURVE, MAP, CUBOID, CUBE_4, and CUBE_5.

For those DataPrototypes, group axis/axes needs to be defined in case SwAxisIndividuals are not used for all SwCalprmAxis definitions.

Thereby, typically the whole array of elements of category CURVE, MAP, CUBOID, CUBE_4, and CUBE_5 is either associated with an array of group axes or alternatively with a single group axis.

In the case of arrays typically the n^{th} CURVE, MAP, CUBOID, CUBE_4, and CUBE_5 is combined with the n^{th} COM_AXIS or RES_AXIS.

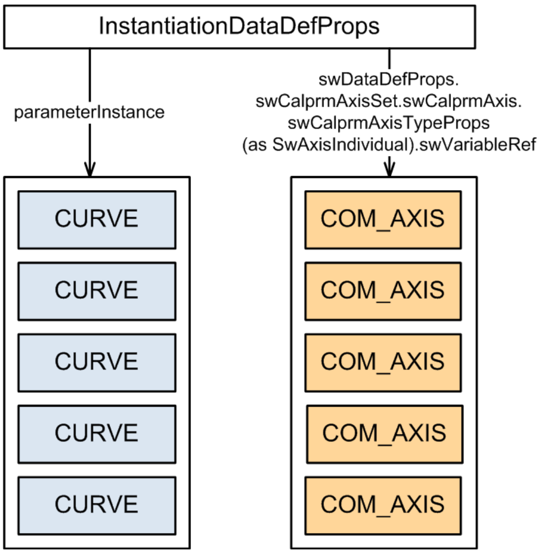


Figure 5.49: The n^{th} CURVE in the array of CURVEs relates to the n^{th} COM_AXIS in the array of COM_AXISs

[constr_1427] Definition of swCalprmAxisSet.swCalprmAxis / SwAxisGrouped.swCalprmRef depending on the capabilities of the data type ⌈The definition of a swCalprmAxisSet.swCalprmAxis / SwAxisGrouped.swCalprmRef in the context of an InstantiationDataDefProps or a ParameterAccess is only supported for a DataPrototype of category ARRAY if the data type of the ApplicationArrayElement also supports the specification of a swCalprmAxisSet.swCalprmAxis/ SwAxisGrouped.swCalprmRef according to [constr_1289].

Thereby, multiple ApplicationArrayDataTypes might be nested to express multiple array dimensions. This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01685] Specification of an array of group axes for an array of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 ⌈For DataPrototypes typed by an array of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 the applied InstantiationDataDefProps or ParameterAccess may reference a DataPrototype typed by an ApplicationArrayDataType with the means of SwAxisGrouped.swCalprmRef.arParameter.

This expresses the semantic that the n^{th} element in the CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 array uses the n^{th} group axis in the COM_AXIS or RES_AXIS array for the specific SwAxisGrouped.swAxisIndex.⌋*()*

Please note that in this case the two associated arrays needs to have same number of dimensions and sizes of the dimensions.

[constr_1428] Consistency of array sizes for arrays of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 arrays and used group axes arrays ⌈The number of array dimension defined by ApplicationArrayDataTypes and the values of attribute maxNumberOfElements attributes for the array of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 needs to be identical to the number of array dimension and according value of the maxNumberOfElements of the DataPrototype referenced by SwAxisGrouped.swCalprmRef.arParameter **at the time when the contract phase generation is executed.**⌋()

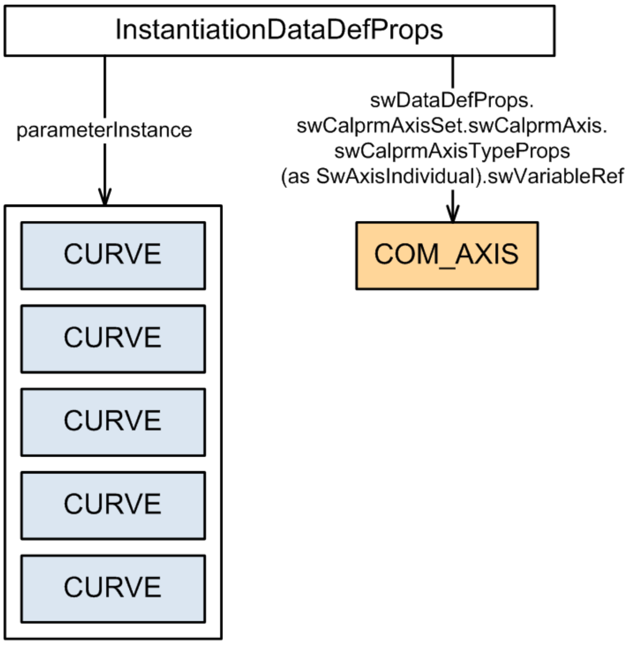


Figure 5.50: Each MAP in the array of CURVEs uses the identical COM_AXIS

[TPS_SWCT_01686] Specification of a single group axis for an array of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 ⌈For DataPrototypes typed by an array of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 the applied InstantiationDataDefProps or ParameterAccess may reference a DataPrototype typed by a ApplicationPrimitiveDataTypes of category COM_AXIS or RES_AXIS with the means of SwAxisGrouped.swCalprmRef.arParameter.

This expresses the semantic that each element in the CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 array uses the identical COM_AXIS or RES_AXIS for the specific SwAxisGrouped.swAxisIndex.⌋()

5.4.7 Specifying Data Dependencies

SwDataDependency allows dependent data elements to be specified. For example, other ParameterDataPrototypes can be combined into one ParameterDataPrototype whose consistent value is automatically derived by the measurement and calibration system. Upon adjusting one of the parameters, the dependent parameter is then also automatically adjusted according to the chosen formula.

Consider for example a rectangular triangle with a hypotenuse of length 1, where the length of the other sides are the parameters A and B. When adjusting parameter A, the parameter B has to be adjusted accordingly to $B = \sqrt{(1 - A * A)}$. Also, other parameters might depend on B, e.g. $B\_AREA = B * B$ or $TRIANGULAR\_AREA = (A * B)/2$. This example is shown in listing 5.14.

A dependent parameter should not be adjustable by itself. The only way to influence its value is through the adjustment of a parameter it depends on.

```
<PER-INSTANCE-PARAMETERS>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>A</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The independent Parameter</L-2>
    </DESC>
    <CATEGORY>VALUE</CATEGORY>
  </PARAMETER-DATA-PROTOTYPE>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>B</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The dependent Parameter</L-2>
    </DESC>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-DATA-DEPENDENCY>
            <SW-DATA-DEPENDENCY-FORMULA>SQRT( X1 * X1)</SW-DATA-DEPENDENCY-
               FORMULA>
            <SW-DATA-DEPENDENCY-ARGS>
              <AR-PARAMETER>
                <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">/
                   DataDependency/foo/bar/A</LOCAL-PARAMETER-REF>
              </AR-PARAMETER>
            </SW-DATA-DEPENDENCY-ARGS>
          </SW-DATA-DEPENDENCY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
  </PARAMETER-DATA-PROTOTYPE>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>B_AREA</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The dependent Parameter</L-2>
    </DESC>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-DATA-DEPENDENCY>
            <SW-DATA-DEPENDENCY-FORMULA>X1 * X1</SW-DATA-DEPENDENCY-FORMULA
               >
            <SW-DATA-DEPENDENCY-ARGS>
              <AR-PARAMETER>
                <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">/
                   DataDependency/foo/bar/B</LOCAL-PARAMETER-REF>
              </AR-PARAMETER>
```

```
                </SW-DATA-DEPENDENCY-ARGS>
              </SW-DATA-DEPENDENCY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
      </PARAMETER-DATA-PROTOTYPE>
      <PARAMETER-DATA-PROTOTYPE>
        <SHORT-NAME>TRIANGULAR_AREA</SHORT-NAME>
        <DESC>
          <L-2 L="DE">The dependent Parameter</L-2>
        </DESC>
        <SW-DATA-DEF-PROPS>
          <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
              <SW-DATA-DEPENDENCY>
                <SW-DATA-DEPENDENCY-FORMULA>(X1 * X2) / 2</SW-DATA-DEPENDENCY-
                   FORMULA>
                <SW-DATA-DEPENDENCY-ARGS>
                  <AR-PARAMETER>
                    <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">/
                       DataDependency/foo/bar/A</LOCAL-PARAMETER-REF>
                  </AR-PARAMETER>
                  <AR-PARAMETER>
                    <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">/
                       DataDependency/foo/bar/B</LOCAL-PARAMETER-REF>
                  </AR-PARAMETER>
                </SW-DATA-DEPENDENCY-ARGS>
              </SW-DATA-DEPENDENCY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
      </PARAMETER-DATA-PROTOTYPE>
    </PER-INSTANCE-PARAMETERS>
```

Listing 5.14: Data Dependency

Class	**SwDataDependency**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This element describes the interdependencies of data objects, e.g. variables and parameters. Use cases: • Calculate the value of a calibration parameter (by the MCD system) from the value(s) of other calibration parameters. • Virtual data - that means the data object is not directly in the ecu and this property describes how the "virtual variable" can be computed from the real ones (by the MCD system).			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swDataDependency			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swData Dependency Args	SwDataDependency Args	0..1	aggr	Specifies the arguments used in the data dependency. Note that this is 0..1 since the aggregated class is a container (atpMixed). **Tags:**xml.sequenceOffset=40

▽

△

Class	SwDataDependency			
swDataDependencyFormula	CompuGenericMath	0..1	aggr	This element describes the formula with which the dependencies between the participating objects are defined. **Tags:**xml.sequenceOffset=30

Table 5.59: SwDataDependency

Class	<<atpMixed>> **SwDataDependencyArgs**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This element specifies the elements used in a SwDataDependency.			
Base	*ARObject*			
Aggregated by	SwDataDependency.swDataDependencyArgs			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swCalprmRef	SwCalprmRefProxy	0..1	aggr	Specifies a calibration parameter as an input argument to the dependency. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=60 xml.typeElement=false xml.typeWrapperElement=false
swVariable	SwVariableRefProxy	0..1	aggr	Specifies a variable as an input argument to the dependency. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=70 xml.typeElement=false xml.typeWrapperElement=false

Table 5.60: SwDataDependencyArgs

5.4.8 Precedence of data properties with respect to data elements, axis elements, computation methods, units

There are similar attributes defined in `SwDataDefProps` as well as in `SwCalprmAxis` as well as in `CompuMethod`. Therefore, we need to define which attribute value wins in the overall process from SWC-Description to MC-Support to ASAM-A2L.

[TPS_SWCT_01496] General precedence rule for attributes of `SwDataDefProps`
⌈The general precedence rule is that

- `SwDataDefProps` wins over `valueAxisDataType` (exception: `compuMethod` and `unit`).
- `SwDataDefProps` wins over `compuMethod`.
- `SwDataDefProps` wins over `swCalprmAxisSet`.

- SwDataDefProps.swCalprmAxisSet wins over swCalprmAxisSet.swCalprmAxis.swCalprmAxisTypeProps.compuMethod or SwAxisIndividual.inputVariableType.
- SwAxisIndividual.inputVariableType wins over SwAxisIndividual.compuMethod, SwAxisIndividual.unit, but **not** over SwAxisIndividual.dataConstr.

⌋()

Figure 5.51 illustrates the fact that some attributes in SwDataDefProps can also be expressed in sub-elements respectively in referenced elements.

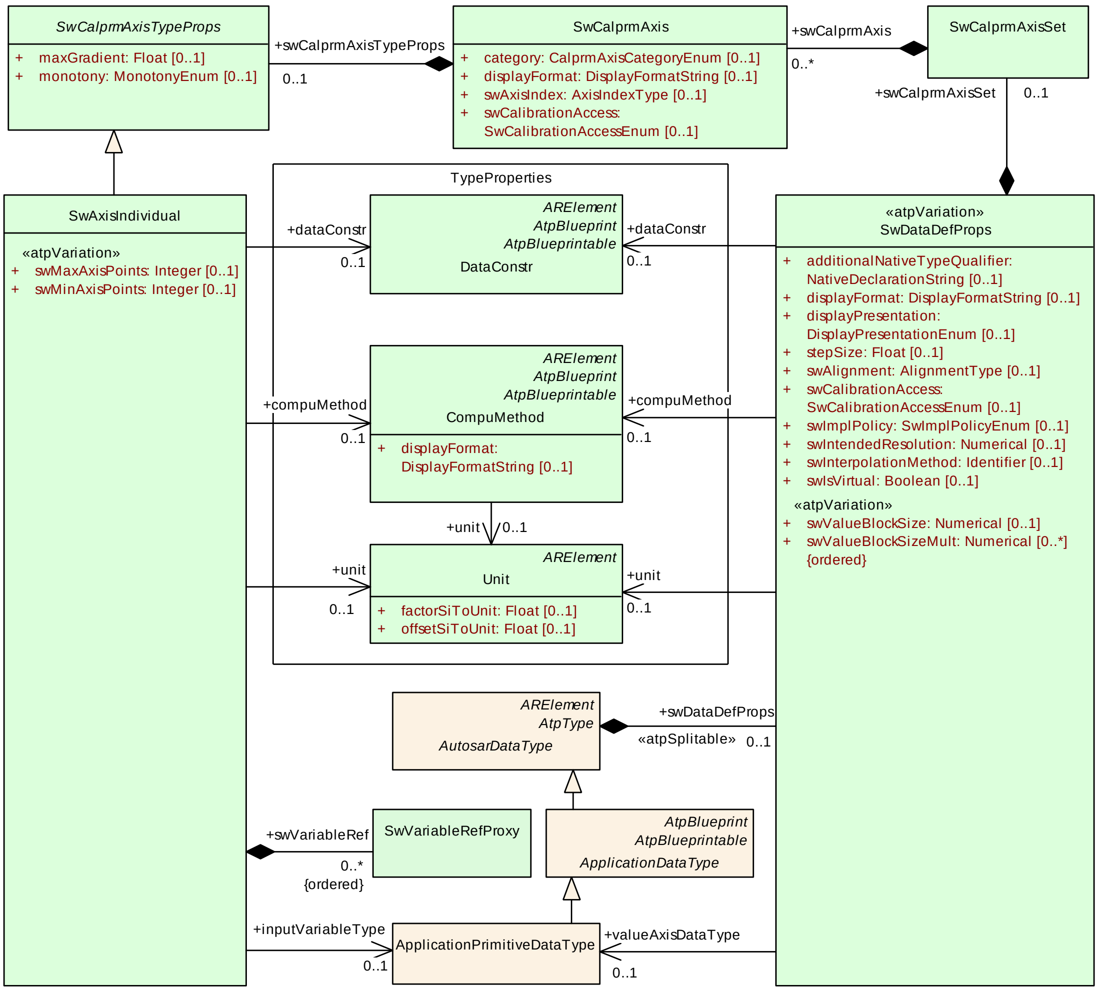


Figure 5.51: Various Attributes in the Context of SwDataDefProps

The following examples illustrate particular cases (the highest precedence comes first):

- **[TPS_SWCT_01497] Precedence of the `unit` of value axis** ⌈For the usage of `unit` of value axis the following precedence rule is defined:
  - SwDataDefProps.valueAxisDataType.swDataDefProps.unit

- SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod.unit
- SwDataDefProps.unit
- SwDataDefProps.compuMethod.unit

⌋()

[constr_2550] Units of value axis shall be consistent ⌈The units specified in the context of value axis shall be the same, even if there is a precedence rule.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

In particular, [constr_2550] reflects the fact that a Unit may be specified in different phases of the development process but finally need to be consistent.

- **[TPS_SWCT_01498] Precedence of the DataConstr of value axis** ⌈For the usage of DataConstr of value axis the following precedence rule is defined:

  - SwDataDefProps.dataConstr
  - SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr

  ⌋()

  [constr_2548] Data constraint of value axis shall match ⌈The values compliant to SwDataDefProps.dataConstr shall also be compliant to SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr.

  In other words SwDataDefProps.dataConstr win over but are not allowed to relax SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr but are not allowed.

  This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

- **[TPS_SWCT_01499] Precedence of the CompuMethod of value axis** ⌈For the usage of CompuMethod of value axis the following precedence rule is defined:

  - SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod
  - SwDataDefProps.compuMethod

  ⌋()

- **[TPS_SWCT_01500] Precedence of the display format of value axis** ⌈For the usage of display format of value axis the following precedence rule is defined:

  - SwDataDefProps.displayFormat

  - SwDataDefProps.valueAxisDataType.swDataDefProps.displayFormat
  - SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod.displayFormat
  - SwDataDefProps.compuMethod.displayFormat

  ⌋()

  Note that this deviates from the general rule since displayFormat is not an essential property. The last item in the list above is the consequence of the fact that if there is a valueAxisDataType it supersedes the compuMethod

- **[TPS_SWCT_01501] Precedence of the calibration access of value axis** ⌈For the usage of calibration access of value axis the following precedence rule is defined:
  - SwDataDefProps.swCalibrationAccess
  - SwDataDefProps.valueAxisDataType.swDataDefProps.swCalibrationAccess

  ⌋()

  Note that this deviates from the general rule since swCalibrationAccess is not such an essential property.

- **[TPS_SWCT_01502] Precedence of the Unit of the input axis** ⌈For the usage of Unit of the input axis the following precedence rule is defined:
  - SwAxisIndividual.unit
  - SwAxisIndividual.compuMethod.unit
  - SwAxisIndividual.inputVariableType.swDataDefProps.unit
  - SwAxisIndividual.swVariableRef.autosarVariable.autosarVariable.type.swDataDefProps.compuMethod.unit
  - SwAxisIndividual.swVariableRef.autosarVariable.autosarVariable.type.swDataDefProps.unit

  ⌋()

  [constr_2549] Units of input axis shall be consistent ⌈The units specified in the context of an input axis shall be compatible, even if there is a precedence rule.

  This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

  [constr_2549] reflects the fact that unit may be specified in different phases of the development process but finally need to be consistent.

- **[TPS_SWCT_01503] Precedence of the DataConstr of the input axis** ⌈For the usage of DataConstr of the input axis the following precedence rule is defined:
  - SwAxisIndividual.dataConstr
  - SwAxisIndividual.inputVariableType.swDataDefProps.dataConstr
  - SwAxisIndividual.swVariableRef.type.swDataDefProps.dataConstr

  ⌋*()*

  Please note that the attribute SwAxisIndividual.inputVariableType.swDataDefProps.dataConstr represents the input value, not the axis itself. For this reason, there is no specific constraint defined that the dataConstr needs to fulfill.

- **[TPS_SWCT_01504] Precedence of the display format of the input axis** ⌈For the usage of display format of the input axis the following precedence rule is defined:
  - SwCalprmAxis.displayFormat
  - SwCalprmAxis.swCalprmAxisTypeProps.compuMethod.displayFormat
  - SwCalprmAxis.swCalprmAxisTypeProps.inputVariableType.swDataDefProps.displayFormat
  - SwCalprmAxis.swCalprmAxisTypeProps.inputVariableType.swDataDefProps.compuMethod.displayFormat
  - SwCalprmAxis.swCalprmAxisTypeProps.swVariableRef.type.swDataDefProps.displayFormat
  - SwCalprmAxis.swCalprmAxisTypeProps.swVariableRef.type.swDataDefProps.compuMethod.displayFormat

  ⌋*()*

  Please note that SwAxisIndividual.inputVariableType.swDataDefProps.dataConstr represent the input value and not the axis itself. For this reason there is no specific constraint that displayFormat needs to match.

- **[TPS_SWCT_01505] Precedence of calibration access along structural hierarchies in complex types** ⌈For the usage of calibration access along structural hierarchies in complex types the precedence rule is defined in table 5.61.⌋*()*

  The interpretation of table 5.61 is that it lists possible combinations of values of SwCalibrationAccessEnum for outer and inner elements of a complex data type and the (in the column "result") indicates value of SwCalibrationAccessEnum applicable for this specific combination.

- **[TPS_SWCT_01506] Precedence of the calibration access of input axis** ⌈For the usage of calibration access of input axis the following precedence rule is defined:
  - SwDataDefProps.swCalibrationAccess
  - SwCalprmAxis.swCalibrationAccess

  ⌋*()*

  Note that the swCalibrationAccess defined on a Compound Primitive Data Type (see [TPS_SWCT_01179]) reflects the entire curve or map.

  Therefore, if the entire curve or map cannot be accessed by the measurement calibration diagnostic system (MCD-System), the axis can also not be accessed. On the other hand it might be that access is granted for the value axis only but not for the axis points.

outer	inner	result
notAccessible	*	notAccessible
readOnly	readOnly	readOnly
readOnly	readWrite	readOnly
readOnly	notAccessible	notAccessible
readWrite	notAccessible	notAccessible
readWrite	readOnly	readOnly
readWrite	readWrite	readWrite

Table 5.61: Precedence of swCalibrationAccess along structural hierarchies

5.5 Elements used in Properties of Data Definitions

This section describes further elements which are attached to SwDataDefProps via associations.

5.5.1 Computation Methods

[TPS_SWCT_01276] Computation methods ⌈An important part of semantics is the specification of a so-called computation method which specifies the conversion between the physical and the internal representation of data. This usually makes sense only for primitive data types.⌋*()*

An ApplicationCompositeDataType cannot be given a particular semantic meaning as a whole but it is obviously possible to specify the semantics of all or a part of the contained elements, i.e. the ApplicationPrimitiveDataTypes.

Class	CompuMethod			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:**atp.recommendedPackage=CompuMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compuInternal ToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:**xml.sequenceOffset=80
compuPhysTo Internal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:**xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:**xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:**xml.sequenceOffset=30

Table 5.62: CompuMethod

This meta-class CompuMethod was actually taken from the *ASAM* standard's *harmonized data objects*. This is also indicated by the green color of the meta-classes in the diagram.

Some categorys of CompuMethod cannot be successfully converted to A2L [28] because A2L does not provide an equivalent semantics that comes close to the respective AUTOSAR semantics.

A prominent example for such a case is a CompuMethod of category BITFIELD_TEXTTABLE.

[constr_1142] category of CompuMethod shall not be extended ⌈In contrast to the general rule that category can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute category of meta-class CompuMethod **at any time in the workflow**.⌋*()*

[TPS_SWCT_01277] Computation methods are used for the conversion of *internal* values into their *physical* representation and vice versa ⌈CompuMethods are used for the conversion of *internal* values into their *physical* representation and vice versa. The direction of the conversion depends on the origin of the value to be converted:

- If the value is provided by the ECU, then the conversion direction is from internal to physical.

- If a physical value is provided by the tester, it is converted to internal values before being sent to the ECU

⌋()

[TPS_SWCT_01548] Limits of a CompuMethod ⌈In case CompuScale.lowerLimit and CompuScale.upperLimit are used to constrain the applicable range of the conversion of a CompuMethod, they logically represent the limiting values **before** the conversion is applied.⌋()

In other words, the limits are applied on the source end of the conversion rather than to the result that comes out at the other end of the conversion.

This is obviously a lot safer than the opposite approach where a given physical/internal value would first be converted to its internal/physical equivalent and then, after the conversion is finished there would be (as a second step) the obligation to check whether the result of the conversion is actually valid in terms of the applicable limits.

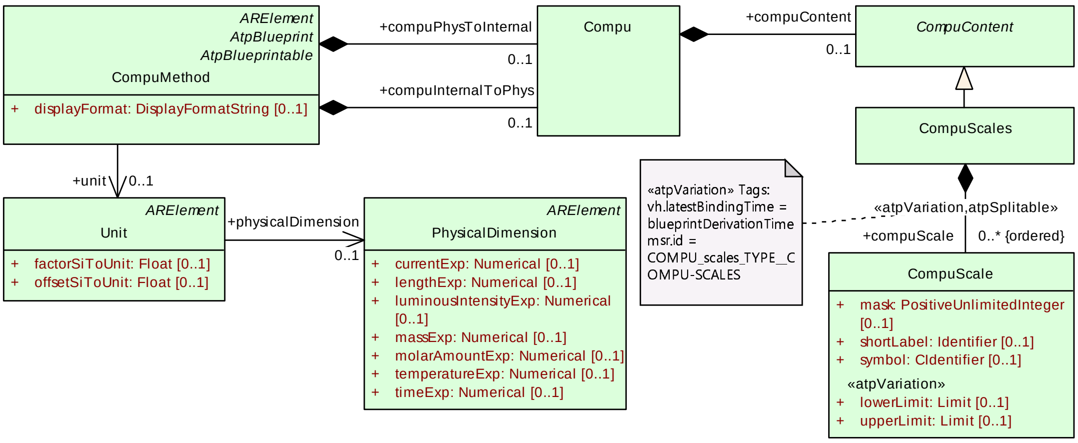


Figure 5.52: A CompuMethod and its attributes define data semantics

[TPS_SWCT_01278] CompuMethods can also be used to assign symbolic names to internal values ⌈CompuMethods can also be used to assign symbolic names to internal values (like an enumeration in C) or to ranges of internal values or to single bits (like a bitfield in C).

This is also considered as a conversion between internal numbers and a semantical representation. Some examples are given below.⌋()

Actually, the preferred conversion direction depends on the use case.

In the following, the internal-to-physical conversion direction is used as the default. Usually a CompuMethod is defined for one conversion direction only even if it is used in both directions.

For simple functions like identical (1:1 conversion) or linear functions this is sufficient because the inverse function can be derived quite easily from the defined function. In this case also the limits for the reverse direction can be gained by applying the forward function to the forward limits.

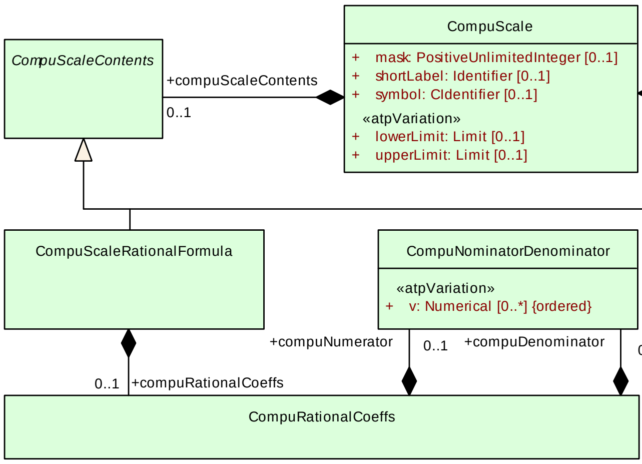


Figure 5.53: A CompuScale and its attributes define data semantics

For more complex functions (e.g. rational functions) it is usually not possible to compute the inverse function automatically. More seriously, the inversion yields ambiguous results if the function is not monotonic. To deal with such possible ambiguities directly, an inverse value can be provided explicitly for the function or for each of its parts respectively.

[constr_1022] Limits shall be defined for each direction of CompuMethod ⌈In case that both domains are specified in the CompuMethod both shall have explicitly defined limits **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01280] CompuMethod applied to values outside of its limits ⌈If a CompuMethod is applied to values outside its limits, it is up to the MCD-tool (Measurement, Calibration, Diagnostic tool) to indicate this to the user. In this case the CompuMethod shall not be applied at all.⌋*()*

[constr_1175] Depending on its category, CompuMethod shall refer to a unit ⌈As a CompuMethod specifies the conversion between the physical world and the numerical values, it shall refer to a unit unless the CompuMethod's category is one of TEXTTABLE, BITFIELD_TEXTTABLE, or IDENTICAL.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1175] does *not* imply that CompuMethods where the category is one of TEXTTABLE, BITFIELD_TEXTTABLE, or IDENTICAL are not *allowed* to refer to a unit. They may still refer to a unit, but according to [constr_1175] this relation is not *mandated*.

A further implication is that the unit itself may not have a dimension, i.e. all exponents of SI units are 0.

Figure 5.52 sketches a conceptual overview of CompuMethod. It consists of the following attributes:

- **[TPS_SWCT_01281] Unit associated with a PhysicalDimension** ⌈A unit (described in next section) can be associated with a PhysicalDimension.⌋*()*

  Note that quantities like “%” are not derived from SI units. However, they have a meaning in the physical world and need to be represented in form of data types. Therefore, a CompuMethod also applies in those cases.

- **[TPS_SWCT_01430] Conversion specification from internal to physical values as well as the reverse conversion** ⌈A conversion specification from internal to physical values, as well as the reverse conversion. Both of them in turn consist of an abstract CompuContent. Derived classes allow the specification of a conversion formula in two different ways.⌋*()*

  [constr_1024] Stepwise definition of CompuMethods ⌈In a bound model, the intervals (i.e. determined by attributes CompuScale.lowerLimit and CompuScale.upperLimit) defined by CompuScales used in the context of a given CompuMethod of all values of category except BITFIELD_TEXTTABLE shall **not** overlap.

  For CompuMethods of category BITFIELD_TEXTTABLE, the combination of the interval created by attributes CompuScale.upperLimit, CompuScale.lowerLimit and CompuScale.mask shall be unique in the context of the enclosing CompuMethod.

  This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

  The possible values of CompuMethod.category are listed in Table 5.73.

  [TPS_SWCT_01667] Avoidance of overlapping of directly adjacent intervals within CompuMethods ⌈Intervals of a given CompuMethod may be **located directly adjacent** to each other.

  This means that the upperLimit of one CompuScale has the same numerical value as the lowerLimit of another CompuScale defined within the context of the CompuMethod.

  In this case, it is necessary to properly set the attribute CompuScale.lowerLimit.intervalType or CompuScale.upperLimit.intervalType in order to avoid an overlapping.

  Specifically, one of the interval boundaries shall be set to intervalType.open in order to avoid an overlapping.⌋*()*

- **[TPS_SWCT_01282] Number of intervals in which a given conversion applies** ⌈CompuScales is a number of intervals (called CompuScale) within which a certain conversion applies. The respective interval is given in terms of upper and lower limit.

  Within each CompuScale we have the abstract CompuScaleContents. To deal with possible ambiguities directly, an inverse value can be provided explicitly for that particular scale (compuInverseValue).⌋*()*

Please note that limits are explained in more detail in chapter 5.2.4.1.

- As the diagram shows, CompuScaleContents is an abstract meta-class. A number of derived meta-classes allow the specification of a conversion formula in a variety of ways, including:
  - mapping the whole interval to a constant (CompuConst)
  - providing rational coefficients of the conversion formula (CompuRationalCoeffs)
- **[TPS_SWCT_01283] Rational function** ⌈The rational function is specified as rational coefficients for the numerator (compuNumerator) and the denominator (compuDenominator). CompuNominatorDenominator can have as many *V* elements as needed for the rational function.

  The sequence of the values *V* carries the information for the exponents, that means the first *V* is the coefficient for x0, the second *V* is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented.⌋*()*

  [constr_1025] Avoid division by zero in rational formula ⌈The rational formula shall not yield any division by zero **at any time in the workflow**.⌋*()*

[TPS_SWCT_01284] CompuScale might require a representation in the generated RTE C code ⌈A CompuScale might require a representation in the generated RTE C code. For this purpose it is necessary to identify a property that controls how to symbol used for the CompuScale in the C code is created.

The symbol itself can be created out of different sources according to a standardized precedence schema.⌋*()*

[TPS_SWCT_01569] Definition of CompuScale Code Symbolic Name ⌈In C code, a CompuScale is represented by an identifier that is, as far as AUTOSAR modeling is concerned, called a CompuScale Code Symbolic Name.

The CompuScale Code Symbolic Name may be taken from CompuScale.symbol, CompuConstTextContent.vt, or CompuScale.shortLabel. The details are explained in [TPS_SWCT_01431].⌋*()*

[TPS_SWCT_01431] Finding the symbol for the representation of a CompuScale with a point-range in C code ⌈In general, the value of the attributes symbol, vt, and shortLabel can be taken as the source for naming the symbol that represents the CompuScale in the C code.

The following rule applies (lower values indicate higher priority) for all CompuScales with a point-range:

1. Take the value of symbol if this attribute exists.
2. Take the value of vt if it makes a valid C identifier.
3. Take the value of shortLabel if it exists.

Fail if none of the possible options apply.

⌋()

[TPS_SWCT_01695] Relation between ValueSpecification and the definition of CompuScales ⌈In order to find a match between the content of a ValueSpecification and a CompuScale the content of the ValueSpecification shall be checked against the CompuScale Value Symbolic Names according to [TPS_SWCT_01696].

If no matching CompuScale Value Symbolic Names can be found then the ValueSpecification shall be considered unusable in the context of the CompuMethod that is subject to [constr_1146].⌋()

[TPS_SWCT_01696] CompuScale Value Symbolic Name ⌈The value of the CompuScale Value Symbolic Name of a given CompuScale shall be obtained by taking the values of the following attributes according to the following priority (lower values indicate higher priority):

1. Take the value of symbol if this attribute exists.
2. Take the value of vt if this attribute exists.
3. Take the value of shortLabel if it exists.

⌋()

Just to be sure, the (obvious) difference between a CompuScale Value Symbolic Name and a CompuScale Code Symbolic Name is that the former is not required to pass as a valid C identifier.

[constr_1434] CompuScales shall not have identical CompuScale Value Symbolic Names ⌈In a CompuMethod that is subject to [constr_1146], no two CompuScales shall have identical CompuScale Value Symbolic Names (according to [TPS_SWCT_01696]) **at the time when the contract phase generation is executed**.⌋()

[constr_1146] Applicability of a symbol for a CompuScale in C code ⌈The symbol attribute shall only be provided for CompuScales where the category of the enclosing CompuMethod is one of the following:

- TEXTTABLE
- SCALE_LINEAR_AND_TEXTTABLE
- SCALE_RATIONAL_AND_TEXTTABLE
- BITFIELD_TEXTTABLE

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Class	**Compu**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express one particular computation.			
Base	*ARObject*			
Aggregated by	CompuMethod.compuInternalToPhys, CompuMethod.compuPhysToInternal			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
compuContent	CompuContent	0..1	aggr	This specifies the details of the computation. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
compuDefault Value	CompuConst	0..1	aggr	This property can be used to specify an output value for a conversion formula, if the value to be converted lies outside the plausibility limit. Although this is possible for all conversion formulae, it is especially valid for variables with tabular conversion formulae. **Tags:**xml.sequenceOffset=70

Table 5.63: Compu

Class	***CompuContent*** (abstract)			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This abstract meta-class represents the various definition means of a computation method.			
Base	*ARObject*			
Subclasses	CompuScales			
Aggregated by	Compu.compuContent			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.64: CompuContent

Class	**CompuScale**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to specify one segment of a segmented computation method.			
Base	*ARObject*			
Aggregated by	CompuScales.compuScale			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
compuInverse Value	CompuConst	0..1	aggr	This is the inverse value of the constraint. This supports the case that the scale is not reversible per se. **Tags:**xml.sequenceOffset=60
compuScale Contents	CompuScaleContents	0..1	aggr	This represents the computation details of the scale. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=70 xml.typeElement=false xml.typeWrapperElement=false

▽

△

Class	CompuScale			
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:**xml.sequenceOffset=30
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
mask	PositiveUnlimitedInteger	0..1	attr	In difference to all the other computational methods every COMPU-SCALE will be applied including the bit MASK. Therefore it is allowed for this type of COMPU-METHOD, that COMPU-SCALES overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the COMPU-SCALE elements. **Tags:**xml.sequenceOffset=35
shortLabel	Identifier	0..1	attr	This element specifies a short name for the particular scale. The name can for example be used to derive a programming language identifier. **Tags:**xml.sequenceOffset=20
symbol	CIdentifier	0..1	attr	The symbol, if provided, is used by code generators to get a C identifier for the CompuScale. The name will be used as is for the code generation, therefore it needs to be unique within the generation context. **Tags:**xml.sequenceOffset=25
upperLimit	Limit	0..1	attr	This specifies the upper limit of a of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50

Table 5.65: CompuScale

Class	CompuScales			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to stepwise express a computation method.			
Base	*ARObject*, *CompuContent*			
Aggregated by	Compu.compuContent			
Attribute	Type	Mult.	Kind	Note

▽

△

Class	CompuScales			
compuScale (ordered)	CompuScale	*	aggr	This represents one scale within the compu method. Note that it contains a Variationpoint in order to support blueprints of enumerations. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=compuScale, compuScale.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false

Table 5.66: CompuScales

Class	*CompuScaleContents* (abstract)			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This abstract meta-class represents the content of one particular scale.			
Base	*ARObject*			
Subclasses	CompuScaleConstantContents, CompuScaleRationalFormula			
Aggregated by	CompuScale.compuScaleContents			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 5.67: CompuScaleContents

Class	CompuRationalCoeffs			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express a rational function by specifying the coefficients of nominator and denominator.			
Base	*ARObject*			
Aggregated by	CompuScaleRationalFormula.compuRationalCoeffs			
Attribute	Type	Mult.	Kind	Note
compuDenominator	CompuNominatorDenominator	0..1	aggr	This is the denominator of the expression. **Tags:**xml.sequenceOffset=30
compuNumerator	CompuNominatorDenominator	0..1	aggr	This is the numerator of the rational expression. **Tags:**xml.sequenceOffset=20

Table 5.68: CompuRationalCoeffs

[constr_1913] Existence of attribute `CompuRationalCoeffs.compuDenominator` ⌈For each `CompuRationalCoeffs`, the attribute `compuDenominator` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

[constr_1914] Existence of attribute `CompuRationalCoeffs.compuNumerator` ⌈For each `CompuRationalCoeffs`, the attribute `compuNumerator` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

Class	CompuConst			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the value of a computation method scale is constant.			
Base	*ARObject*			
Aggregated by	Compu.compuDefaultValue, CompuScale.compuInverseValue, CompuScaleConstantContents.compuConst			
Attribute	Type	Mult.	Kind	Note
compuConstContentType	CompuConstContent	0..1	aggr	This is the actual content of the constant compu method scale. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=10 xml.typeElement=false xml.typeWrapperElement=false

Table 5.69: CompuConst

Class	CompuScaleRationalFormula			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the computation in this scale is represented as rational term.			
Base	*ARObject*, *CompuScaleContents*			
Aggregated by	CompuScale.compuScaleContents			
Attribute	Type	Mult.	Kind	Note
compuRationalCoeffs	CompuRationalCoeffs	0..1	aggr	This specifies the coefficients of the rational formula. **Tags:**xml.sequenceOffset=110

Table 5.70: CompuScaleRationalFormula

Class	CompuScaleConstantContents			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that a particular scale of the computation method is constant.			
Base	*ARObject*, *CompuScaleContents*			
Aggregated by	CompuScale.compuScaleContents			
Attribute	Type	Mult.	Kind	Note
compuConst	CompuConst	0..1	aggr	This represents the fact that the scale is a constant. The use case is mainly a non interpolated scale. It is a simplification of the fact that a constant scale can also be expressed as rational function of order 0. **Tags:**xml.sequenceOffset=90

Table 5.71: CompuScaleConstantContents

Class	CompuNominatorDenominator			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This class represents the ability to express a polynomial either as Nominator or as Denominator.			
Base	*ARObject*			
Aggregated by	CompuRationalCoeffs.compuDenominator, CompuRationalCoeffs.compuNumerator			
Attribute	Type	Mult.	Kind	Note

▽

△

Class	**CompuNominatorDenominator**			
v (ordered)	Numerical	*	attr	this is the list of polynomial factors. Note that the first vf represents the power=0. The polynomial is v[0] * xˆ0 + v[1] * xˆ1 ... **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 5.72: CompuNominatorDenominator

Please note that the values of coefficients within a rational formula are **not restricted** to integer values. It is possible to use floating-point values as well.

The values of exponents **cannot be set arbitrarily** but are implicitly defined by the appearance of coefficients in `CompuNominatorDenominator.v`, i.e. the first value in the ordered list of `CompuNominatorDenominator.v` represents the exponent 0, the second `CompuNominatorDenominator.v` represents the exponent 1, and so on.

5.5.1.1 Category Values in the context of a CompuMethod

For a detailed description of `CompuMethod`s, please refer to the *ASAM MCD 2 Harmonized Data Objects* [29].

Table 5.73 contains a definition of possible values for the attribute `category`.

ASAM Category	*Meaning*	*Specific properties*
IDENTICAL	This `CompuMethod` just hands over the internal value with an optional unit.	Only the base elements are allowed and `unit`, `physConstrs` and `internalConstrs` are optional. This is the simplest type of a `CompuMethod`.
LINEAR	A linear conversion can be performed in two steps: The internal value is multiplied with a factor; after that, an offset is added to the result of the multiplication.	Exactly one `CompuScale`, with two `v` in `compuNumerator` and one `v` in `compuDenominator`.
SCALE_LINEAR	Used for a piecewise linear conversion.	More than one `compuScale` can be defined. Additionally there have to be the `upperLimit` and `lowerLimit` elements which define the region of validity for the linear function. The boundaries of the regions shall not overlap.
SCALE_LINEAR_AND_TEXTTABLE	Used for piecewise definition of one or more linear and several texttable scales.	Properties depend on the used scale function. For details see definition of `SCALE_LINEAR` and `TEXTTABLE`. The scales shall each provide `lowerLimit` and `upperLimit` definitions.

▽

△

ASAM Category	*Meaning*	*Specific properties*
RAT_FUNC	The rational function type is similar to the linear type without the restrictions for the compuNumerators and compuDenominators.	It can have as many v elements as needed for the rational function. The sequence of the values *v* carries the information for the exponents, that means the first *v* is the coefficient for x0, the second *v* is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented. A rational function is only applicable for conversions in the direction that it is defined for, i.e. the automatic calculation of the inverse function is not supported by the MCD system.
SCALE_RAT_FUNC	Used for piecewise defined rational conversion.	
SCALE_RATIONAL_AND_TEXTTABLE	Used for piecewise definition of one rational and several text-table scales.	Properties depend on the used scale function. For details see definition of SCALE_RAT_FUNC and TEXTTABLE. The scales shall each provide lowerLimit and upperLimit definitions.
TEXTTABLE	The type TEXTTABLE is used for transformations of the internal value into textual elements.	The result is placed in the vt member of CompuConst. The compuDefaultValue is optional. If the reverse calculation is needed then for each scale the compuInverseValue can be used to define the reverse calculation result. If no inverse value is explicitly defined then the smallest possible value of the scale will be used as result of the reverse calculation. [constr_1134] applies!
TAB_NOINTP	Similar to TEXTTABLE, but for numerical values.	The values per scale are defined in CompuConst.
BITFIELD_TEXTTABLE	Similar to TEXTTABLE but for bit fields.	BITFIELD_TEXTTABLE is derived from TEXTTABLE. The main difference is that TEXTTABLE results to a single value while BITFIELD_TEXTTABLE results to a concatenated value set. In difference to all the other computational methods **every** CompuScale will be applied including the bit mask specified in mask. Therefore it is allowed for this type of CompuMethod, that CompuScales overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the CompuScale elements. [constr_1135] applies!

Table 5.73: ASAM compuMethod

[constr_1134] Allowed structure of TEXTTABLE ⌈The existence of physConstrs is not allowed and compuInternalToPhys shall exist with compuScales consisting of upperLimit and lowerLimit **at any time in the workflow**.⌋*()*

[constr_1135] Limit of vt in BITFIELD_TEXTTABLE ⌈The separator for splitting the string representing the value is “|” and is therefore forbidden to appear in vt **at any time in the workflow**.⌋*()*

5.5.1.2 Applicability of Attributes in the context of a CompuMethod

This section summarizes the applicability of CompuMethod in terms of which attributes of CompuMethod and related meta-classes (e.g. CompuScale, CompuConst) shall be used depending on the nature of the CompuMethod, expressed by means of the value of attribute category.

[constr_1375] Existence of attributes of CompuMethod and related meta-classes depending on the value of the category ⌈

	Attribute Existence per Category									
Attributes of CompuMethod	IDENTICAL	LINEAR	SCALE_LINEAR	RAT_FUNC	SCALE_RAT_FUNC	TEXTTABLE	BITFIELD_TEXTTABLE	SCALE_LINEAR_AND_TEXTTABLE	SCALE_RATIONAL_AND_TEXTTABLE	TAB_NOINTP
compuInternalToPhys	N/A	D(1)	D(1)	D(2)	D(2)	D	D	D(8)	D(2)	D
compuPhysToInternal	N/A	D(1)	D(1)	D(2)	D(2)	N/A	N/A	N/A	D(2,3)	N/A
Attributes of meta-classes related to CompuMethod										
compuDefaultValue	N/A	O(6)	O(6)	O(6)	O(6)	O(6)	O(6)	O(6)	O(6)	O(6)
CompuScale	N/A	D/1..1	D/1..n	D/1..1	D/1..n	D/1..n	D/1..n	D/1..n	D/1..n	D/1..n
CompuScale.compuInverseValue	N/A	N/A	N/A	O(2)	O(2)	O(5)	N/A	O(2,5)	O(2,5)	O(5)
CompuScale.lowerLimit	N/A	O	D	D(4)	D(4)	D	D	D	D(4)	D
CompuScale.mask	N/A	N/A	N/A	N/A	N/A	N/A	D	N/A	N/A	N/A
CompuScale.shortLabel	N/A	N/A	N/A	N/A	N/A	O(7)	O(7)	O(7)	O(7)	N/A
CompuScale.symbol	N/A	N/A	N/A	N/A	N/A	O(7)	O(7)	O(7)	O(7)	N/A
CompuScale.upperLimit	N/A	O	D	D(4)	D(4)	D	D	D	D(4)	D
CompuConst	N/A	N/A	N/A	N/A	N/A	D/vt	D/vt	D/vt	D/vt	D/vt or vf
CompuRationalCoeffs	N/A	D	D	D	D	N/A	N/A	D	D	N/A
CompuRationalCoeffs.compuDenominator	N/A	D/1v	D/1v	D	D	N/A	N/A	D/1v	D	N/A
CompuRationalCoeffs.compuNumerator	N/A	D/2v	D/2v	D	D	N/A	N/A	D/2v	D	N/A

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

For clarification, the first two rows of the table contained in [constr_1375] define the applicability of the immediate attributes of meta-class CompuMethod, the remainder of the table then goes into further detail regarding the usage of the attributes of related meta-classes (e.g. CompuScale, CompuConst).

Please note that annotations apply to the individual cell values. These annotations are formulated by means of a numerical value in parentheses, e.g. (1).

The legend for the individual annotations can be found below [constr_1375].

The following legend applies to the cells in the table contained in [constr_1375]:

D **Define** the attribute.

N/A Attribute is **not applicable** for usage in the scope of this element.

O **Optionally** define the attribute.

In addition to the primary cell legend the following annotations apply to the cells in the table contained in [constr_1375]:

(1) In this case **either** compuPhysToInternal **or** compuInternalToPhys shall be defined.

(2) In this case both compuPhysToInternal and compuInternalToPhys shall be defined unless compuInverseValue exists (see [TPS_SWCT_01282]). In other words, if the explicit definition of a compuInverseValue exists then there is no need to define conversions from internal to physical **and** vice versa.

(3) Not applicable for CompuScales where attribute compuScaleContents.compuConst exists.

(4) Limits shall be defined according to [constr_1022].

(5) Restrictions on the structure of the CompuMethod according to [constr_1134] apply.

(6) Specify an output value for a conversion formula if the value to be converted yields outside the plausibility limit (for more information, please refer to the class table of Compu).

(7) Restricted applicability for the attribute CompuScale.symbol, see [constr_1146].

(8) Mandatory for CompuConst; enforced for CompuRationalCoeffs.

5.5.1.3 CompuMethod and AutosarDataType

This chapter clarifies the applicability of CompuMethod for the relevant concrete subclasses of AutosarDataType.

5.5.1.3.1 CompuMethod and ApplicationDataType

For ApplicationDataType, there are (see [constr_1007]) a number values of category that allow for the definition of a ApplicationDataType.swDataDefProps.compuMethod.

[constr_1634] visualizes the allowed combinations of ApplicationDataType.category vs. CompuMethod.category.

The rows of the table contained in [constr_1634] represent values of category for ApplicationDataType that are cleared for the definition of a CompuMethod according to [constr_1007].

The columns of the table contained in [constr_1634] represent values of category for CompuMethod.

[constr_1634] Allowed combinations of ApplicationDataType.category vs. CompuMethod.category ⌈

	IDENTICAL	LINEAR	SCALE_LINEAR	SCALE_LINEAR_AND_TEXTTABLE	RAT_FUNC	SCALE_RATIONAL_AND_TEXTTABLE	TEXTTABLE	TAB_NOINTP	BITFIELD_TEXTTABLE
VALUE	x	x	x	x	x	x	x	x	x
VAL_BLK	x	x	x	x	x	x	x	x	x
BOOLEAN	n/a	n/a	n/a	n/a	n/a	n/a	x	n/a	n/a
CURVE	x	x	x	x	x	x	x	x	x
MAP	x	x	x	x	x	x	x	x	x
CUBOID	x	x	x	x	x	x	x	x	x
CUBE_4	x	x	x	x	x	x	x	x	x
CUBE_5	x	x	x	x	x	x	x	x	x

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

5.5.1.3.2 CompuMethod and ImplementationDataType

For ImplementationDataType, there are (see [constr_1009]) only two values of category that allow for the definition of a ImplementationDataType.swDataDefProps.compuMethod: TEXTTABLE and BITFIELD_TEXTTABLE.

The rows of the table contained in [constr_1158] represent values of category for ImplementationDataType that are cleared for the definition of a CompuMethod according to [constr_1009].

The columns of the table contained in [constr_1158] represent values of category for CompuMethod.

[constr_1158] Applicable categorys for attribute ImplementationDataType.swDataDefProps.compuMethod ⌈

	IDENTICAL	LINEAR	SCALE_LINEAR	SCALE_LINEAR_AND_TEXTTABLE	RAT_FUNC	SCALE_RATIONAL_AND_TEXTTABLE	TEXTTABLE	TAB_NOINTP	BITFIELD_TEXTTABLE
VALUE	n/a	n/a	n/a	n/a	n/a	n/a	x	n/a	x
TYPE_REFERENCE	n/a	n/a	n/a	n/a	n/a	n/a	x	n/a	x

This rule shall be imposed at the time when the contract phase generation is executed.

⌋()

5.5.1.4 Example for Enumeration

The following example illustrates how an enumeration is specified using CompuMethod.

```
<COMPU-METHOD>
  <SHORT-NAME>boolean</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>
```

Listing 5.15: example for enumeration

5.5.1.5 Example for Linear Conversion

The following examples illustrate how a linear conversion is specified using CompuMethod.

$$F_{[km/h]} = 30_{[km/h]} + 2_{[km/h]} * x$$

```
<COMPU-METHOD>
  <SHORT-NAME>linear</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <UNIT-REF DEST="UNIT">kmph</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>30</V>
            <V>2</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Listing 5.16: example for linear CompuMethod

5.5.1.6 Example for Linear Conversion with texttable

The following example illustrates how a linear conversion with a texttable is specified using CompuMethod.

```
<COMPU-METHOD>
  <SHORT-NAME>linearAndTexttable</SHORT-NAME>
  <CATEGORY>SCALE_LINEAR_AND_TEXTTABLE</CATEGORY>
  <UNIT-REF DEST="UNIT">kmph</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">300</UPPER-LIMIT>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>30</V>
            <V>2</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
```

```
      </COMPU-RATIONAL-COEFFS>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">350</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">350</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>SensorError</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">351</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">351</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>SignalNotAvailable</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
  </COMPU-SCALES>
 </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Listing 5.17: example for linear and texttable CompuMethod

5.5.1.7 Example for conversion specified by a rational function

The semantics of rational function is:

$$Internal = \frac{v_0 * phys^0 + v_1 * phys^1 + v_2 * phys^2 + ...}{v_0 * phys^0 + v_1^* phys^1 + v_2 * phys^2 + ...}$$

The following example illustrates a reciprocal conversion.

$$I = \frac{1000}{60 + 2_{[K^{-1}]} * P_{[K]}}$$

```
<COMPU-METHOD>
  <SHORT-NAME>rational</SHORT-NAME>
  <CATEGORY>RAT_FUNC</CATEGORY>
  <UNIT-REF DEST="UNIT">Kelvin</UNIT-REF>
  <COMPU-PHYS-TO-INTERNAL>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-29</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="OPEN">INF</UPPER-LIMIT>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>1000</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>60</V>
            <V>2</V>
          </COMPU-DENOMINATOR>
```

```
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>
```

Listing 5.18: example for rational CompuMethod

5.5.1.8 Example for BITFIELD_TEXTTABLE

The following example shows how a CompuMethod of category BITFIELD_TEXTTABLE can be used to assign a special meaning to each bit of an AutosarDataType of category VALUE:

Bit 0	front left	0(0) = no, 1(1) = yes
Bit 1	front right	0(0) = no, 1(2) = yes
Bit 2	rear left	0(0) = no, 1(4) = yes
Bit 3	rear right	0(0) = no, 1(8) = yes
Bit 4-5	problem	00(00) = flat tire 01(16) = low pressure 10(32) = unbalanced 11(48) = unknown
All Bits	error	11111111 = invalid value

Table 5.74: Example Bitfield

Note that this example is somehow tricky. Bit 6+7 are not used for valid data, but are part of the mask. By this the error can safely be masked out.

Internal: 28

```
28 = 0b0001_1100
Bit    7654 3210
```

Physical:

“problem = low pressure | rear right = yes | rear left = yes | front right = no | front left = no”

```
<COMPU-METHOD>
  <SHORT-NAME>Texttable</SHORT-NAME>
  <CATEGORY>BITFIELD_TEXTTABLE</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <!-- problem -->
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <SYMBOL>problem_flat_tire</SYMBOL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>flat tire</VT>
```

```
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>problem</SHORT-LABEL>
      <SYMBOL>problem_low_pressure</SYMBOL>
      <MASK>0b11110000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00010000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00010000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>low pressure</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>problem</SHORT-LABEL>
      <SYMBOL>problem_unbalanced</SYMBOL>
      <MASK>0b11110000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00100000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00100000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>unbalanced</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>problem</SHORT-LABEL>
      <SYMBOL>problem_unknown</SYMBOL>
      <MASK>0b11110000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00110000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00110000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>unknown</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>problem</SHORT-LABEL>
      <SYMBOL>problem_invalid</SYMBOL>
      <MASK>0b11110000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b11110000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b11110000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>invalid</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- rear right -->
    <COMPU-SCALE>
      <SHORT-LABEL>rearRight</SHORT-LABEL>
      <SYMBOL>rearRight_no</SYMBOL>
      <MASK>0b11001000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>no</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>rearRight</SHORT-LABEL>
      <SYMBOL>rearRight_yes</SYMBOL>
```

```
      <MASK>0b11001000</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00001000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00001000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>yes</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- rear left -->
    <COMPU-SCALE>
      <SHORT-LABEL>rearLeft</SHORT-LABEL>
      <SYMBOL>rearLeft_no</SYMBOL>
      <MASK>0b11000100</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>no</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>rearLeft</SHORT-LABEL>
      <SYMBOL>rearLeft_yes</SYMBOL>
      <MASK>0b11000100</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000100</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000100</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>yes</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- front right -->
    <COMPU-SCALE>
      <SHORT-LABEL>frontRight</SHORT-LABEL>
      <SYMBOL>frontRight_no</SYMBOL>
      <MASK>0b11000010</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>no</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <COMPU-SCALE>
      <SHORT-LABEL>frontRight</SHORT-LABEL>
      <SYMBOL>frontRight_yes</SYMBOL>
      <MASK>0b11000010</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000010</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000010</UPPER-LIMIT>
      <COMPU-CONST>
        <VT>yes</VT>
      </COMPU-CONST>
    </COMPU-SCALE>
    <!-- front left -->
    <COMPU-SCALE>
      <SHORT-LABEL>frontLeft</SHORT-LABEL>
      <SYMBOL>frontLeft_no</SYMBOL>
      <MASK>0b11000001</MASK>
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000000</UPPER-LIMIT>
```

```
          <COMPU-CONST>
            <VT>no</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <SHORT-LABEL>frontLeft</SHORT-LABEL>
          <SYMBOL>frontLeft_yes</SYMBOL>
          <MASK>0b11000001</MASK>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0b00000001</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0b00000001</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>yes</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

Listing 5.19: example for bit field text table CompuMethod

Class	**CompuConstTextContent**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the textual content of a scale.			
Base	*ARObject*, *CompuConstContent*			
Aggregated by	CompuConst.compuConstContentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
vt	VerbatimString	0..1	attr	This represents a textual constant in the computation method.

Table 5.75: CompuConstTextContent

Class	**CompuConstNumericContent**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the constant value of the computation method is a numerical value. It is separated from CompuConstFormulaContent to support compatibility with ASAM HDO.			
Base	*ARObject*, *CompuConstContent*			
Aggregated by	CompuConst.compuConstContentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
v	Numerical	0..1	attr	This represents the numerical value. **Tags:**xml.sequenceOffset=50

Table 5.76: CompuConstNumericContent

5.5.2 Physical Units, Physical Dimensions and Unit Groups

[TPS_SWCT_01285] Physical dimension ⌈Another important part of the semantics associated with a data type is its physical dimension. Units are used to augment the value with additional information like *m/s* or *liter*. This is necessary for a correct interpretation of the physical value for input and output processes.

The conversion of values into other units like *km/h* into *miles/h* is also possible. Therefore, the unit involves information about its physical dimensions.⌋*()*

[TPS_SWCT_01056] Physical dimension ⌈The substructure of physical dimensions defines all used quantities in the SI-System[19] (e.g. velocity as length/time corresponds to m/s).⌋*(RS_SWCT_02100)*

[TPS_SWCT_01057] Unit references one physical dimension ⌈The unit references one physical dimension. If the physical dimensions of two units are identical, a conversion between them is basically possible.⌋*(RS_SWCT_02100)*

Class	**Unit**			
Package	M2::MSR::AsamHdo::Units			
Note	This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiTo Unit) are applied as follows: x [{unit}] := y * [{siUnit}] * factorSiToUnit [[unit]/{siUnit}] + offsetSiToUnit [{unit}] For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied. y {siUnit} := (x*{unit} - offsetSiToUnit [{unit}]) / (factorSiToUnit [[unit]/{siUnit}] **Tags:**atp.recommendedPackage=Units			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
displayName	SingleLanguageUnit Names	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:**xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from SI Units to units. The inverse is used for conversion from units to SI Units. **Tags:**xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:**xml.sequenceOffset=40
physical Dimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:**xml.sequenceOffset=50

Table 5.77: Unit

[TPS_SWCT_01058] `UnitGroup` ⌈The `UnitGroup`s determine if such a conversion is appropriate.⌋*(RS_SWCT_02100)*

Figure 5.54 depicts the concept how units are defined.

For a detailed description of these elements please refer to the [29]. Standard units are already predefined for AUTOSAR in form of a description file.

[19]For the definition of what SI units are, see http://physics.nist.gov/cuu/Units/

[TPS_SWCT_01736] Default values for `Unit.physicalDimension` ⌈If a `Unit` does not define the attribute `Unit.physicalDimension`, the default `PhysicalDimension` with the `shortName` `NoDimension` applies where all physical exponents are set to 0.⌋*(RS_SWCT_02100)*

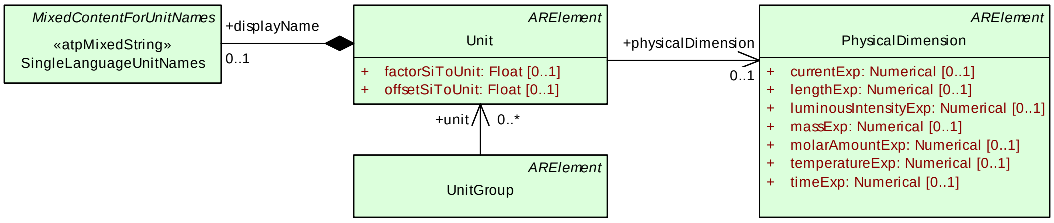


Figure 5.54: Definition of SI based units

[TPS_SWCT_01059] Exponent for each of the seven fundamental dimensions ⌈For basing a new unit directly upon SI units an exponent for each of the seven fundamental dimensions and its corresponding SI unit needs to be specified.⌋*(RS_SWCT_02100)*

[TPS_SWCT_01737] Default values for physical exponents ⌈The default value of attributes `currentExp`, `lengthExp`, `luminousIntensityExp`, `massExp`, `molarAmountExp`, `temperatureExp`, `timeExp` is 0.⌋*(RS_SWCT_02100)*

[TPS_SWCT_01060] Negative exponents ⌈Negative exponents are allowed.⌋*(RS_SWCT_02100)*

Note that quantities like "%" are not derived from SI units and therefore have no association to a physical dimension.

Class	**PhysicalDimension**			
Package	M2::MSR::AsamHdo::Units			
Note	This class represents a physical dimension. If the physical dimension of two units is identical, then a conversion between them is possible. The conversion between units is related to the definition of the physical dimension. Note that the equivalence of the exponents does not per se define the convertibility. For example Energy and Torque share the same exponents (Nm). Please note further the value of an exponent does not necessarily have to be an integer number. It is also possible that the value yields a rational number, e.g. to compute the square root of a given physical quantity. In this case the exponent value would be a rational number where the numerator value is 1 and the denominator value is 2. **Tags:**atp.recommendedPackage=PhysicalDimensions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
currentExp	Numerical	0..1	attr	This attribute represents the exponent of the physical dimension "electric current". **Tags:**xml.sequenceOffset=50
lengthExp	Numerical	0..1	attr	The exponent of the physical dimension "length". **Tags:**xml.sequenceOffset=20

▽

△

Class	**PhysicalDimension**			
luminous IntensityExp	Numerical	0..1	attr	The exponent of the physical dimension "luminous intensity". **Tags:**xml.sequenceOffset=80
massExp	Numerical	0..1	attr	The exponent of the physical dimension "mass". **Tags:**xml.sequenceOffset=30
molarAmount Exp	Numerical	0..1	attr	The exponent of the physical dimension "quantity of substance". **Tags:**xml.sequenceOffset=70
temperatureExp	Numerical	0..1	attr	The exponent of the physical dimension "temperature". **Tags:**xml.sequenceOffset=60
timeExp	Numerical	0..1	attr	The exponent of the physical dimension "time". **Tags:**xml.sequenceOffset=40

Table 5.78: PhysicalDimension

AUTOSAR provides the ability to map two `PhysicalDimension`s onto each other with the implication that the two mapped `PhysicalDimension`s shall be considered compatible (for more explanation please refer to [constr_1053]).

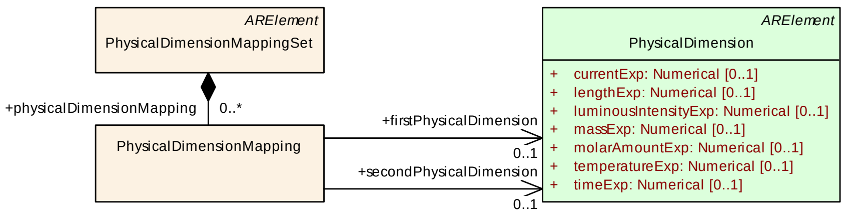


Figure 5.55: Modeling of `PhysicalDimensionMapping`

`PhysicalDimensionMapping`s are aggregated in form of `PhysicalDimensionMappingSet`s. This allows for gathering semantically related `PhysicalDimensionMapping`s into the same `PhysicalDimensionMappingSet`.

Class	**PhysicalDimensionMappingSet**			
Package	M2::MSR::AsamHdo::Units			
Note	This class represents a container for a list of mappings between PhysicalDimensions. **Tags:**atp.recommendedPackage=PhysicalDimensionMappingSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
physical Dimension Mapping	PhysicalDimension Mapping	*	aggr	This aggregation represents a concrete collections of PhysicalDimensionMappings in the context of one PhysicalDimensionMappingSet.

Table 5.79: PhysicalDimensionMappingSet

Class	PhysicalDimensionMapping			
Package	M2::MSR::AsamHdo::Units			
Note	This class represents a specific mapping between two PhysicalDimensions.			
Base	*ARObject*			
Aggregated by	PhysicalDimensionMappingSet.physicalDimensionMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
firstPhysicalDimension	PhysicalDimension	0..1	ref	This represents the first PhysicalDimension of the enclosing PhysicalDimensionMapping.
secondPhysicalDimension	PhysicalDimension	0..1	ref	This represents the first PhysicalDimension of the enclosing PhysicalDimensionMapping.

Table 5.80: PhysicalDimensionMapping

[constr_1915] Existence of attribute PhysicalDimensionMapping.firstPhysicalDimension ⌈For each PhysicalDimensionMapping, attribute firstPhysicalDimension shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1916] Existence of attribute PhysicalDimensionMapping.secondPhysicalDimension ⌈For each PhysicalDimensionMapping, attribute secondPhysicalDimension shall exist **at the time when the contract phase generation is executed**.⌋*()*

In the following example, the units “km” and “m” and their physical dimension named “Len1” are specified. The SI base unit is “m” (Meter).

The default value of attribute Unit.offsetSiToUnit is 0, the default of Unit.factorSiToUnit is 1 (see [TPS_SWCT_01492]).

Given the equality 1 km == 1000 m, the following equation applies:

$$x[km] := y * [m] * 0.001[km/m] + 0[km]$$

This correlation is reflected in the example ARXML contained in Listing 5.20.

```
<UNIT>
  <SHORT-NAME>KiloMtr</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Kilo Meter</L-4>
  </LONG-NAME>
  <DISPLAY-NAME>km</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>0.001</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO-UNIT>0</OFFSET-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF BASE="PhysicalDimensions" DEST="PHYSICAL-
     DIMENSION">/SiUnit/Len1</PHYSICAL-DIMENSION-REF>
</UNIT>

<UNIT>
  <SHORT-NAME>Mtr</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Meter</L-4>
  </LONG-NAME>
```

```
  <DISPLAY-NAME>m</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>1</FACTOR-SI-TO-UNIT>
  <OFFSET-SI-TO-UNIT>0</OFFSET-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF BASE="PhysicalDimensions" DEST="PHYSICAL-
     DIMENSION">/SiUnit/Len1</PHYSICAL-DIMENSION-REF>
</UNIT>

<PHYSICAL-DIMENSION>
  <SHORT-NAME>Len1</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Length 1</L-4>
  </LONG-NAME>
  <LENGTH-EXP>1</LENGTH-EXP>
</PHYSICAL-DIMENSION>
```

Listing 5.20: Example for Unit and PhysicalDimension

[constr_1026] Compatibility of Units ⌈If a SwDataDefProps references a Unit and the SwDataDefProps has a reference to either of/or both

- a CompuMethod that in turn references a Unit
- a DataConstr that in turn references a Unit

then the Units referenced from

- SwDataDefProps
- SwDataDefProps.compuMethod
- SwDataDefProps.dataConstr

shall be compatible **at the time when the contract phase generation is executed**.⌋*()*

Please note that for the sake of model consistency, it is also possible to define a meaningless Unit for all the pieces of data that conceptually do not really have a Unit attached to them (e.g. ApplicationPrimitiveDataTypes of category BOOLEAN).

By looking at the model, it becomes clear that the subject of whether to assign a Unit has been given a thought and the lack of a Unit is not simply the result of an oversight. For example, the AUTOSAR General Blueprints [30] define the Unit NoUnit for exactly this purpose.

[constr_1255] ApplicationPrimitiveDataTypes of category BOOLEAN and STRING ⌈If a Unit is referenced from within SwDataDefProps and/or PhysConstrs owned by an ApplicationPrimitiveDataTypes of category BOOLEAN and STRING it is required that this Unit represents a meaningless unit, i.e. the referenced physicalDimension shall not define any exponent value other than 0 **at any time in the workflow**.⌋*()*

[TPS_SWCT_01068] Units can be grouped with the help of UnitGroup ⌈Units can be grouped with the help of UnitGroup. This grouping is intended as a logical grouping which allows for example an MCD (Measurement Calibration Diagnostic) device to present different unit systems to the user such that he can choose the most appropriate one.⌋*(RS_SWCT_02100)*

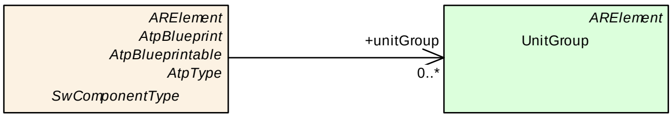


Figure 5.56: Relation of SwComponentType to UnitGroup

Class	**UnitGroup**			
Package	M2::MSR::AsamHdo::Units			
Note	This meta-class represents the ability to specify a logical grouping of units.The category denotes the unit system that the referenced units are associated to. In this way, e.g. country-specific unit systems (CATEGORY="COUNTRY") can be defined as well as specific unit systems for certain application domains. In the same way a group of equivalent units, can be defined which are used in different countries, by setting CATEGORY="EQUIV_UNITS". KmPerHour and MilesPerHour could such be combined to one group named "vehicle_speed". The unit MeterPerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named "speed". Note that the UnitGroup does not ensure the physical compliance of the units. This is maintained by the physical dimension. **Tags:**atp.recommendedPackage=UnitGroups			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
unit	Unit	*	ref	This represents one particular unit in the UnitGroup. **Tags:**xml.sequenceOffset=20

Table 5.81: UnitGroup

The association from SwComponentType to UnitGroup (beside the obvious use case to allow for the specification of unitGroups relevant for the enclosing SwComponentType in particular) is supposed to support the identification of UnitGroups relevant for the enclosing System. This aspect facilitates the creation of ASAM MCD2 files for a concrete ECU.

According to [29] the following three values for categorys are recommended in the context of UnitGroup:

- COUNTRY collects units which are common in a particular country, denoted by the shortName / longName of the UnitGroup
- CALCULATION refers to specific units intended for the creation of data types. In this category of UnitGroup, several Units may refer to the same PhysicalDimension as well as to different PhysicalDimension.
- EQUIV_UNITS define a group of equivalent units, which are used for example in different countries.

Additional values for category may be mutually agreed between the stakeholders.

In the example shown in Figure 5.57, Units are classified by country and use.

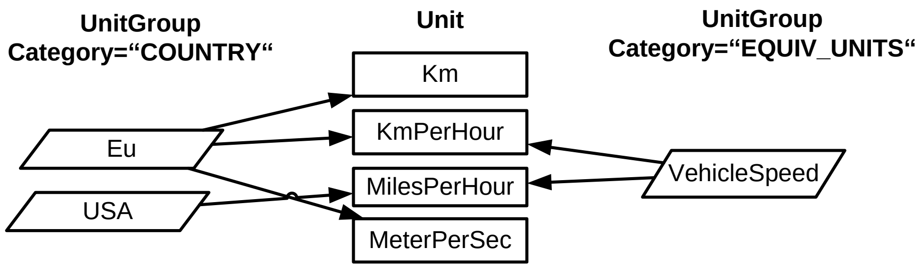


Figure 5.57: Example for units and unit groups

Assume "MilesPerHour" should be converted to a European unit: Based on the physicalDimension a conversion to "MeterPerSec" as well as "KmPerHour" is possible. In this case "KmPerHour" is preferred because "MilesPerHour" and "KmPerHour" are both members of the UnitGroup named "VehicleSpeed".

In contrast to this, "MeterPerSec" is not considered as appropriate for "VehicleSpeed" in this specific example.

[TPS_SWCT_01061] Conversion of units ⌈If a unit has to be converted according to the chosen country code, the physicalDimension of both units shall be the same. If another unit shares the same UnitGroup with a category of EQUIV_UNITS it is preferred as target of the conversion.⌋ *(RS_SWCT_02100)*

5.5.3 Data Constraints

Section 5.2.4.1 already shows an example on how to define constraints for the physical range of a data type, see Figure 5.4.

[TPS_SWCT_01286] DataConstr ⌈In general, the meta-class DataConstr can be aggregated (via SwDataDefProps.dataConstr) to define various constraints for the possible values of a data type. This includes limits for the physical and internal range, as well as special constraints (monotony) for the setup of axis definition.⌋ *()*

Figure 5.58 and the following class tables show the meta-classes involved in the definition of constraints.

A more detailed documentation of these meta-classes can be found in [29]. As refinement of these definitions, the following values apply for constrLevel:

[constr_2561] Application of DataConstrRule.constrLevel ⌈DataConstrRule.constrLevel is limited to

0: This represents so called “hard limits”. They shall always be specified.

1: This represents so called “soft limits”. Soft limits may be violated after confirmation by the user of an MCD-System.

This rule applies **at any time in the workflow**. Other values may exist, but the semantics is outside the AUTOSAR scope.

⌋*()*

[TPS_SWCT_01287] Standard limits and extended limits in the ASAM-MCD2 (ASAP2) specification ⌈The ASAM-MCD2 (ASAP2) specification [28] defines standard limits and extended limits. If extended limits exist, the standard limits may be violated upon user confirmation. Note that in consequence, of this definition, the following approach applies for A2L generation:

- If only one DataConstrRule with constrLevel set to **0** is specified, it represents the standard limits in A2L. No extended limits are generated.
- If two DataConstrRule exist, then:
  - the one with constrLevel set to **0** represents to the extended limits
  - the one with constrLevel set to **1** represents to the standard limits

Note that even if this is somehow counter-intuitive (since the one with constrLevel set to 0 changes its role), it matches the best to the definitions in ASAM-MCD2.⌋*()*

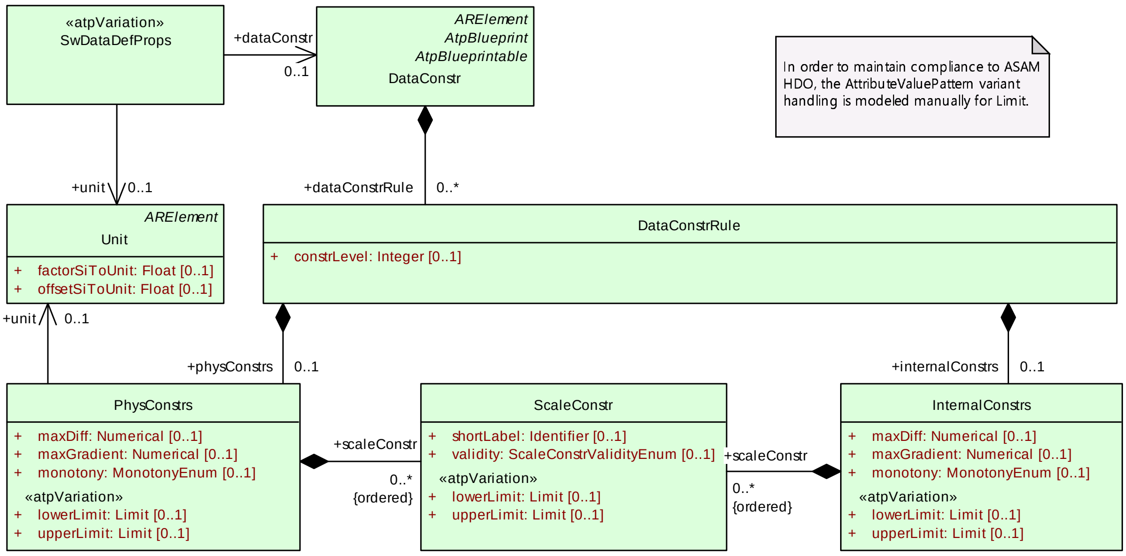


Figure 5.58: Meta-model for defining Data Constraints

Class	**DataConstr**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints on data. **Tags:**atp.recommendedPackage=DataConstrs			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	DataConstr			
dataConstrRule	DataConstrRule	*	aggr	This is one particular rule within the data constraints. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 5.82: DataConstr

Class	**DataConstrRule**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to express one specific data constraint rule.			
Base	*ARObject*			
Aggregated by	DataConstr.dataConstrRule			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
constrLevel	Integer	0..1	attr	This attribute describes the category of a constraint. One of its functions is in the area of constraint violation, where it can be used from a certain level, to produce error messages. The lower the level, the more stringent the check. Used to distinguish hard or soft limits. **Tags:**xml.sequenceOffset=20
internalConstrs	InternalConstrs	0..1	aggr	Describes the limitations applicable on the internal domain (as opposed to the physical domain). **Tags:**xml.sequenceOffset=40
physConstrs	PhysConstrs	0..1	aggr	Describes the limitations applicable on the physical domain (as opposed to the internal domain). **Tags:**xml.sequenceOffset=30

Table 5.83: DataConstrRule

Class	**PhysConstrs**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to express physical constraints. Therefore it has (in opposite to InternalConstrs) a reference to a Unit.			
Base	*ARObject*			
Aggregated by	DataConstrRule.physConstrs			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
maxDiff	Numerical	0..1	attr	Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. **Tags:**xml.sequenceOffset=60

▽

△

Class	**PhysConstrs**			
maxGradient	Numerical	0..1	attr	This element specifies the maximum slope that may be used in curves and maps. **Tags:**xml.sequenceOffset=50
monotony	MonotonyEnum	0..1	attr	This specifies the monotony constraints on the data object. Note that this applies only to curves and maps. **Tags:**xml.sequenceOffset=70
scaleConstr (ordered)	ScaleConstr	*	aggr	This is one particular scale which contributes to the data constraints. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false
unit	Unit	0..1	ref	This is the unit to which the physical constraints relate to. In particular, it is the physical unit of the specified limits. **Tags:**xml.sequenceOffset=80
upperLimit	Limit	0..1	attr	This specifies the upper limit of the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=30

Table 5.84: PhysConstrs

Class	**InternalConstrs**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to express internal constraints.			
Base	*ARObject*			
Aggregated by	DataConstrRule.internalConstrs			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
maxDiff	Numerical	0..1	attr	Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. **Tags:**xml.sequenceOffset=60
maxGradient	Numerical	0..1	attr	This element specifies the maximum slope that may be used in maps and curves. **Tags:**xml.sequenceOffset=50
monotony	MonotonyEnum	0..1	attr	This element specifies the monotony characteristics of the current internal or physical limits. The following table shows the monotony characteristics which are to be filled through the corresponding values. If the element has no contents or if it is omitted, "no Monotony" is the default content. **Tags:**xml.sequenceOffset=70

▽

△

Class	**InternalConstrs**			
scaleConstr (ordered)	ScaleConstr	*	aggr	This is one particular scale which contributes to the data constraints. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false
upperLimit	Limit	0..1	attr	This specifies the upper limit defined by the constraint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=30

Table 5.85: InternalConstrs

Class	**ScaleConstr**			
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints as a list of intervals (called scales).			
Base	*ARObject*			
Aggregated by	InternalConstrs.scaleConstr, PhysConstrs.scaleConstr			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:**xml.sequenceOffset=30
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
shortLabel	Identifier	0..1	attr	This element specifies a short name for the scaleConstr. This can for example be used to create more specific messages of a constraint checker. The constraints cannot be associated in the meta-model, therefore shortLabel is somehow a substitute for shortName. **Tags:**xml.sequenceOffset=20
upperLimit	Limit	0..1	attr	This specifies the upper limit of a the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
validity	ScaleConstrValidityEnum	0..1	attr	Specifies if the values defined by the scales are considered to be valid. If the attribute is missing then the default value is "VALID". **Tags:**xml.attribute=true

Table 5.86: ScaleConstr

Enumeration	**ScaleConstrValidityEnum**
Package	M2::MSR::AsamHdo::Constraints::GlobalConstraints
Note	This enumerator specifies the possible values of a scale.
Aggregated by	ScaleConstr.validity
Literal	*Description*
notAvailable	Currently invalid area The value usually is presented by the ECU but can currently not be performed due to e.g. initialization or temporary problems. Please note, that this behavior appears during runtime and cannot be handled while data is edited. **Tags:**atp.EnumerationLiteralIndex=0
notDefined	Indicates an area which is marked in a specification (e.g. as reserved) Shall usually not be set by the ECU but is used by a tester to verify correct ECU. **Tags:**atp.EnumerationLiteralIndex=1
notValid	The ECU cannot process the requested data. **Tags:**atp.EnumerationLiteralIndex=2
valid	Current value is within a valid range and can be presented to user as is. **Tags:**atp.EnumerationLiteralIndex=3

Table 5.87: ScaleConstrValidityEnum

Primitive	**Limit**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This class represents the ability to express a numerical limit. Note that this is in fact a NumericalVariation Point but has the additional attribute intervalType. **Tags:** xml.xsd.customType=LIMIT-VALUE xml.xsd.pattern=(0[xX][0-9a-fA-F]+)|(0[0-7]+)|(0[bB][0-1]+)|(([+\-]?[1-9][0-9]+(\.[0-9]+)?|[+\-]?[0-9](\.[0-9]+)?)([eE]([+\-]?)[0-9]+)?)|\.0|INF|-INF|NaN xml.xsd.type=string			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
intervalType	IntervalTypeEnum	0..1	attr	This specifies the type of the interval. If the attribute is missing the interval shall be considered as "CLOSED". **Tags:**xml.attribute=true

Table 5.88: Limit

Enumeration	**MonotonyEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This enumerator denotes the values for specification of monotony for e.g. curves.
Aggregated by	InternalConstrs.monotony, PhysConstrs.monotony, *SwCalprmAxisTypeProps*.monotony
Literal	*Description*
decreasing	This indicates that the related curve needs to be monotony decreasing. **Tags:**atp.EnumerationLiteralIndex=0
increasing	This indicates that the related curve needs to be monotony increasing. **Tags:**atp.EnumerationLiteralIndex=1
monotonous	This indicates that the values shall be monotonously decreasing or increasing, depending on the trend set by the first values of the series. **Tags:**atp.EnumerationLiteralIndex=2
noMonotony	This indicates that the related curve needs not to be monotony. **Tags:**atp.EnumerationLiteralIndex=3

▽

△

Enumeration	**MonotonyEnum**
strictlyDecreasing	This indicates that the related curve needs to be strictly monotony decreasing. **Tags:**atp.EnumerationLiteralIndex=4
strictlyIncreasing	This indicates that the related curve needs to be strictly monotony increasing. **Tags:**atp.EnumerationLiteralIndex=5
strictMonotonous	This indicates that the values shall be strict monotonously decreasing or increasing, depending on the trend set by the first values of the series. **Tags:**atp.EnumerationLiteralIndex=6

Table 5.89: MonotonyEnum

[TPS_SWCT_01288] Interpretation of `PhysConstrs` and `InternalConstrs` by tools ⌈`DataConstr` is an `ARElement` which can be reused by several data type specifications. Especially an `ImplementationDataType` and an `ApplicationDataType` which are mapped to each other, can refer to the same constraints, or they can define their own constraints.

To avoid conflicts, in both cases `PhysConstrs` shall be interpreted by tools only with respect to application data types while `InternalConstrs` shall be interpreted only with respect to implementation data types.

If only `PhysConstrs` are provided to `ApplicationDataType`s the `CompuMethod` can be used to compute the `InternalConstrs`.⌋*()*

[TPS_SWCT_01289] Semantics of `Limit` ⌈Technically, a `Limit` specifies a boundary of the interval of valid values for a given context (i.e. a data type). Please note that the boundary might or might not be part of the interval itself, i.e. the interval might be open or closed. From the formal point of view, the range represents all real numbers defined by:

$$\begin{aligned} range \;=\; & \{x \in \Re \,\|\, lowerLimit.value < x < upperLimit.value\} \\ & \cup \{lowerLimit.value \,\|\, lowerLimit.intervalType == \text{“}CLOSED''\} \\ & \cup \{upperLimit.value \,\|\, upperLimit.intervalType == \text{“}CLOSED''\} \end{aligned}$$

⌋*()*

Please note that `Limit` inherits from `AbstractNumericalVariationPoint`. This means it is a number which may be subject to variability. For this reason, it is not possible to constrain the content already in the xml schema.

[constr_1191] Value of `Limit` shall yield a numerical value ⌈After all variability is bound, the content obtained from a limit shall yield a numerical value **`at any time in the workflow`**.⌋*()*

Nevertheless, it is not possible to distinguish on this level between float and integer values. Consequently, [constr_1191] will not take the burden from an AUTOSAR tool to decide whether the value provided as a limit actually makes sense in any of the given contexts.

Enumeration	**IntervalTypeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This enumerator specifies the type of an interval.
Aggregated by	Limit.intervalType, LimitValueVariationPoint.intervalType
Literal	**Description**
closed	The area is limited by the value given. The value itself is included. **Tags:**atp.EnumerationLiteralIndex=0
open	The area is limited by the value given. The value itself is not included. **Tags:**atp.EnumerationLiteralIndex=2

Table 5.90: IntervalTypeEnum

5.5.3.1 Physical Limits

Physical limits can be given at various places in the AUTOSAR Meta-Model, e.g. in context of `ApplicationDataType`s, `DataPrototype`s but also without the usage of the type-prototype-pattern in `Compound Primitive Data Type`s (e.g `SwAxisIndividual.dataConstr`).

Nevertheless, the usage of `PhysConstrs` requires a `CompuMethod` for the calculation of the numerical limits, which cannot be applied for textual conversions. For this reason following definition applies:

[TPS_SWCT_01761] Physical limits of pure textual conversions ⌈It is not possible to define the lower or upper limit of a set of textual labels. Therefore, it is not possible to define limits for an object that can only take elements of a set of textual labels as the value.⌋*()*

Please note, as a consequence of [TPS_SWCT_01761] for data defined by means of a `CompuMethod` of `category` `TEXTTABLE` or `BITFIELD_TEXTTABLE` and additionally a `DataConstr` with a `dataConstrRule.physConstrs` the given `physConstrs` has no meaning.

[TPS_SWCT_01762] Physical limits of mixed textual conversions ⌈The definition of the physical limits of a piece of data described by a `CompuMethod` of `category` `SCALE_LINEAR_AND_TEXTTABLE` and `SCALE_RATIONAL_AND_TEXTTABLE` can only be specified for the **linear** or **rational** part.

In addition, the defined textual labels can be used for the conversion.⌋*()*

For clarification, [TPS_SWCT_01761] and [TPS_SWCT_01762] do not limit the usage of `DataConstr.dataConstrRule.internalConstrs` which may define further and even tighter constraints on implementation level.

Those internalConstrs might be even given in context of a Compound Primitive Data Type (for example, in the context of an SwAxisIndividual.inputVariableType (see [TPS_SWCT_01865]) or SwAxisIndividual.dataConstr).

5.5.4 Addressing Methods

In an ECU there might be various methods to access a particular object (e.g. measurement or calibration parameter) according to a given address. This variety might come from different kind of memory (near, far, . . .) but also from indirections which are introduced by the compiler.

[TPS_SWCT_01290] SwAddrMethod ⌈In order to allow a measurement and calibration system to access such objects SwAddrMethods are specified. Another purpose of this feature is to support the definition of abstract memory sections, i.e. to specify which variables shall be put together in the same sections in case of generated code (especially for data allocated by the RTE).

SwAddrMethod will be used to group data, for example, to cover the fact that sometimes it is required that one or more calibration parameters out of the overall collection of calibration parameters of a SwComponentPrototype respectively an AUTOSAR software component shall be placed in another memory location than the other parameters of the SwComponentPrototype respectively the AUTOSAR software component.⌋*()*

[TPS_SWCT_01291] Association of MemorySection with SwAddrMethod ⌈In Implementation the particular MemorySection is associated with the SwAddrMethod. This association indicates that all objects of the associated addressing method shall be placed in the given memory section.⌋*()*

Class	**MemorySection**
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage
Note	Provides a description of an abstract memory section used in the Implementation for code or data. It shall be declared by the Implementation Description of the module or component, which actually allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the generated Implementation Description of the RTE shall contain the corresponding MemorySections. The attribute "symbol" (if symbol is missing: "shortName") defines the module or component specific section name used in the code. For details see the document "Specification of Memory Mapping". Typically the section name is build according the pattern: <SwAddrMethod shortName>[_<further specialization nominator>][_<alignment>] where • **[<SwAddrMethod shortName>]** is the shortName of the referenced SwAddrMethod • **[_<further specialization nominator>]** is an optional infix to indicate the specialization in the case that several MemorySections for different purpose of the same Implementation Description referring to the same or equally named SwAddrMethods. • **[_<alignment>]** is the alignment attributes value and is only applicable in the case that the memoryAllocationKeywordPolicy value of the referenced SwAddrMethod is set to addrMethodShortNameAndAlignment ▽

▽

△

Class	MemorySection			
	△ MemorySection used to Implement the code of RunnableEntitys and BswSchedulableEntitys shall have a symbol (if missing: shortName) identical to the referred SwAddrMethod to conform to the generated RTE header files. In addition to the section name described above, a prefix is used in the corresponding macro code in order to define a name space. This prefix is by default given by the shortName of the BswModule Description resp. the SwComponentType. It can be superseded by the prefix attribute.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ResourceConsumption.memorySection			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
alignment	AlignmentType	0..1	attr	The attribute describes the typical alignment of objects within this memory section.
executableEntity	ExecutableEntity	*	ref	Reference to the ExecutableEntitites located in this section. This allows to locate different Executable Entitities in different sections even if the associated Sw Addrmethod is the same. This is applicable to code sections only.
option	Identifier	*	attr	The service (in AUTOSAR: BswModuleEntry) is implemented in a way that it either resolves to aninline function or to a standard function depending on conditions set at a later point in time. The following two values are standardized (to be used for code sections only and exclusively to each other): • INLINE - The code section is declared with the keyword "inline". • LOCAL_INLINE - The code section is declared with the keyword "static inline". In both cases (INLINE and LOCAL_INLINE) the inline expansion depends on the compiler. Depending on this, the code section either corresponds to an actual section in memory or is put into the section of the caller.
prefix	SectionNamePrefix	0..1	ref	The prefix used to set the memory section's namespace in the code. The existence of a prefix element supersedes rules for a default prefix (such as the Bsw ModuleDescription's shortName). This allows the user to define several name spaces for memory sections within the scope of one module, cluster or SWC.
size	PositiveInteger	0..1	attr	The size in bytes of the section.
swAddrmethod	SwAddrMethod	0..1	ref	This association indicates that this module specific (abstract) memory section is part of an overall SwAddr Method, referred by the upstream declarations (e.g. calibration parameters, data element prototypes, code entities) which share a common addressing strategy. This can be evaluated for the ECU configuration of the build support. This association shall always be declared by the Implementation description of the module or component, which allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the software components only declare the grouping of its data prototypes to SwAddrMethods, and the generated Implementation Description of the RTE actually sets up this association.

▽

△

Class	MemorySection			
symbol	Identifier	0..1	attr	Defines the section name as explained in the main description. By using this attribute for code generation (instead of the shortName) it is possible to define several different MemorySections having the same name - e.g. symbol = CODE - but using different sectionName Prefixes.

Table 5.91: MemorySection

[constr_10033] Existence of MemorySection.swAddrmethod ⌈For each MemorySection, attribute swAddrmethod shall exist **at the time when the contract phase generation is executed.**⌋*()*

[constr_10034] Existence of MemorySection.alignment ⌈For each MemorySection, attribute alignment shall exist **at the time when the contract phase generation is executed** if the attribute MemorySection.swAddrmethod.memoryAllocationKeywordPolicy is set to MemoryAllocationKeywordPolicyType.addrMethodShortNameAndAlignment.⌋*()*

Rationale for the existence of [constr_10033]: If the MemorySection is configured such that MemorySection.swAddrmethod.memoryAllocationKeywordPolicy is set to MemoryAllocationKeywordPolicyType.addrMethodShortNameAndAlignment, then the alignment attribute is mandatory because the implementation of the memory mapping shall be able to formally consider the alignment for the choice of memory sections.

Class	**SwAddrMethod**			
Package	M2::MSR::DataDictionary::AuxillaryObjects			
Note	Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components. **Tags:**atp.recommendedPackage=SwAddrMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
memory Allocation KeywordPolicy	MemoryAllocation KeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of the MemorySection in with the related objects shall be placed. These properties are handled as to be selected. The intended options are mentioned in the list. In the Memory Mapping configuration, this option list is used to determine an appropriate MemMapAddressing ModeSet.

▽

△

Class	SwAddrMethod			
section Initialization Policy	SectionInitialization PolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableData Prototypes). Therefore this is an implementation constraint for initialization code of BSW modules (especially RTE) as well as the start-up code which initializes the memory segment to which the AutosarData Prototypes referring to the SwAddrMethod's are later on mapped. If the attribute is not defined it has the identical semantic as the attribute value "INIT"
sectionType	MemorySectionType	0..1	attr	Defines the type of memory sections which can be associated with this addressing method.

Table 5.92: SwAddrMethod

[TPS_SWCT_01456] Predefined values for `MemorySection.option` and `SwAddrMethod.option` ⌈The following values of `MemorySection.option` and `SwAddrMethod.option` are predefined by AUTOSAR:

`resetSafe` This corresponds to variables of ECU-functions which values shall endure a ECU reset.

`protected` This corresponds to variables, constants, and code which shall not be accessible and modifiable from the outside without a security mechanism.

`offline` This corresponds to calibration parameters which shall not be modifiable during ECU operation.

`coreGlobal` This corresponds to variables, constants, and code which have to be accessible by any core in case of multi-core ECUs.

`coreLocal` This corresponds to variables, constants, and code which have to be accessible by one core in case of multi-core ECUs.

`nvData` This corresponds to variables of ECU-functions which shall be stored in non-volatile data. This option is applicable for memory used as a `RAM Block` managed by the NvM.

`safetyQM` This corresponds to variables, constants, and code without any safety integrity level and therefore having a QM rating.

`safetyAsilA` This corresponds to variables, constants, and code with the safety integrity level A.

`safetyAsilB` This corresponds to variables, constants, and code with the safety integrity level B.

`safetyAsilC` This corresponds to variables, constants, and code with the safety integrity level C.

`safetyAsilD` This corresponds to variables, constants, and code with the safety integrity level D.

configClassPreBuild This corresponds to config data which is assigned at pre-compile or link time.

configClassPostBuild This corresponds to config data which is assigned at post-build time.

⌋()

Obviously, the multiplicity of both the attribute MemorySection.option and SwAddrMethod.option allows for the appearance of more than one value. For example, a combination of the values resetSafe, protected, and safetyAsilC makes perfect sense on a particular list and can be used to express a meaning that combines the semantics of both values with each other.

However, this combination of values is not arbitrarily possible. It is therefore necessary to formulate a constraint that regulates the appearance of the safety-related values mentioned in [TPS_SWCT_01456].

In other words, it would not make any sense to attribute a given memory object with two different ASIL [31] values appearing on the same list.

If these values were combined on a particular list, the intended semantics would be ambiguous and could not clearly be determined. Therefore, [constr_1311] applies.

[constr_1311] Appearance of safety-related possible values of MemorySection.option or SwAddrMethod.option ⌈Any given collection of values stored in the attributes MemorySection.option or SwAddrMethod.option according to [TPS_SWCT_01456] shall at most include a single value out of the following list **at the time when the RTE is generated**:

- **safetyQM**
- **safetyAsilA**
- **safetyAsilB**
- **safetyAsilC**
- **safetyAsilD**

⌋()

[constr_1381] Appearance of core-related possible values of MemorySection.option or SwAddrMethod.option ⌈Any given collection of values stored in the attributes MemorySection.option or SwAddrMethod.option according to [TPS_SWCT_01456] shall at most include a single value out of the following list **at the time when the RTE is generated**:

- **coreGlobal**
- **coreLocal**

⌋()

[TPS_SWCT_01294] Missing SwDataDefProps.swAddrMethod ⌈If the association SwDataDefProps.swAddrMethod is missing the object can be placed anywhere without restriction, e.g. using a default behavior of the RTE generator. Contradicting specifications (e.g. two different component types request different associations for one particular SwAddrMethod) shall be flagged as an error.⌋*()*

Figure 5.60 illustrates the usage of SwAddrMethod in the context of a DataPrototype.

[TPS_SWCT_01292] Usage of SwAddrMethod in the context of a DataPrototype ⌈The software component which defines the DataPrototype will in general not be the same to which the Implementation that actually contains the description of the MemorySection belongs.

The reason for this is that the resources for data allocated by the RTE will be described in the Implementation of the RTE. The indirection via SwAddrMethod makes this possible.⌋*()*

[TPS_SWCT_01293] RTE Generator has to derive the Memory Allocation Keyword ⌈Please note that the RTE Generator has to derive the Memory Allocation Keyword used for RunnableEntitys and BswSchedulableEntitys from the shortName of the SwAddrMethod only because the alignment defined in MemorySection is not known at contract phase.⌋*()*

[constr_2034] SwAddrMethod referenced by RunnableEntitys, BswCalledEntitys, or BswSchedulableEntitys ⌈RunnableEntitys, BswCalledEntitys, and BswSchedulableEntitys shall not reference a SwAddrMethod which attribute memoryAllocationKeywordPolicy is set to addrMethodShortNameAndAlignment **at the time when the RTE is generated**.⌋*()*

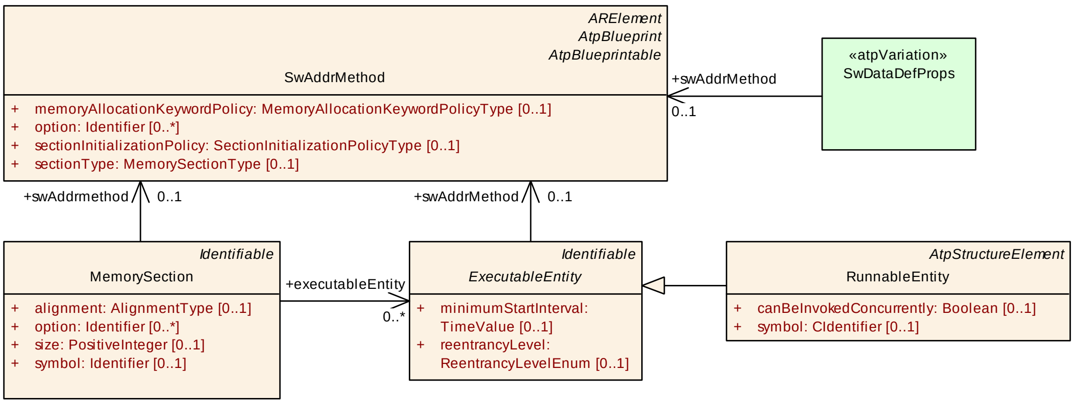


Figure 5.59: SwAddrMethod

[constr_1402] Applicability of core-related possible values of MemorySection.option or SwAddrMethod.option related to SwAddrMethod.sectionInitializationPolicy ⌈If the attribute SwAddrMethod.option or MemorySection.option is set to coreLocal then the attribute SwAddrMethod.sectionInitializationPolicy of the same SwAddrMethod respectively the MemorySection.swAddrmethod shall be either set to INIT or CLEARED **at the time when the RTE is generated**.⌋*()*

The purpose of [constr_1402] is a reduction of the complexity of memory layouts and reduce the amount of memory gaps due to allocation restrictions.

Primitive	**SectionInitializationPolicyType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	SectionInitializationPolicyType describes the intended initialization of MemorySections. The following values are standardized in AUTOSAR Methodology: • **INIT**: To be used for (explicitly or not explicitly) initialized variables. • **CLEARED**: To be used for not explicitly initialized variables. • **POWER-ON-CLEARED**: To be used for variables that are not explicitly initialized (cleared) during normal start-up. Instead these are cleared only after power on reset. Please note that the values are defined similar to the representation of enumeration types in the XML schema to ensure backward compatibility. **Tags:** xml.xsd.customType=SECTION-INITIALIZATION-POLICY-TYPE xml.xsd.type=NMTOKEN

Table 5.93: SectionInitializationPolicyType

[constr_10068] Standardized values for SectionInitializationPolicyType ⌈The following values for SectionInitializationPolicyType are reserved by the AUTOSAR standard:

INIT To be used for (explicitly or not explicitly) initialized variables.

CLEARED To be used for (explicitly or not explicitly) initialized variables.

POWER-ON-CLEARED To be used for variables that are not explicitly initialized (cleared) during normal start-up. Instead these are cleared only after power on reset.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

Please note that custom values of SectionInitializationPolicyType are currently not supported.

Enumeration	**MemorySectionType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the essential nature of the data which can be allocated in a common memory class by the means of the AUTOSAR Memory Mapping.
Aggregated by	SwAddrMethod.sectionType
Literal	*Description*

▽

△

Enumeration	**MemorySectionType**
calibrationVariables	This memory section is reserved for "virtual variables" that are computed by an MCD system during a measurement session but do not exist in the ECU memory. **Tags:**atp.EnumerationLiteralIndex=2
calprm	To be used for calibratable constants of ECU-functions. **Tags:**atp.EnumerationLiteralIndex=3
code	To be used for mapping code to application block, boot block, external flash etc. **Tags:**atp.EnumerationLiteralIndex=4
configData	Constants with attributes that show that they reside in one segment for module configuration. **Tags:**atp.EnumerationLiteralIndex=5
const	To be used for global or static constants. **Tags:**atp.EnumerationLiteralIndex=6
excludeFromFlash	This memory section is reserved for "virtual parameters" that are taken for computing the values of so-called dependent parameter of an MCD system. Dependent Parameters that are not at the same time "virtual parameters" are allocated in the ECU memory. Virtual parameters, on the other hand, are not allocated in the ECU memory. Virtual parameters exist in the ECU Hex file for the purpose of being considered (for computing the values of dependent parameters) during an offline-calibration session. **Tags:**atp.EnumerationLiteralIndex=7
var	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy. **Tags:**atp.EnumerationLiteralIndex=9

Table 5.94: MemorySectionType

Enumeration	**MemoryAllocationKeywordPolicyType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Aggregated by	SwAddrMethod.memoryAllocationKeywordPolicy
Literal	*Description*
addrMethodShort Name	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod. This is the default value if the attribute does not exist. **Tags:**atp.EnumerationLiteralIndex=0
addrMethodShort NameAndAlignment	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod and a variable alignment postfix. Thereby the alignment postfix needs to be consistent with the alignment attribute of the related MemorySection. **Tags:**atp.EnumerationLiteralIndex=1

Table 5.95: MemoryAllocationKeywordPolicyType

Primitive	**AlignmentType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents the alignment of objects within a memory section. The value is in number of bits or UNKNOWN (deprecated), 8 , 16, 32, 64 UNSPECIFIED, BOOLEAN, or PTR. Typical values for numbers are 8, 16, 32, 64. **Tags:** xml.xsd.customType=ALIGNMENT-TYPE xml.xsd.pattern=[1-9][0-9]*|0[xX][0-9a-fA-F]*|0[bB][0-1]+|0[0-7]*|UNSPECIFIED|UNKNOWN|BOOLEAN|PTR xml.xsd.type=string

Table 5.96: AlignmentType

For more information on the specification of a MemorySection refer to [6].

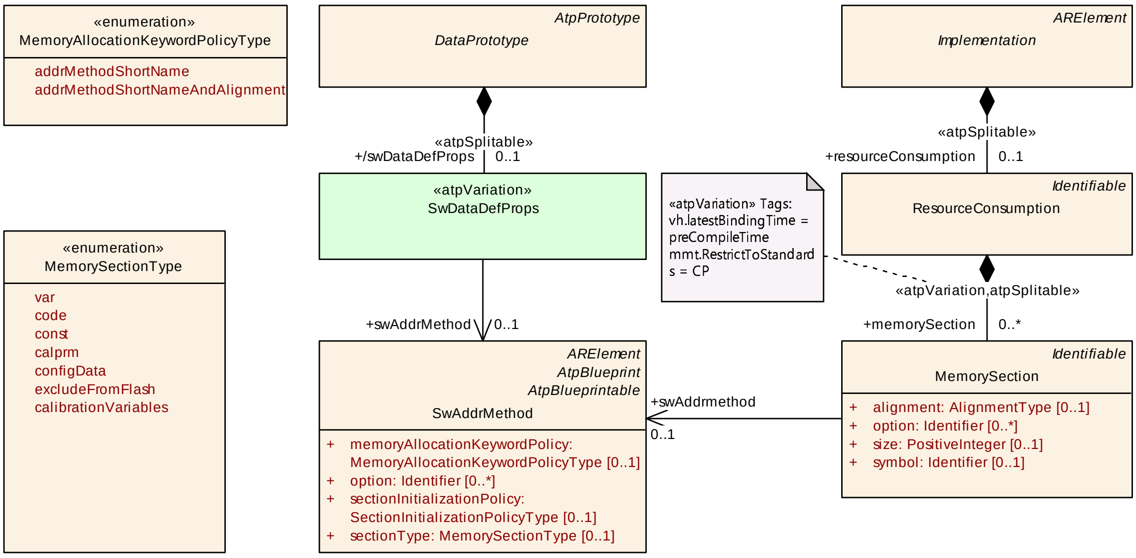


Figure 5.60: Assigning an address method to a memory section

5.5.5 Record Layouts

[TPS_SWCT_01295] **SwRecordLayout** ⌈The SwRecordLayout describes how data is serialized in the memory of an ECU. This information is important with respect to the following aspects:

- to inform a measurement and calibration system how the data is serialized in the memory of an ECU
- to make sure that the software development results in the intended data structures
- to identify the proper interpolation routines

Via the SwDataDefProps, a record-layout can be associated to a data entity. If the very same serialization approach is used for multiple ApplicationDataTypes all of

these may refer to the same SwRecordLayout even if the size of the data is different.⌋
(RS_SWCT_03215)

5.5.5.1 Specifying Record Layouts

As mentioned above, the purpose of record layout is to specify how an object (e.g. a calibration parameter) is serialized in memory of an ECU. The canonical approach for this is to define nested groups (SwRecordLayoutGroup).

These groups indicate the structure of the corresponding ImplementationDataType. The serialization is then executed by iterating over the axes of a curve, a map, or iterating along a string.

The contents of such a record layout group (SwRecordLayoutGroupContent) is a mixture of (thus nested) groups and values (SwRecordLayoutV).

These values refer to particular properties of the object (e.g. value, count, ...). By application of this pattern, the serialization of any complex object can be specified.

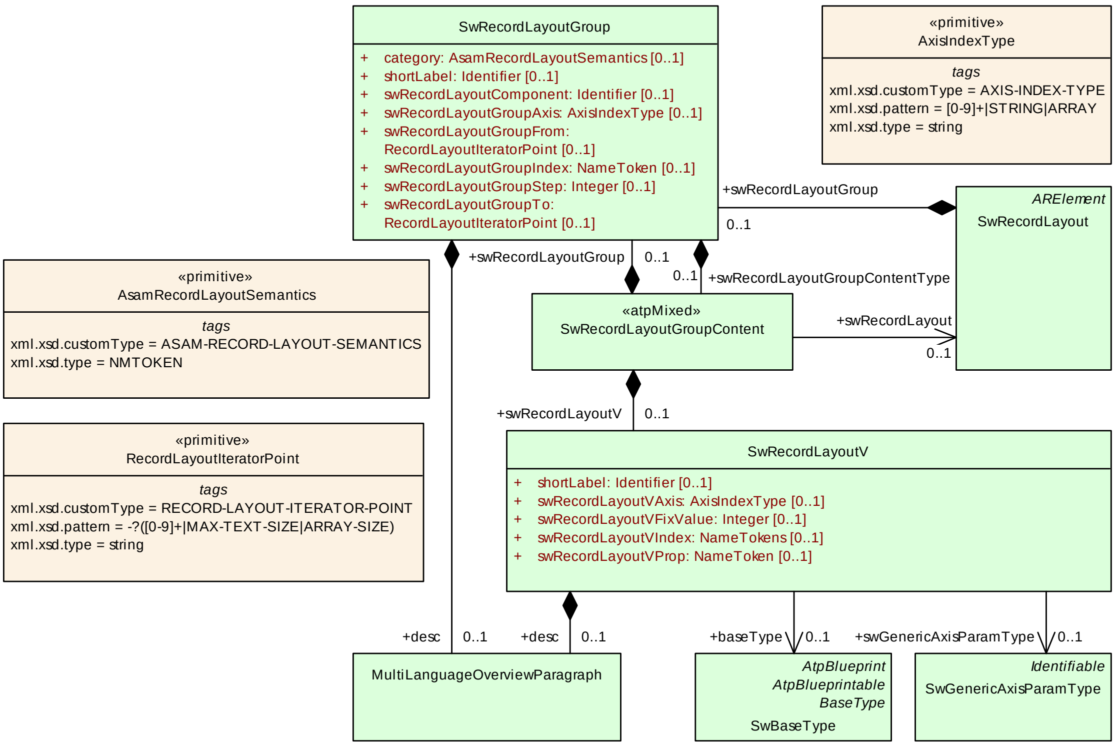


Figure 5.61: Specification of a record layout

Class	SwRecordLayout			
Package	M2::MSR::DataDictionary::RecordLayout			
Note	Defines how the data objects (variables, calibration parameters etc.) are to be stored in the ECU memory. As an example, this definition specifies the sequence of axis points in the ECU memory. Iterations through axis values are stored within the sub-elements swRecordLayoutGroup. **Tags:**atp.recommendedPackage=SwRecordLayouts			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swRecordLayoutGroup	SwRecordLayoutGroup	0..1	aggr	This is the top level record layout group. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table 5.97: SwRecordLayout

Class	SwRecordLayoutV			
Package	M2::MSR::DataDictionary::RecordLayout			
Note	This element specifies which values are stored for the current SwRecordLayoutGroup. If no baseType is present, the SwBaseType referenced initially in the parent SwRecordLayoutGroup is valid. The specification of swRecordLayoutVAxis gives the axis of the values which shall be stored in accordance with the current record layout SwRecordLayoutGroup. In swRecordLayoutVProp one can specify the information which shall be stored.			
Base	*ARObject*			
Aggregated by	SwRecordLayoutGroupContent.swRecordLayoutV			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
baseType	SwBaseType	0..1	ref	This association allows to refer to a base type in case a specific encoding is intended. If no base type is referred, the base type referenced initially in the corresponding DataPrototype is to be used. **Tags:**xml.sequenceOffset=30
desc	MultiLanguageOverviewParagraph	0..1	aggr	This aggregation allows for a brief description about the particular record layout value which can help to identify the entry. In-depth documentation should be added to the introduction of the surrounding record layout. **Tags:**xml.sequenceOffset=20
shortLabel	Identifier	0..1	attr	This attribute specifies a name which can be used e.g. when ECU code is generated from the record layout value. **Tags:**xml.sequenceOffset=3
swGenericAxisParamType	SwGenericAxisParamType	0..1	ref	This association supports the case that a value from a generic axis definition shall be stored. This value is denoted by a particular generic axis parameter type. **Tags:**xml.sequenceOffset=70

▽

△

Class	SwRecordLayoutV			
swRecordLayoutVAxis	AxisIndexType	0..1	attr	This attribute gives the index of the axis of which values that are stored in the record. swRecordVIndex refers to the symbolic names of the iterators for which the axis value shall be stored in the record. In case of nested iterators (mainly for multidimensional objects) the iterator names are specified as whitespace-separated names. These symbolic names relate to swRecordLayoutGroupIndex. The iterators are processed from left to right in such a manner that they symbolize the loop index from the outside to the inside. It is considered an error if more components are specified than axes exist in the related ApplicationDataType. **Tags:**xml.sequenceOffset=40
swRecordLayoutVFixValue	Integer	0..1	attr	This attribute specifies the filler character for the current record layout, in the form of hex digits. It is also used to specify the fix value for e.g. FIXRIGHTDIFF. **Tags:**xml.sequenceOffset=80
swRecordLayoutVIndex	NameTokens	0..1	attr	The symbolic value for iteration, or the symbolic values separated by whitespaces, refer to the symbolic values given in swRecordLayoutGroupIndex . The iterators are processed from left to right, in such a manner that they symbolize the loop index from the outside to the inside. It is considered an error if the record layout is referenced by an entity which has less number of axes than index names referenced here. **Tags:**xml.sequenceOffset=60
swRecordLayoutVProp	NameToken	0..1	attr	This attribute describes the kind of values to be stored. More details see below. The standardized values foreseen for this attribute are defined in [TPS_SWCT_01489]. **Tags:**xml.sequenceOffset=50

Table 5.98: SwRecordLayoutV

Class	SwRecordLayoutGroup			
Package	M2::MSR::DataDictionary::RecordLayout			
Note	Specifies how a record layout is set up. Using SwRecordLayoutGroup it recursively models iterations through axis values. The subelement swRecordLayoutGroupContentType may reference other SwRecordLayouts, SwRecordLayoutVs and SwRecordLayoutGroups for the modeled record layout.			
Base	*ARObject*			
Aggregated by	SwRecordLayout.swRecordLayoutGroup, SwRecordLayoutGroupContent.swRecordLayoutGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**SwRecordLayoutGroup**			
category	AsamRecordLayoutSemantics	0..1	attr	This attribute denotes the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2l keywords. It is possible to express the specific semantics of A2l recordlayout keywords in swRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. **Tags:**xml.sequenceOffset=5
desc	MultiLanguageOverviewParagraph	0..1	aggr	This aggregation allows a brief description about the particular record layout group which can help to identify the entry. In-depth documentation should be added to the introduction of the surrounding record layout. **Tags:**xml.sequenceOffset=20
shortLabel	Identifier	0..1	attr	This attribute specifies a name which can be used e.g. when ECU code is generated from the record layout group. **Tags:**xml.sequenceOffset=3
swGenericAxisParamType	SwGenericAxisParamType	0..1	ref	This association allows to specify record layout groups to iterate over generic axis parameters. For example, if the generic axis parameter is an array, the record layout group will iterate over this array. Obviously, the axis referred to by swRecordLayoutGroupAxis shall be a generic axis in which the referenced SwGenericAxisType is aggregated. **Tags:**xml.sequenceOffset=50
swRecordLayoutComponent	Identifier	0..1	attr	This attribute is used to denote the component to which the group in question applies. Thus, the record layout supports structured objects. This secures independence from the sequence of components, because they can be referred to via name. **Tags:**xml.sequenceOffset=90
swRecordLayoutGroupAxis	AxisIndexType	0..1	attr	This attribute specifies the iteration axis number for a SwRecordLayoutGroup. The current record layout group then refers exactly to the axis with this number. This means that the values are taken by iterating along the thus referenced axis. **Tags:**xml.sequenceOffset=30
swRecordLayoutGroupContentType	SwRecordLayoutGroupContent	0..1	aggr	This is the contents of the recordLayout which is produced for every step of iteration. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=100 xml.typeElement=false xml.typeWrapperElement=false
swRecordLayoutGroupFrom	RecordLayoutIteratorPoint	0..1	attr	This attribute specifies the iterator index for the point in the axis from which a record layout group is commenced. Negative values are also possible, i.e. the value -4 counts from the fourth value from the end. If this property is missing, the iteration starts with '1'. **Tags:**xml.sequenceOffset=60

▽

△

Class	SwRecordLayoutGroup			
swRecordLayoutGroupIndex	NameToken	0..1	attr	This attribute attributes a symbolic name to the iterator of the superimposed record layout group. This can be referenced as a loop index in contained SwRecordLayoutV elements. **Tags:**xml.sequenceOffset=40
swRecordLayoutGroupStep	Integer	0..1	attr	This attribute specifies the step width for the iterator index that is used for the current record layout group. Note that negative values are also possible, in case of the starting point is higher than the endpoint. If the property is missing, the step width is "1". **Tags:**xml.sequenceOffset=80
swRecordLayoutGroupTo	RecordLayoutIteratorPoint	0..1	attr	This attribute specifies the end point for the iteration. Negative values are also possible, i.e. the value -4 counts up to the fourth value from the end. If this property is not there, the iteration ends at "-1" which is the last element. Note that depending on the arraySizeSemantics of SwTextProps the iteration ends at the value specified in swMaxTextSize. **Tags:**xml.sequenceOffset=70

Table 5.99: SwRecordLayoutGroup

Class	<<atpMixed>> **SwRecordLayoutGroupContent**			
Package	M2::MSR::DataDictionary::RecordLayout			
Note	This is the contents of a RecordLayout which is inserted for every iteration. Note that since this is atpMixed, multiple properties can be inserted for each iteration.			
Base	*ARObject*			
Aggregated by	SwRecordLayoutGroup.swRecordLayoutGroupContentType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swRecordLayout	SwRecordLayout	0..1	ref	This association allows to support reusable "sub"-record layouts. In particular, the contents of the referenced record layout shall be used as if the record layout group in the referenced record layout was aggregated in the current record layout group. So, semantically it would be equivalent to replace the particular association with an aggregation of the swRecordLayoutGroup of the referenced SwRecordLayout. **Tags:**xml.sequenceOffset=110
swRecordLayoutGroup	SwRecordLayoutGroup	0..1	aggr	This aggregation provides support for nested iterations. For example, if a map is to be handled, then we might have two nested SwRecordLayoutGroups, one for the x-axis and one for the y-axis. The inner iteration runs faster. **Tags:**xml.sequenceOffset=130
swRecordLayoutV	SwRecordLayoutV	0..1	aggr	Particular Value specification for this record layout group. **Tags:**xml.sequenceOffset=120

Table 5.100: SwRecordLayoutGroupContent

For `CURVE`, `MAP`, etc. the iteration shall be performed along the input axis.

Primitive	**AxisIndexType**
Package	M2::MSR::DataDictionary::RecordLayout
Note	This meta-class specifies an axis in a curve/map data object. The index satisfies the following convention: • 0 output "axis" • 1 input axis 1 (X input axis e.g. of a CURVE) • 2 input axis 2 (Y input axis e.g. of a MAP) • 3 input axis 3 (Z input axis e.g. of a CUBOID) • 4 input axis 3 (Z4 input axis e.g. of a CUBE_4) • 5 input axis 3 (Z5 input axis e.g. of a CUBE_5) • 6..9 etc. The output "axis" provides access to the output value of the parameter. Note that this access is usually performed via an index according to the input axis. In addition to this, the Values STRING and ARRAY support specific iterations. **Tags:** xml.xsd.customType=AXIS-INDEX-TYPE xml.xsd.pattern=[0-9]+|STRING|ARRAY xml.xsd.type=string

Table 5.101: AxisIndexType

Primitive	**RecordLayoutIteratorPoint**
Package	M2::MSR::DataDictionary::RecordLayout
Note	This meta-class denotes a start / endpoint for the iteration of a SwRecordLayoutGroup. It can be an integer or one of the keywords MAX-TEXT-SIZE|ARRAY-SIZE. Note that negative numbers are counted backwards. Therefore e.g. -1 refers to the last value. **Tags:** xml.xsd.customType=RECORD-LAYOUT-ITERATOR-POINT xml.xsd.pattern=-?([0-9]+|MAX-TEXT-SIZE|ARRAY-SIZE) xml.xsd.type=string

Table 5.102: RecordLayoutIteratorPoint

[TPS_SWCT_01489] Standardized values of `SwRecordLayoutV.swRecordLayoutVProp` ⌈`SwRecordLayoutV.swRecordLayoutVProp` describes the type of values to be stored. The standardized values for `SwRecordLayoutV.swRecordLayoutVProp` are listed in Table 5.103.⌋*(RS_SWCT_03215)*

Property	Description
VALUE	The value of the axis for the current iterator point. This is e.g. the particular point on an input-axis, but also the particular character in a string.
COUNT	The amount of values of the axis.
LEFTDIFF	The difference to the previous axis point.
RIGHTDIFF	The difference to the next axis point.
DIST	The distance value of this axis in case of a fixed axis with distance specification.
SHIFT	The shift value of this axis in case of a fixed axis with shift/offset.
OFFSET	The offset value of this axis in case of a fixed axis with shift/offset.
SOURCE-ADR	The address of the source of this axis (Note that this does not apply to the value axis).
RESULT-ADR	The address of the result for this axis (note that this does not apply to input axis).
ADDRESS	The address of the axis point.

▽

△

`FILL`	Fill with the hex value specified as contents of `swRecordLayoutVFixValue`.
`RESERVED`	Position shall be ignored by MCD tools.
`IDENTIFIER`	An “identifier” is deposited at the position.
`FIXLEFTDIFF`	Difference between this and a fixed left-hand value specified in `swRecordLayoutVFixValue`.
`FIXRIGHTDIFF`	Difference between this and a fixed right-hand value specified in `swRecordLayoutVFixValue`.

Table 5.103: swRecordLayoutVProp

Figure 5.62 and Figure 5.63 illustrate most of these properties.

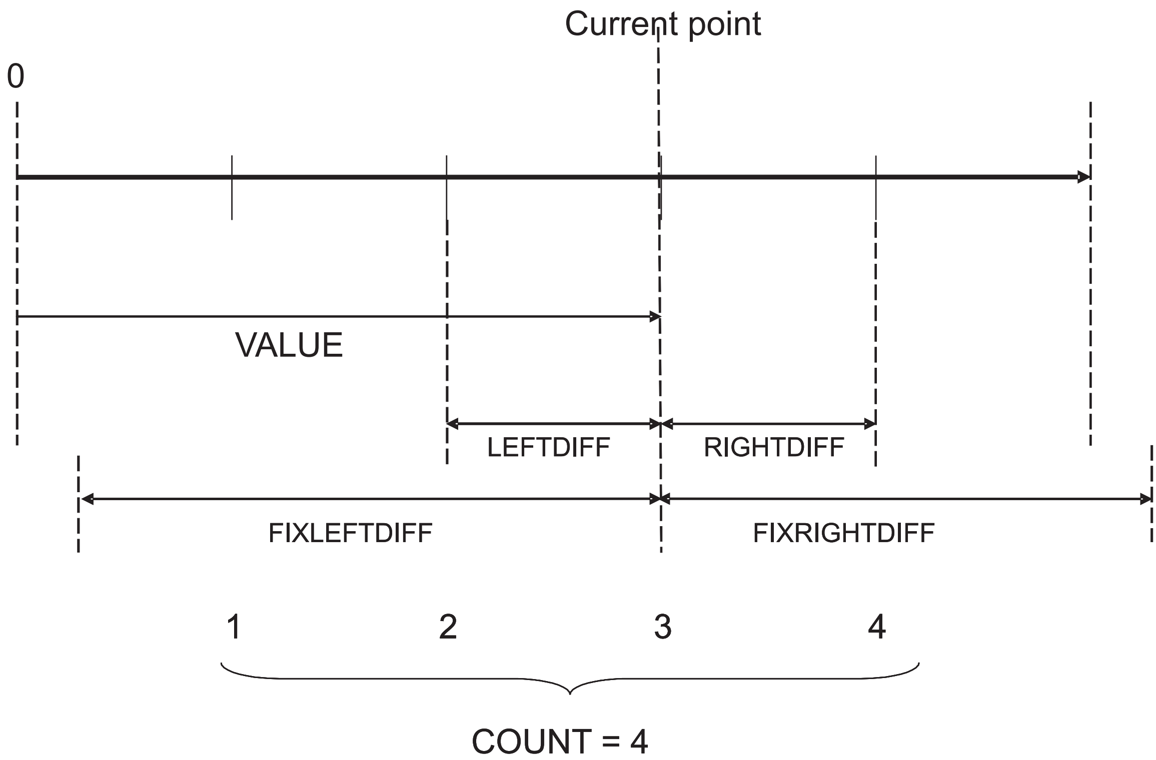


Figure 5.62: Values for swRecordLayoutVProp for individual axis

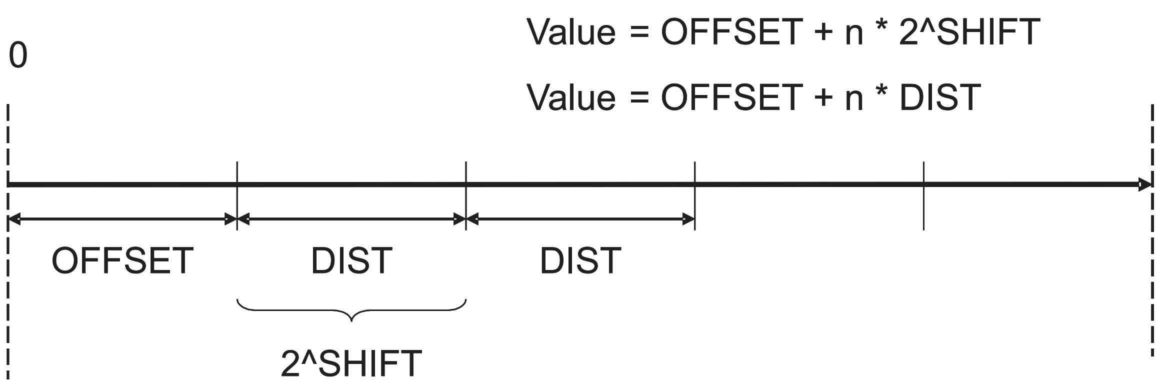


Figure 5.63: Values for swRecordLayoutVProp for fixed axis

[TPS_SWCT_01296] Different approaches of ASAM MCD-2MC and AUTOSAR with respect to `SwRecordLayout` ⌈ASAM MCD-2D specification (also known as

A2L, or ASAP) uses keywords in record layouts where MSR/AUTOSAR uses the more generic approach specified here.

It may happen that this generic approach cannot always be safely mapped to the A2L keywords. Therefore, SwRecordLayoutGroup.category can assist the conversion to the current A2L format.⌋*(RS_SWCT_03215)*

Primitive	**AsamRecordLayoutSemantics**
Package	M2::MSR::DataDictionary::RecordLayout
Note	This meta-class is used to denote the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2L keywords. It is possible to express the specific semantics of A2l RecordLayout keywords in SwRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. It is specified as NMTOKEN to reduce the direct dependency of ASAM an AUTOSAR standards. **Tags:** xml.xsd.customType=ASAM-RECORD-LAYOUT-SEMANTICS xml.xsd.type=NMTOKEN

Table 5.104: AsamRecordLayoutSemantics

The values of SwRecordLayoutGroup.category can, for example, be taken from the ASAM MCD 2D specification provided in [28]. Examples are:

- INDEX_INCR
- INDEX_DECR
- COLUMN_DIR
- ROW_DIR
- ALTERNATE_WITH_X
- ALTERNATE_WITH_Y
- ALTERNATE_CURVES

Note that there are keywords in A2L bound to a calibration parameter which in MSR/AUTOSAR are represented by the SwRecordLayout (DEPOSIT etc.).

The following XML fragment provides an example for a SwRecordLayout for a curve. Note that in this case recognizing the patterns represented by the A2L-Keywords (shown in XML-Comment) is pretty straight forward, even if the keywords were not provided in the SwRecordLayoutGroup.category.

```
<SW-RECORD-LAYOUT>
  <SHORT-NAME>RecordLayoutCurve</SHORT-NAME>
  <SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-V><!--  SRC_ADDR_X  -->
      <SHORT-LABEL>srcAdr</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-PROP>SOURCE-ADR</SW-RECORD-LAYOUT-V-PROP>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-V><!--  NO_AXIS_PTS_X -->
      <SHORT-LABEL>noOfAxisPts</SHORT-LABEL>
```

```
      <SW-RECORD-LAYOUT-V-PROP>COUNT</SW-RECORD-LAYOUT-V-PROP>
      <SW-RECORD-LAYOUT-V-INDEX>1</SW-RECORD-LAYOUT-V-INDEX>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-GROUP><!-- AXIS_PTS_X  -->
      <SHORT-LABEL>xPts</SHORT-LABEL>
      <CATEGORY>INDEX_INCR</CATEGORY>
      <SW-RECORD-LAYOUT-GROUP-AXIS>1</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SHORT-LABEL>xPt</SHORT-LABEL>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS> <!--
            AXIS_PTS_X  -->
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-GROUP>
      <SHORT-LABEL>values</SHORT-LABEL><!-- FNC_VALUES -->
      <CATEGORY>COLUMN_DIR</CATEGORY>
      <SW-RECORD-LAYOUT-GROUP-AXIS>0</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SHORT-LABEL>value</SHORT-LABEL>
        <SW-RECORD-LAYOUT-V-AXIS>0</SW-RECORD-LAYOUT-V-AXIS><!-- FNC_VALUES
            -->
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
  </SW-RECORD-LAYOUT-GROUP>
</SW-RECORD-LAYOUT>
```

Listing 5.21: Example for RecordLayout of a curve

5.5.5.2 RecordLayouts and DataTypes

[TPS_SWCT_01837] Types for record layouts ⌈Because ParameterDataPrototypes have a ≪isOfType≫-relation to ApplicationDataTypes or ImplementationDataTypes, the related data types shall properly match to the details as specified in swDataDefProps.⌋*(RS_SWCT_03215)*

This is exemplified in figure 5.64.

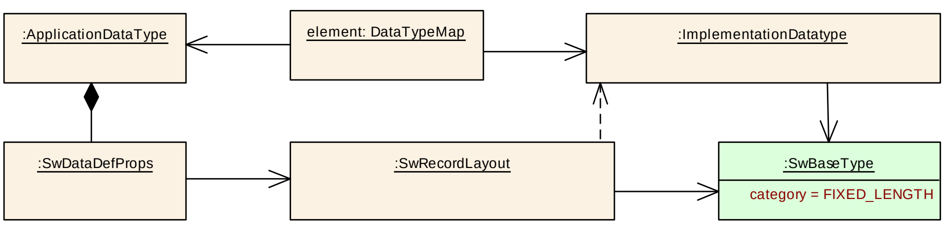


Figure 5.64: Dependency of AutosarDataTypes and SwRecordLayouts

[TPS_SWCT_01297] Compliance of ApplicationDataTypes or ImplementationDataTypes to swDataDefProps ⌈In order to maintain this compliance the following options exist:

- Manually create ImplementationDataTypes from corresponding ApplicationDataTypes and the referenced SwRecordLayouts
- Automatically create ImplementationDataTypes according to the existing definition of SwRecordLayouts. This could be performed by a model transformation according to the algorithm shown below.

⌋*()*

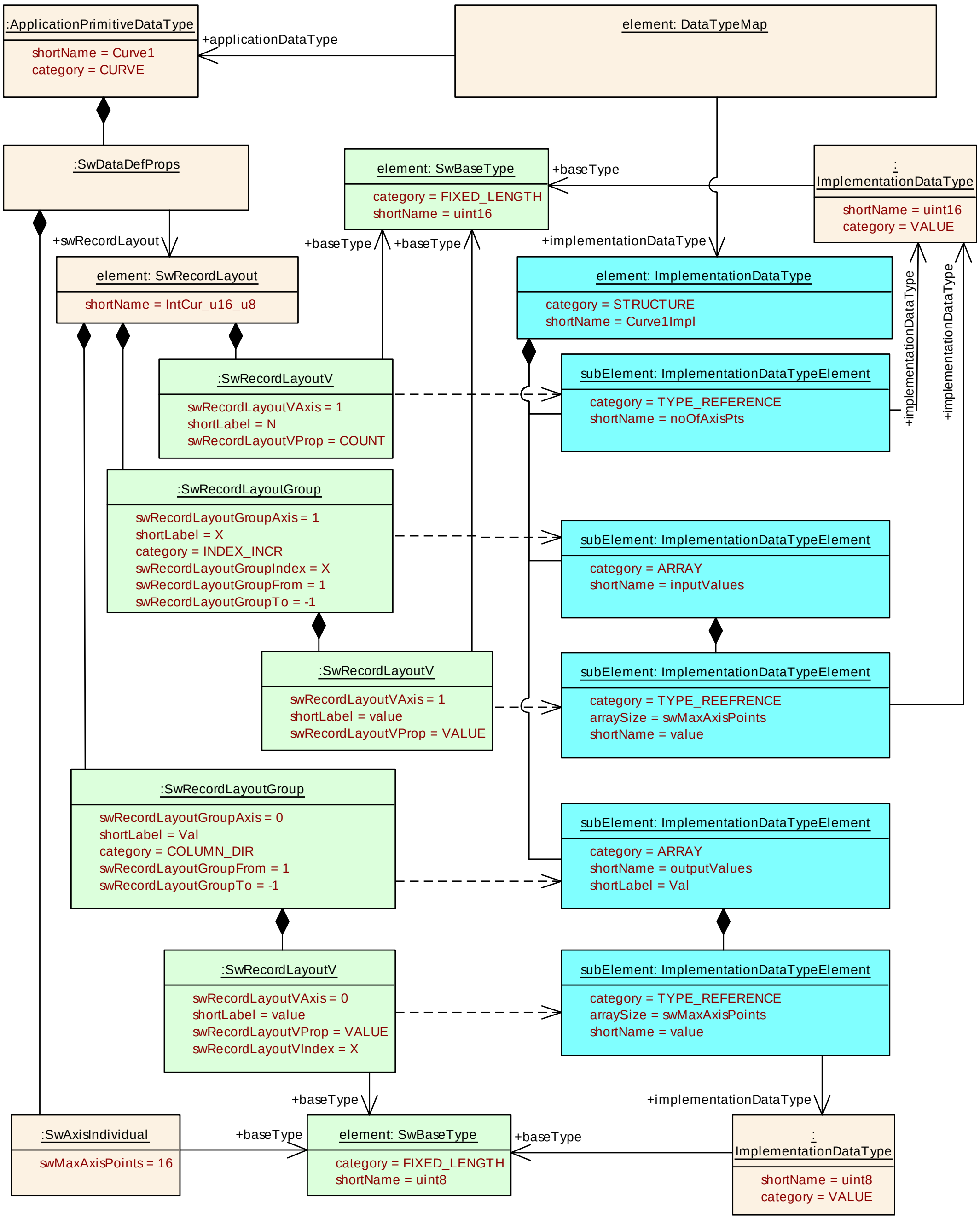


Figure 5.65: Curve implemented as two consecutive arrays

[TPS_SWCT_01298] Computing SwRecordLayout from ImplementationDataTypes is not possible ⌈Note that computing SwRecordLayouts from ImplementationDataTypes is not really possible because the particular semantics of the components is not available (swRecordLayoutVProp).⌋*(RS_SWCT_03215)*

Figures 5.65, 5.66, 5.67, 5.68, and 5.70 illustrate how data types can be derived from SwRecordLayouts. Please note that the figures simplify some aspects of the actual modeling. In particular, aggregations of SwDataDefProps are left out for the sake of visual clarity.

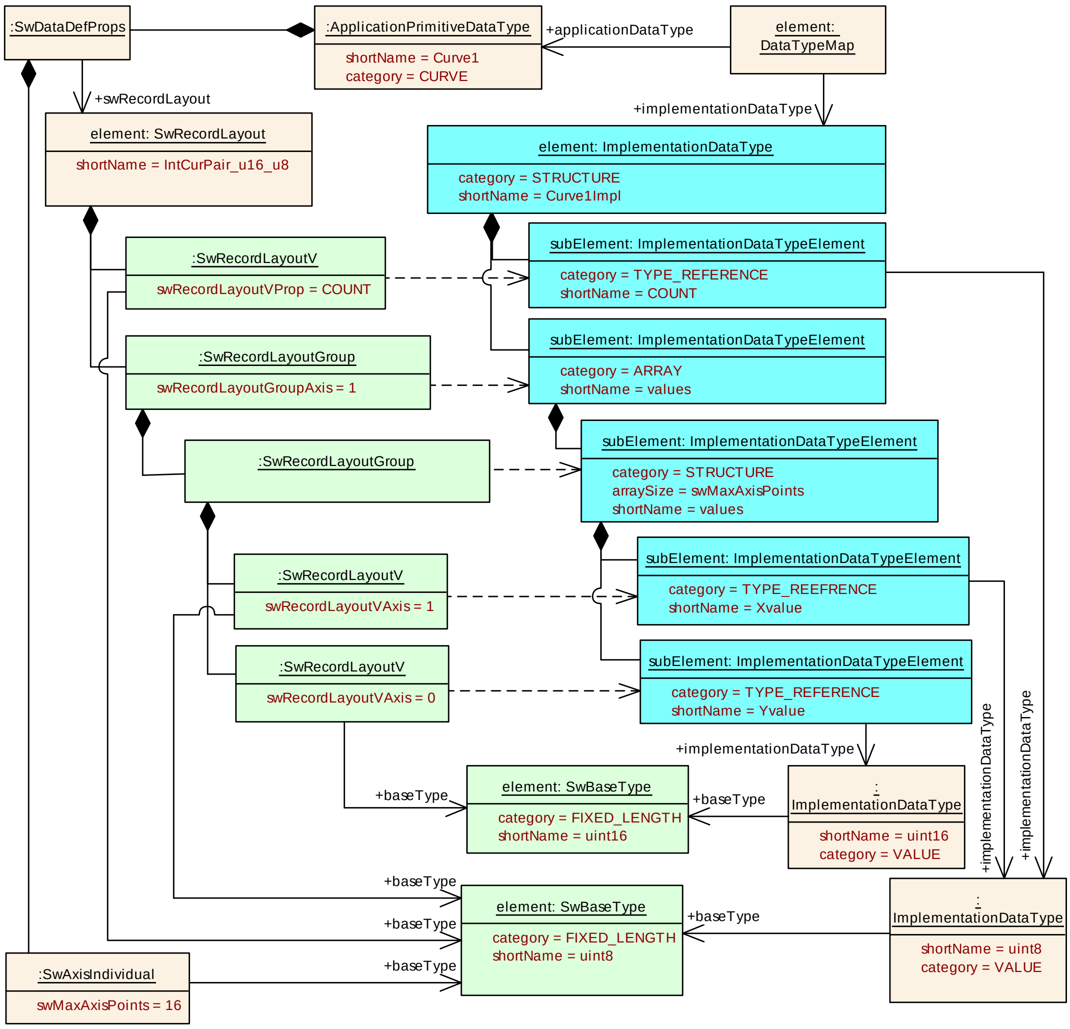


Figure 5.66: Curve implemented as array of value pairs

Note that in each of these diagrams, the "blue" data types are derived from the record layout.

These diagrams illustrate in particular the fact that on the level of ApplicationDataType even complex entities such as curves and maps appear primitive data types. The inner details of such entities are handled e.g. by service libraries.

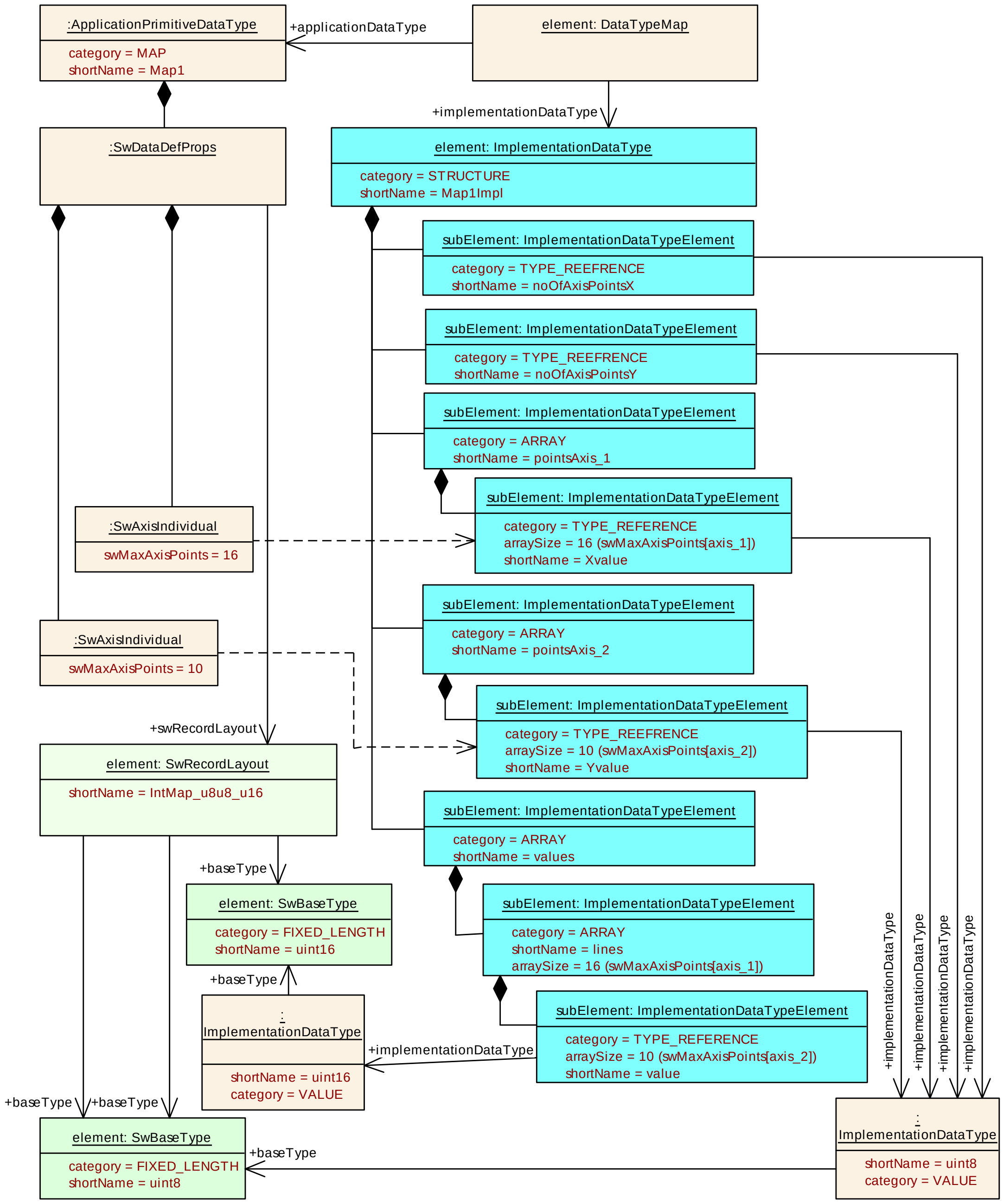


Figure 5.67: Record layout and data type for a map

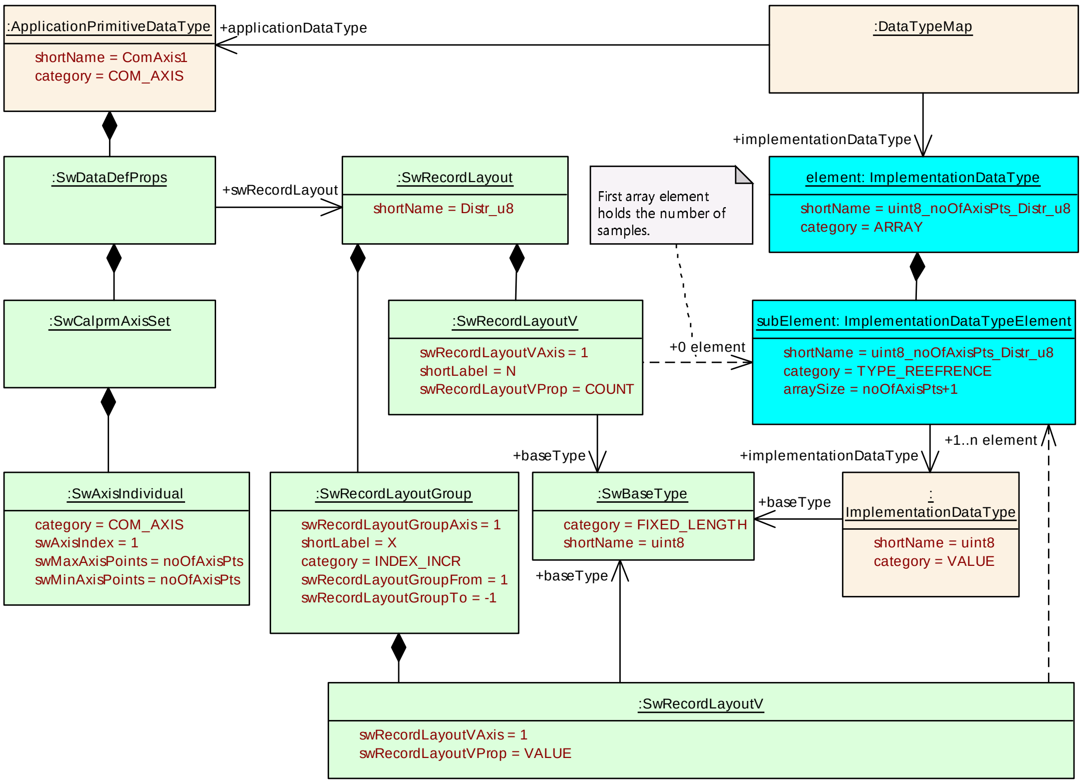


Figure 5.68: Record layout for the definition of a group axis

The algorithm to generate the desired data types is illustrated in the following two diagrams.

We create an ImplementationDataType for each ApplicationDataType. Figure 5.69 illustrates how to map the details.

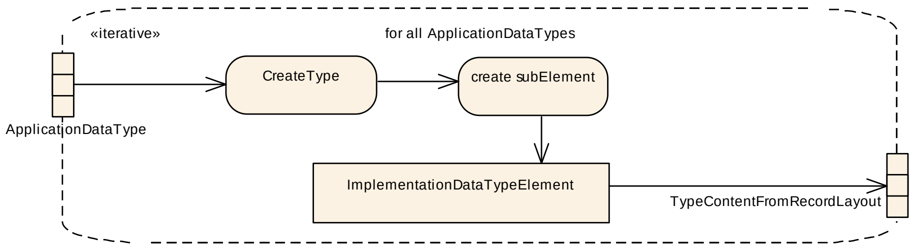


Figure 5.69: algorithm to map the details of an application data type to the corresponding implementation data type according to the record layout

[TPS_SWCT_01299] Relation of swRecordLayoutGroup to subElement ⌈For each swRecordLayoutGroup an appropriate subElement shall be created.

The algorithm shall be recursively applied to the newly created ImplementationDataTypeElements. As the record layout groups are nested, this recursion yields the complete structure in the ImplementationDataType.⌋*(RS_SWCT_03215)*

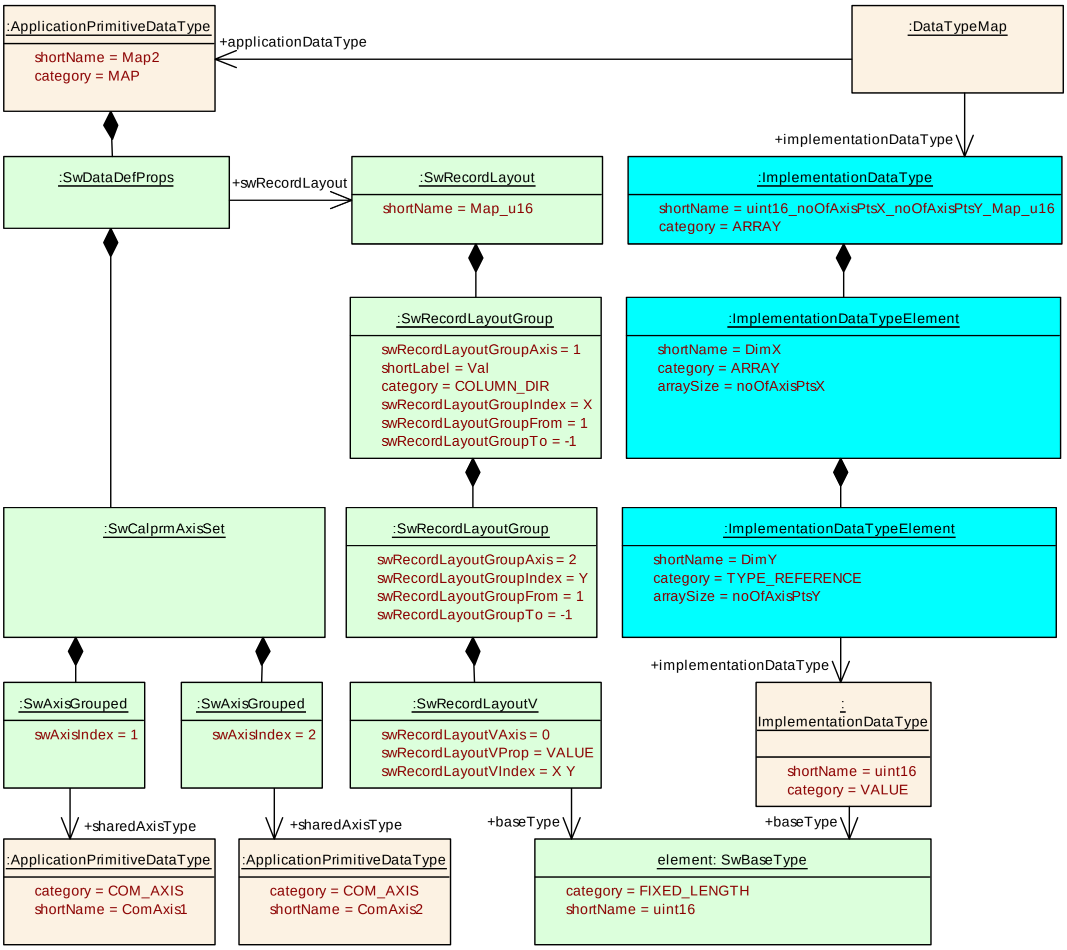


Figure 5.70: Record layout for the definition of a map implemented by an array data type

Please note that the refinement of the sub element happens according to the approach sketched in figure 5.71.

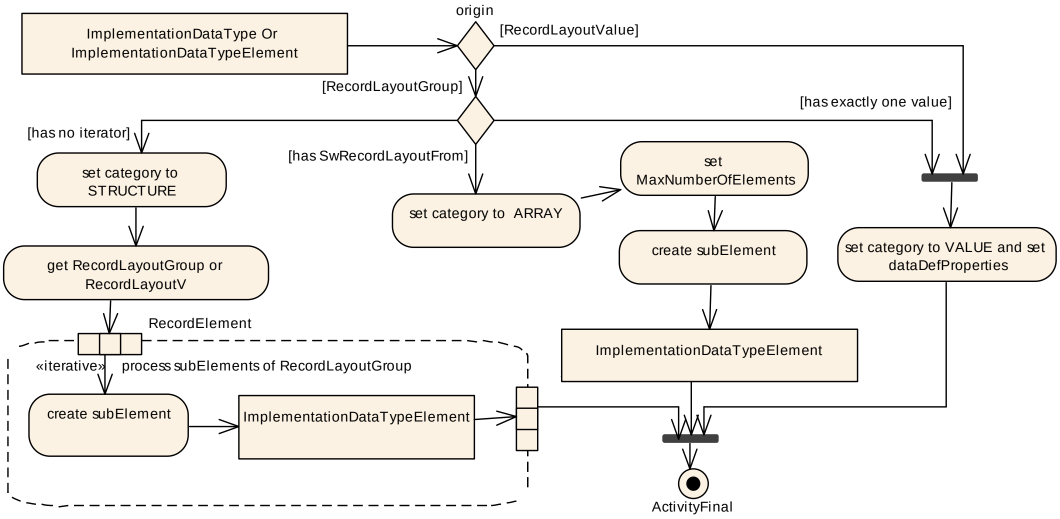


Figure 5.71: refining subElements

5.5.5.3 Record Layouts and Interpolation Routines

[TPS_SWCT_01300] Relationship between record layouts and interpolation routines ⌈The relationship between record layouts and interpolation routines can be specified in `InterpolationRoutineMappingSet`.

The interpolation routine is represented as `BswModuleEntry` and implements a particular interpolation method which is denoted in the value of `InterpolationRoutine.shortLabel`.

The intended interpolation method is denoted in the value of attribute `SwDataDefProps.swInterpolationMethod`.⌋*(RS_SWCT_03215)*

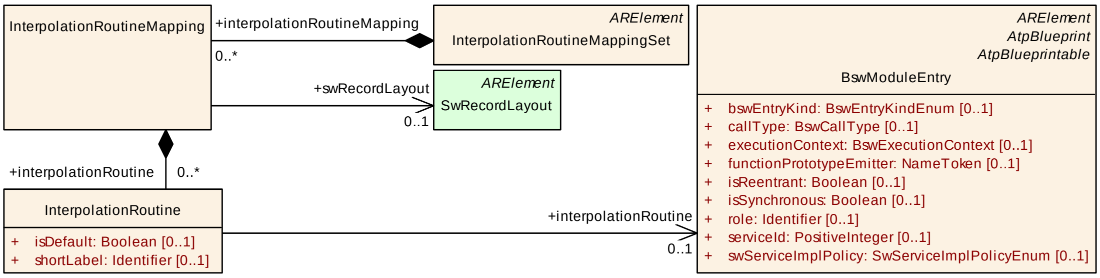


Figure 5.72: Mapping of Record Layouts and Interpolation Routines

Class	**InterpolationRoutineMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutineMappingSet			
Note	This meta-class specifies a set of interpolation routine mappings. **Tags:**atp.recommendedPackage=InterpolationRoutineMappingSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
interpolationRoutineMapping	InterpolationRoutineMapping	*	aggr	This specifies one particular mapping of recordlayout and its matching interpolationRoutines.

Table 5.105: InterpolationRoutineMappingSet

Class	**InterpolationRoutineMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutineMappingSet			
Note	This meta-class provides a mapping between one record layout and its matching interpolation routines. This allows to formally specify the semantics of the interpolation routines. The use case is such that the curves/Maps define an interpolation method. This mapping table specifies which interpolation routine implements methods for a particular record layout. Using this information, the implementer of a software-component can select the appropriate interpolation routine.			
Base	*ARObject*			
Aggregated by	InterpolationRoutineMappingSet.interpolationRoutineMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	InterpolationRoutineMapping			
interpolation Routine	InterpolationRoutine	*	aggr	This is one particular interpolation routine which is mapped to the record layout.
swRecord Layout	SwRecordLayout	0..1	ref	This refers to the record layout which is mapped to interpolation routines.

Table 5.106: InterpolationRoutineMapping

Class	**InterpolationRoutine**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutine MappingSet			
Note	This represents an interpolation routine taken to evaluate the contents of a curve or map against a specific input value.			
Base	*ARObject*			
Aggregated by	InterpolationRoutineMapping.interpolationRoutine			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
interpolation Routine	BswModuleEntry	0..1	ref	This specifies a BswModuleEntry which implements the current interpolation method for the given record layout. **Tags:**xml.sequenceOffset=30
isDefault	Boolean	0..1	attr	This attribute specifies whether the enclosing InterpolationRoutine is considered the default in the context (defined by the System Template) of a given collection InterpolationRoutineMapping that owns the enclosing InterpolationRoutine. **Tags:**xml.sequenceOffset=20
shortLabel	Identifier	0..1	attr	This is the name of the interpolation method which is implemented by the referenced bswModuleEntry. It corresponds to swInterpolationMethod in SwDataDef Props. **Tags:**xml.sequenceOffset=10

Table 5.107: InterpolationRoutine

5.5.6 Display Presentation

[TPS_SWCT_01756] Semantics of SwDataDefProps.displayPresentation
⌈The attribute SwDataDefProps.displayPresentation is used to control the presentation of data within measurement and calibration tools.

When such a tool displays a series of measurement values its useful to indicate to the displaying tool whether the series of measurement values can be seen as a continuous graph or as a set of discrete values, i.e. step-wise.

For instance, a continuous graph is appropriate for the case that the values do not bounce arbitrarily within one measurement cycle, e.g. a temperature variable.⌋*()*

On the other hand, a discrete handling is correct if each value of the measured variable has a distinct meaning and therefore may arbitrarily change within one measurement cycle, e.g. a state variable.

Another use case is the indication of how an ECU utilizes a DataPrototype of category CURVE, MAP, or CUBOID to determine a single value out of one or several working points in axis.

This can be either done via interpolation between the sampling points on each axis or without interpolation by taking the nearest sampling point.

The first option requires the continuous representation for the determined value in the displaying tool whereas the second option expects a discrete handling of the determined value.

[constr_1592] Definition of SwDataDefProps.displayPresentation depending on the capabilities of the data type ⌈The definition of a SwDataDefProps.displayPresentation according to [constr_1288] and [constr_1289] shall only be applied for a DataPrototype of category ARRAY if the corresponding ApplicationArrayDataType or ImplementationDataType of category ARRAY supports the specification of a SwDataDefProps.displayPresentation.

This rule shall be imposed **at any time in the workflow**.⌋*()*

[constr_1602] Definition of SwDataDefProps.displayPresentation depending on the capabilities of the element ⌈The definition of a SwDataDefProps.displayPresentation according to [constr_1007] and [constr_1009] is only supported for an ApplicationArrayDataType or an ImplementationDataType of category ARRAY if the aggregated ApplicationArrayDataType.element or ImplementationDataType.subElement also supports the specification of a SwDataDefProps.displayPresentation.

This rule shall be imposed **at any time in the workflow**.⌋*()*

[TPS_SWCT_01757] Not-applicable scenario for presentationContinuous ⌈If the semantics of the DataPrototype is described by means of a CompuMethod of category TEXTTABLE, BITFIELD_TEXTTABLE or TAB_NOINTP the option to set attribute displayPresentation is meaningless because the step-wise change of data is an intrinsic property of the data object.⌋*()*

[TPS_SWCT_01758] Applicable value range of SwDataDefProps.displayPresentation ⌈If the semantics of a DataPrototype is described by means of a CompuMethod of category IDENTICAL, LINEAR, RAT_FUNC the attribute SwDataDefProps.displayPresentation describes the presentation of data for the complete value range.

If the semantics of a DataPrototype is described by means of a CompuMethod of category SCALE_LINEAR_AND_TEXTTABLE or SCALE_RATIONAL_AND_TEXTTABLE the attribute SwDataDefProps.displayPresentation describes the presentation of data only for the value range outside the TEXTTABLE values.⌋*()*

Enumeration	**DisplayPresentationEnum**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	This meta-class represents the ability to provide values for controlling the presentation of data within measurement and calibration tools.
Aggregated by	SwDataDefProps.displayPresentation
Literal	*Description*
presentation Continuous	The presentation of data shall form a continuous graph between data points. **Tags:**atp.EnumerationLiteralIndex=0
presentation Discrete	The presentation of data shall be step-shaped between data points. **Tags:**atp.EnumerationLiteralIndex=1

Table 5.108: DisplayPresentationEnum

5.6 Specification of Constant Values

5.6.1 Overview

[TPS_SWCT_01177] Assignment of constant values ⌈Constant values can be assigned to a meta-class by aggregating the meta-class `ValueSpecification`. This aggregation can be used in two ways:

1. by referencing to a reusable `ConstantSpecification` which contains another `ValueSpecification`
2. or through an inline aggregation of a value specification of various kind.

⌋*(RS_SWCT_03175)*

Class	**ConstantSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specification of a constant that can be part of a package, i.e. it can be defined stand-alone. **Tags:**atp.recommendedPackage=ConstantSpecifications			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
valueSpec	ValueSpecification	0..1	aggr	Specification of an expression leading to a value for this constant.

Table 5.109: ConstantSpecification

[constr_1917] Existence of `ConstantSpecification.valueSpec` ⌈For each `ConstantSpecification`, the aggregation of `ValueSpecification` in the role `valueSpec` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

Class	*ValueSpecification* (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Base class for expressions leading to a value which can be used to initialize a data object.			
Base	*ARObject*			
Subclasses	*AbstractRuleBasedValueSpecification*, ApplicationValueSpecification, *CompositeValueSpecification*, ConstantReference, NotAvailableValueSpecification, NumericalValueSpecification, ReferenceValueSpecification, TextValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	Type	Mult.	Kind	Note
shortLabel	Identifier	0..1	attr	This can be used to identify particular value specifications for human readers, for example elements of a record type.

Table 5.110: ValueSpecification

Class	*CompositeValueSpecification* (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This abstract meta-class acts a base class for ValueSpecifications that have a composite form.			
Base	*ARObject*, *ValueSpecification*			
Subclasses	ArrayValueSpecification, RecordValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, CompositeRuleBasedValueSpecification.argument, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 5.111: CompositeValueSpecification

Class	ArrayValueSpecification
Package	M2::AUTOSARTemplates::CommonStructure::Constants
Note	Specifies the values for an array.
Base	*ARObject*, *CompositeValueSpecification*, *ValueSpecification*

▽

△

Class	ArrayValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, CompositeRuleBasedValueSpecification.argument, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
element (ordered)	ValueSpecification	*	aggr	The value for a single array element. All ValueSpecifications aggregated by ArrayValueSpecification shall have the same structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element, element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
intendedPartialInitializationCount	PositiveInteger	0..1	attr	This attribute shall only have a meaning for dynamic arrays and shall be taken as a sanity check: the number filled in the attribute shall be identical to the number of ArrayValueSpecification.element. If the attribute does not exist it means that no partial initialization is intended.

Table 5.112: ArrayValueSpecification

Class	RecordValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies the values for a record.			
Base	*ARObject*, *CompositeValueSpecification*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, CompositeRuleBasedValueSpecification.argument, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	RecordValueSpecification			
field (ordered)	ValueSpecification	*	aggr	The value for a single record field. This could also be mapped explicitly to a record element of the data type using the shortName of the ValueSpecification. But this would introduce a relationship to the data type that is too strong. As of now, it is only important that the structure of the data type matches the structure of the Value Specification independently of the shortNames. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=field, field.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 5.113: RecordValueSpecification

[constr_1918] Existence of RecordValueSpecification.field ⌈For each RecordValueSpecification, the aggregation of ValueSpecification in the role field shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	TextValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	The purpose of TextValueSpecification is to define the labels that correspond to enumeration values.			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, MetaDataItem.metaDataItemType, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
value	VerbatimString	0..1	attr	This is the value itself. Note that vt uses the | operator to separate the values for the different bitfield masks in case that the semantics of the related DataPrototype is described by means of a BITFIELD_TEXTTABLE in the associated CompuMethod.

Table 5.114: TextValueSpecification

[constr_1919] Existence of TextValueSpecification.value ⌈For each TextValueSpecification, attribute value shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	NumericalValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	A numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula.			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	Numerical	0..1	attr	This is the value itself. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 5.115: NumericalValueSpecification

[constr_1920] Existence of `NumericalValueSpecification.value` ⌈For each `NumericalValueSpecification`, attribute `value` shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	ReferenceValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies a reference to a data prototype to be used as an initial value for a pointer in the software.			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
referenceValue	DataPrototype	0..1	ref	The referenced data prototype.

Table 5.116: ReferenceValueSpecification

[constr_1921] Existence of `ReferenceValueSpecification.referenceValue` ⌈For each `ReferenceValueSpecification`, attribute `referenceValue` shall exist **at the time when the contract phase generation is executed**.⌋*()*

Figure 5.73 shows the specialized subclasses of `ValueSpecification` which allow defining values for different use cases.

[TPS_SWCT_01178] Specialized subclasses of ValueSpecification ⌈The use case for specialized subclasses of ValueSpecification are:

- Reference to a constant (which is actually a reusable value specification) by means of a ConstantReference.
- TextValueSpecification
- NumericalValueSpecification
- ArrayValueSpecification
- RecordValueSpecification
- ApplicationValueSpecification: this can be used to specify the value of Compound Primitive Data Types (see [TPS_SWCT_01179]) such as curves and maps. It is also possible to use this in general (e.g. for a primitive calibration value) for the specification of a value of a DataPrototype typed by an ApplicationDataType.

  Note that ApplicationValueSpecification is modeled along the example of ASAM CDF (for more information please refer to [32]).
- reference to a DataPrototype: this can be used to describe initial values for pointer variables in the basic software. One use case is the exchange of data descriptions used to access calibration data for software emulation methods (see [6] for details).
- ApplicationRuleBasedValueSpecification, inside an ArrayValueSpecification, see [constr_1779].
- NumericalRuleBasedValueSpecification, inside an ArrayValueSpecification, see [constr_1779].
- CompositeRuleBasedValueSpecification, inside an ArrayValueSpecification, see [constr_1779].

⌋*(RS_SWCT_03175)*

It's important to understand that although the name of the meta-class TextValueSpecification suggests that it is the preferred way for the definition of an invalidValue or initValue of a VariableDataPrototype/ParameterDataPrototype typed by an ApplicationPrimitiveDataType of category STRING the TextValueSpecification actually has a different purpose (as defined by [constr_1284]).

[constr_1284] Limitation of the use of TextValueSpecification ⌈TextValueSpecification shall **only** be used in the context of an AutosarDataType that references a CompuMethod in the role ImplementationDataType.swDataDefProps.compuMethod of category TEXTTABLE and BITFIELD_TEXTTABLE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

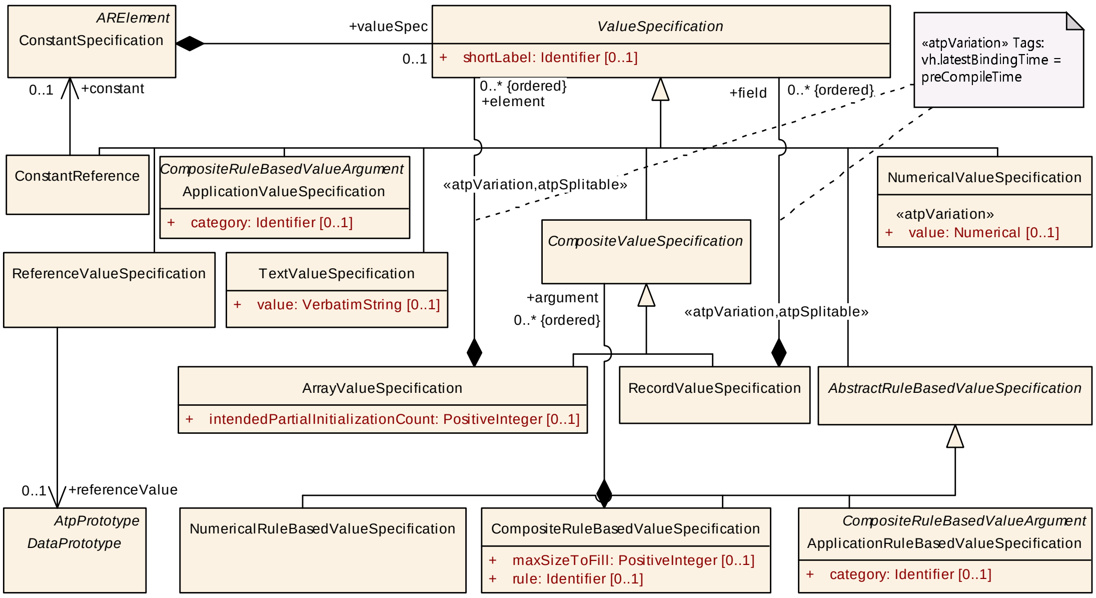


Figure 5.73: Summary of ValueSpecification

In other words, the purpose of TextValueSpecification is to define the labels that correspond to enumeration values.

The constraints [constr_1225] and [constr_1284] correspond to each other such that [constr_1225] demands the usage of TextValueSpecification for the definition of labels for enumeration values while [constr_1284] says that the definition of labels for enumeration values is the only use case for TextValueSpecification.

Note that ValueSpecification does not inherit from any data type. This would cause a redundancy[20] in the meta-model since the intended data type of a given ValueSpecification is already determined by the context in which it is aggregated.

Nonetheless, the intended data type imposes a certain constraint on the content of a ValueSpecification:

[TPS_SWCT_01838] ValueSpecification shall fit into data type ⌈An instance of ValueSpecification which is used to assign a value to a software object typed by an AutosarDataType shall fit into this AutosarDataType without losing information.⌋*()*

For example, it is not allowed to assign the numerical value “1.5” as initial value to a data prototype typed by an ImplementationDataType which has an integer base type.

[20]For example, “1” can be taken as a constant value for many data types. If the ValueSpecification were instead referring to a specific AutosarDataType it would be necessary to define a “1” for every single AutosarDataType this value is supposed to be used in combination with.

[constr_1271] RecordValueSpecification.fields shall be identical to the number of ApplicationRecordDataType.elements ⌈The initialization of a DataPrototype typed by an ApplicationRecordDataType by means of a RecordValueSpecification shall exactly match the structure of the ApplicationRecordDataType.

For this means, it is required that the number of RecordValueSpecification.fields shall be identical to the number of ApplicationRecordDataType.elements.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1272] RecordValueSpecification.fields shall be identical to the number of subElements of ImplementationDataType of category STRUCTURE ⌈The initialization of an DataPrototype typed by an ImplementationDataType of category STRUCTURE by means of a RecordValueSpecification shall exactly match the structure of the ImplementationDataType of category STRUCTURE.

For this means, it is required that the number of RecordValueSpecification.fields shall be identical to the number of ImplementationDataType.subElements.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

If the corresponding ApplicationRecordElement is typed by an ApplicationRecordDataType then the comparison of structural compliance between ApplicationRecordDataType and ValueSpecification shall continue beyond the encountered NotAvailableValueSpecification.

Class	**NotAvailableValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class provides the ability to specify a ValueSpecification to state that the respective element is not available. This ability is needed to support the existence of ApplicationRecordElements where attribute isOptional ist set to the value true. **Tags:**atp.Status=draft			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**NotAvailableValueSpecification**			
defaultPattern	PositiveInteger	0..1	attr	The content of this attribute shall be used to initialize gaps in the memory occupied by a structured data type in the case that an NotAvailableValueSpecification is used. Note that this pattern is only applied during initialization!

Table 5.117: NotAvailableValueSpecification

For deeply nested composite data types (including `ImplementationDataType`s created in response to the existence of a `Compound Primitive Data Type`) [constr_1271], [constr_1272], and [constr_1273] shall be applied recursively according to the nature of the given nesting levels. For the "leaf" elements [TPS_SWCT_01838] applies.

Please find more information about the creation of rule-based `ValueSpecification`s in section 5.6.9.

5.6.2 Reference to Constant

Note the specific meaning of `ConstantReference`: it passes the definition of the value on to a `ConstantSpecification` that is defined as part of an AUTOSAR `ARPackage`.

Class	**ConstantReference**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Instead of defining this value inline, a constant is referenced.			
Base	*ARObject*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
constant	ConstantSpecification	0..1	ref	The referenced constant.

Table 5.118: ConstantReference

[constr_1930] Existence of `ConstantReference.constant` ⌈For each `ConstantReference`, attribute `constant` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

5.6.3 Constant Specification Mapping

[TPS_SWCT_01186] **ConstantSpecificationMapping** ⌈The ConstantSpecificationMapping is used to associate ValueSpecifications defined in the implementation domain with corresponding ValueSpecifications defined in the application domain.

To make this possible the ValueSpecification actually needs to be a ConstantReference.

The ConstantSpecification referenced by the ConstantReference is also the target of the references owned by ConstantSpecificationMapping.⌋ *()*

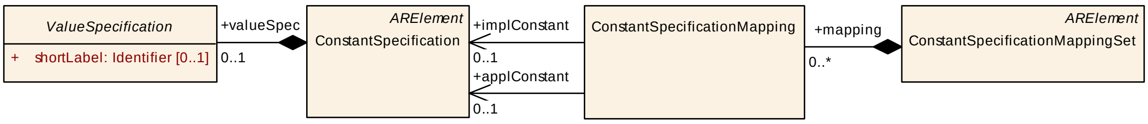


Figure 5.74: Constant Mapping

Class	**ConstantSpecificationMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to create an association of two ConstantSpecifications. One Constant Specification is supposed to be defined in the application domain while the other should be defined in the implementation domain. Hence the ConstantSpecificationMapping needs to be used where a ConstantSpecification defined in one domain needs to be associated to a ConstantSpecification in the other domain. This information is crucial for the RTE generator.			
Base	*ARObject*			
Aggregated by	ConstantSpecificationMappingSet.mapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applConstant	ConstantSpecification	0..1	ref	A ConstantSpecification defined in the application domain.
implConstant	ConstantSpecification	0..1	ref	A ConstantSpecification defined in the implementation domain.

Table 5.119: ConstantSpecificationMapping

[constr_1029] **ConstantSpecificationMapping and ConstantSpecification** ⌈It is required that one ConstantSpecification referenced from a ConstantSpecificationMapping needs to be defined in the application domain (applConstant) and the other referenced ConstantSpecification needs to be defined in the implementation domain (implConstant).

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋ *()*

[TPS_SWCT_01187] **ConstantSpecificationMappingSet referenced by the InternalBehavior** ⌈In most cases the meta-class ConstantSpecificationMappingSet will be referenced by the InternalBehavior. This ConstantSpecificationMappingSet contains the applicable ConstantSpecificationMappings.⌋ *()*

However, in some specializations the software-components will not have an InternalBehavior:

- **[TPS_SWCT_01840] A ParameterSwComponentType references a ConstantSpecificationMappingSet** ⌈ParameterSwComponentType: here, the ConstantSpecificationMappingSet is directly associated by the ParameterSwComponentType.⌋*()*
- **[TPS_SWCT_01841] A NvBlockSwComponentType references a ConstantSpecificationMappingSet** ⌈NvBlockSwComponentType: in this case, the ConstantSpecificationMappingSet is associated with the aggregated NvBlockDescriptor.⌋*()*

[constr_1931] Existence of ConstantSpecificationMapping.applConstant ⌈For each ConstantSpecificationMapping, the reference to meta-class ConstantSpecification in the role applConstant shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1932] Existence of ConstantSpecificationMapping.implConstant ⌈For each ConstantSpecificationMapping, the reference to meta-class ConstantSpecification in the role implConstant shall exist **at the time when the contract phase generation is executed**.⌋*()*

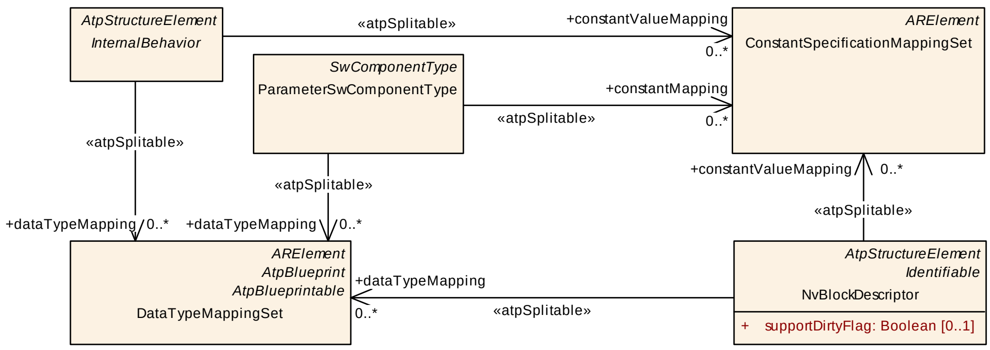


Figure 5.75: Relation between data type mapping and constant mapping

Class	**ConstantSpecificationMappingSet**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents the ability to map two ConstantSpecifications to each others. One Constant Specification is supposed to be described in the application domain and the other should be described in the implementation domain. **Tags:**atp.recommendedPackage=ConstantSpecificationMappingSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
mapping	ConstantSpecification Mapping	*	aggr	ConstantSpecificationMappings owned by the Constant SpecificationMappingSet.

Table 5.120: ConstantSpecificationMappingSet

5.6.4 Values for Variable-Size Array

Variable-size data types have the ability to change the number of valid elements at run-time. However, in many situations it is necessary to define an ArrayValueSpecification for such a data type.

An ArrayValueSpecification that can be used for a *variable-size array* data type needs to be able to handle the following cases:

- Full initialization of the entire array-data type. This case is identical to the creation of a ArrayValueSpecification for a fixed-size array data type.
- Provision of values for the **first n elements** of the *variable-size array*. This case is also known as *partial initialization*.
- Creation of an empty ArrayValueSpecification, i.e. an ArrayValueSpecification carries the semantics of intentionally initializing 0 elements of a *variable-size array*. Note the semantical difference between not initializing at all and intentionally initializing 0 elements of the *variable-size array*.

All the described cases shall be supported by AUTOSAR. As already described, the existence of an ArrayValueSpecification with the full number of elements is identical to the fixed size case. The "empty" case could be seen as a subset of the partial initialization.

The partial initialization of *variable-size array*s has two facets:

[TPS_SWCT_01793] Initialization of a *variable-size array* typed by an ImplementationDataType ⌈A *variable-size array* that is modeled by means of an ImplementationDataType is actually existing as a structure consisting of a *size indicator* and an *array that carries the payload*.

Therefore, the *partial initialization* shall be implemented by explicitly initializing the *size indicator* to a value between 0 and the applicable ImplementationDataTypeElement.arraySize and provide the corresponding number of ValueSpecifications for the *payload*.⌋*(RS_SWCT_03175, RS_SWCT_03181)*

[TPS_SWCT_01794] Initialization of a *variable-size array* typed by an ApplicationArrayDataType ⌈A *variable-size array* that is modeled by means of an ApplicationArrayDataType where attribute arraySizeSemantics is set to variableSize does not contain any size-indicator element and therefore requires a different approach for *partial initialization*.

For this purpose, ArrayValueSpecification.intendedPartialInitializationCount shall be used for the specification of the number of elements that shall be initialized.⌋*(RS_SWCT_03175, RS_SWCT_03181)*

The applicability of attribute ArrayValueSpecification.intendedPartialInitializationCount is limited to the use case of initializing a *variable-size array* typed by an ApplicationArrayDataType. AUTOSAR does not foresee any other use case for this attribute.

[constr_1712] Existence of attribute ArrayValueSpecification.intendedPartialInitializationCount ⌈An ArrayValueSpecification where attribute intendedPartialInitializationCount exists shall only be applied for the initialization of an ApplicationArrayDataType where attribute arraySizeSemantics is set to variableSize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1273] Rules for the initialization of ApplicationArrayDataType by means of ArrayValueSpecification ⌈The following rules apply for the initialization of a DataPrototype typed by an ApplicationArrayDataType by means of an ArrayValueSpecification:

- If the attribute ApplicationArrayDataType.element.arraySizeSemantics is set to **fixedSize** then the ArrayValueSpecification shall exactly match the structure of the ApplicationArrayDataType.

  This means that the number of ArrayValueSpecification.elements shall be identical to the value of ApplicationArrayDataType.element.maxNumberOfElements.

- If the attribute ApplicationArrayDataType.element.arraySizeSemantics is set to **variableSize** and the ArrayValueSpecification **does not define** attribute intendedPartialInitializationCount then ArrayValueSpecification shall **exactly** match the structure of the ApplicationArrayDataType.

  This means that the number of ArrayValueSpecification.elements shall be identical to the value of ApplicationArrayDataType.element.maxNumberOfElements.

- If the attribute ApplicationArrayDataType.element.arraySizeSemantics is set to **variableSize** and the ArrayValueSpecification specifies a value for attribute intendedPartialInitializationCount then ArrayValueSpecification shall contain **exactly** intendedPartialInitializationCount elements.

  This includes the case that the value of intendedPartialInitializationCount is set to 0 (i.e. "empty" initialization) and the case that the intendedPartialInitializationCount is set to the value of the respective ApplicationArrayElement.maxNumberOfElements (i.e. "full" initialization).

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1274] Rules for the initialization of array-shaped ImplementationDataType with a fixed size by means of ArrayValueSpecification ⌈The following rule applies for the initialization of a DataPrototype typed by an ImplementationDataType of category ARRAY where attribute ImplementationDataType.

subElement.arraySizeSemantics is set to fixedSize by means of an ArrayValueSpecification: the ArrayValueSpecification shall exactly match the structure of the ImplementationDataType.

This means that the number of ArrayValueSpecification.elements shall be identical to the value of ImplementationDataType.subElement.arraySize.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Please note that the initialization of an ImplementationDataType that represents a *variable-size array* is clarified in [TPS_SWCT_01793].

More details can be found in section 5.6.9.

5.6.5 Values for Compound Primitive Data Types

[TPS_SWCT_01180] Maximum possible size of Compound Primitive Data Type ⌈Note that if the size of the Compound Primitive Data Type (see [TPS_SWCT_01179]) (curve/map) is defined using an AttributeValueVariationPoint (in other words swMaxAxisPoints, swValueBlockSize, swValueBlockSizeMult dependent on the value of SwSystemconst) the initValue shall provide the maximum possible amount of values.⌋*(RS_SWCT_03216)*

In this case it is the responsibility of model author to ensure that the size of the specified init values matches the range of the involved system constants.

[TPS_SWCT_01839] Size of Compound Primitive Data Type is variant ⌈For Compound Primitive Data Types (see [TPS_SWCT_01179]) where the size is subject to variation the size of the specified initValues shall match the range of the involved SwSystemconst.⌋*(RS_SWCT_03216, RS_SWCT_03148)*

Class	**SwSystemconst**			
Package	M2::MSR::DataDictionary::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:**atp.recommendedPackage=SwSystemconsts			
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	SwSystemconst			
swDataDef Props	SwDataDefProps	0..1	aggr	This denotes the data definition properties of the system constant. This supports to express the limits and optionally a conversion within the internal to physical values by a compu method. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps xml.sequenceOffset=40

Table 5.121: SwSystemconst

[TPS_SWCT_01181] Bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst ⌈The processing tools shall take the lower part of the initValues in case the bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst.⌋*(RS_SWCT_03216, RS_SWCT_03148)*

The consequences of [TPS_SWCT_01181] are exemplified by Figure 5.76.

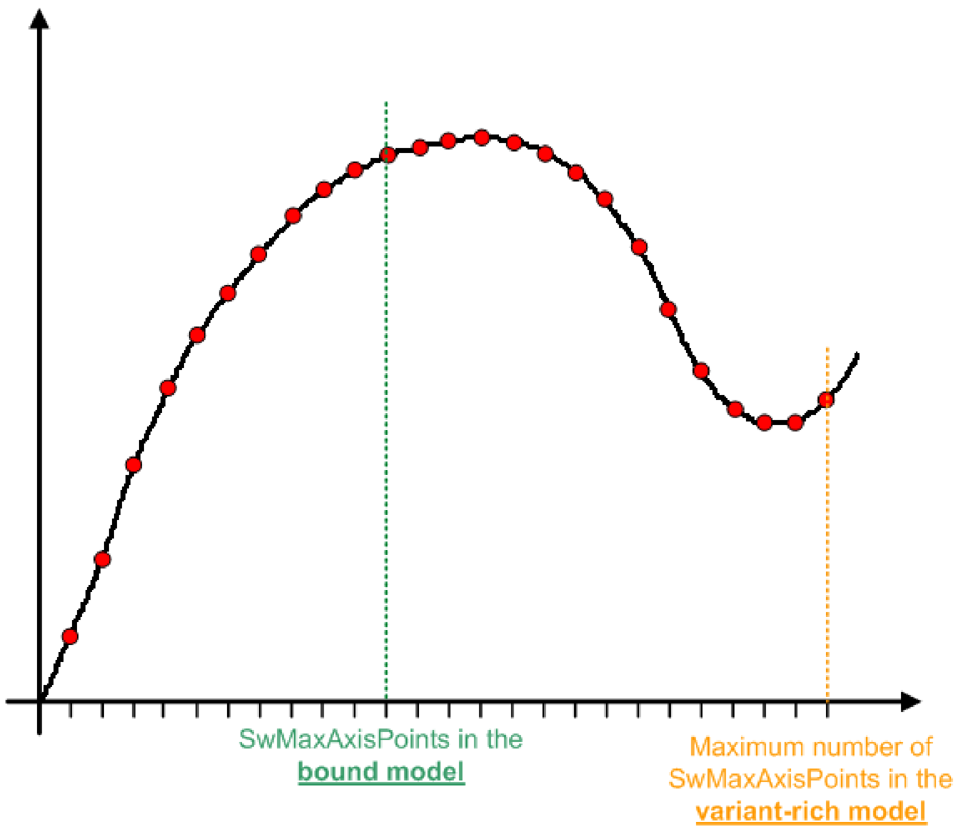


Figure 5.76: Explanation of swMaxAxisPoints

[constr_2050] Mandatory information of a SwAxisCont ⌈If the attribute swAxisCont is defined for an ApplicationValueSpecification the SwAxisCont shall define one swAxisIndex value and one swArraysize value per dimension, even in the case when the owning ApplicationValueSpecification defines only the content of a single dimensional object like a CURVE.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋()

[constr_2051] Mandatory information of a SwValueCont ⌈If the attribute swValueCont is defined for an ApplicationValueSpecification the SwValueCont shall always define the attribute swArraysize if the ApplicationValueSpecification is of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, or VAL_BLK.

This rule shall be imposed **at the time when the contract phase generation is executed.**⌋()

Class	**SwValueCont**			
Package	M2::MSR::CalibrationData::CalibrationValue			
Note	This metaclass represents the content of one particular SwInstance.			
Base	*ARObject*			
Aggregated by	ApplicationValueSpecification.swValueCont			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swArraysize	ValueList	0..1	aggr	This attribute defines the size of each dimension for compound primitives CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, VAL_BLK. For each dimension one value has to be defined, e.g. one in case of COM_AXIS and two or more in case of MAP. **Tags:**xml.sequenceOffset=40
swValuesPhys	SwValues	0..1	aggr	swValuesPhys represents the values in the physical domain. **Tags:**xml.sequenceOffset=50
unit	Unit	0..1	ref	This represents the physical unit of the provided values. **Tags:**xml.sequenceOffset=20
unitDisplayName	SingleLanguageUnitNames	0..1	aggr	This specifies how the physical units of the current value set shall be displayed in documents or in user interfaces of tools. **Tags:**xml.sequenceOffset=30

Table 5.122: SwValueCont

Please note that for multidimensional Compound Primitive Data Types (e.g. MAP) it is necessary to know the dimensions in order to be able to process the SwValues. [constr_2050] and [constr_2051] shall support a consistent handling of single and multidimensional Compound Primitive Data Types.

[constr_2052] Values of swArraysize and the number of values provided by swValuesPhys shall be consistent. ⌈swValuesPhys shall define as many values as the attribute swArraysize (if this attribute exists) defines.

In other words, in the bound model the number of descendants (v, or vf, or vt, or vtf) shall be identical to the number of elements of the related DataPrototype typed by an ApplicationPrimitiveDataType.

If several swArraysize values are provided, the values have to be multiplied in order to get the total number of swValuesPhys values.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

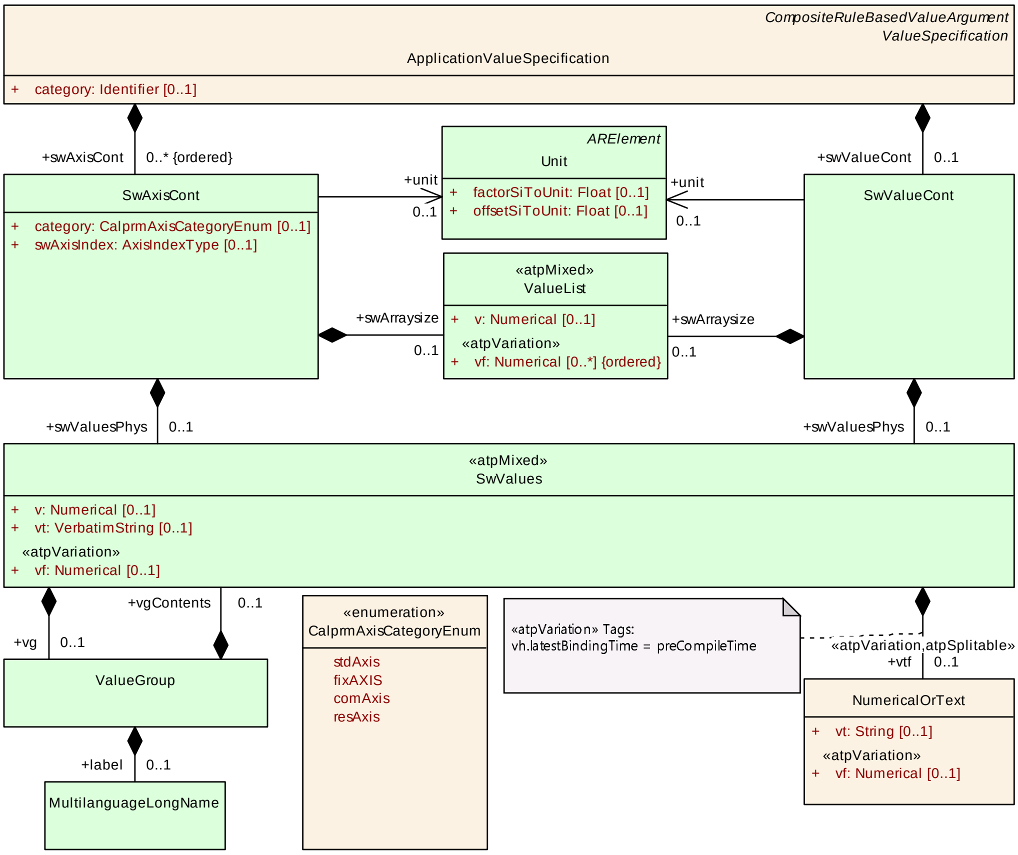


Figure 5.77: Definition of an ApplicationValueSpecification

Please note that case of Compound Primitive Data Types typically the attribute swValuesPhys defines more than one value. [constr_2051] and [constr_2052] shall enable a consistent handling of the swValuesPhys values regardless how many dimensions the related Compound Primitive Data Type defines.

If the ApplicationValueSpecification defines values for a Compound Primitive Data Type with more than one input axis the swArraysize gets mandatory to ensure the correct processing of the swValuesPhys values independent of the existence of SwValues.vg.

[TPS_SWCT_02001] Values of SwAxisCont with the category COM_AXIS, RES_AXIS are for display only ⌈In case of ApplicationValueSpecifications of category MAP, CUBOID, CUBE_4, CUBE_5, and CURVE it is possible that the SwAxisCont of axes can be omitted if the axis is of category COM_AXIS or RES_AXIS.

If SwAxisCont values exists in such cases for the axes these are for display purpose only because the related DataPrototype of the MAP, CUBOID, CUBE_4, CUBE_5, or CURVE does not hold the values of such axes. These are properties of the DataPrototype of the COM_AXIS or RES_AXIS.⌋()

Class	**ApplicationValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents values for DataPrototypes typed by ApplicationDataTypes (this includes in particular compound primitives). For further details refer to ASAM CDF 2.0. This meta-class corresponds to some extent with SW-INSTANCE in ASAM CDF 2.0.			
Base	*ARObject*, *CompositeRuleBasedValueArgument*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, CompositeRuleBasedValueSpecification.compoundPrimitiveArgument, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
category	Identifier	0..1	attr	Specifies to which category of ApplicationDataType this ApplicationValueSpecification can be applied (e.g. as an initial value), thus imposing constraints on the structure and semantics of the contained values, see [constr_1006] and [constr_2051].
swAxisCont (ordered)	SwAxisCont	*	aggr	This represents the axis values of a Compound Primitive Data Type (curve or map). The first swAxisCont describes the x-axis, the second swAxisCont describes the y-axis, the third swAxisCont describes the z-axis. In addition to this, the axis can be denoted in swAxisIndex.
swValueCont	SwValueCont	0..1	aggr	This represents the values of a Compound Primitive Data Type.

Table 5.123: ApplicationValueSpecification

Hence, values of the COM_AXIS itself are described by SwValueCont.

[constr_1243] NumericalOrText shall either define vf or vt ⌈Within the context of one NumericalOrText, **either** the attribute vf **or** the attribute vt shall be defined. The existence of both attributes at the same time is not permitted.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Class	NumericalOrText			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents the ability to yield either a numerical or a string. A typical use case is that two or more instances of this meta-class are aggregated with a VariationPoint where some instances yield strings while other instances yield numerical depending on the resolution of the binding expression.			
Base	*ARObject*			
Aggregated by	RuleArguments.vtf, SwValues.vtf			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
vf	Numerical	0..1	attr	This attribute represents the ability to provide a numerical value. The latest binding time of the VariationPoint shall be preCompileTime. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
vt	String	0..1	attr	This attribute represents the ability to provide a textual value. **Tags:**xml.sequenceOffset=20

Table 5.124: NumericalOrText

[constr_1519] Existence of attributes vs. category of ApplicationValueSpecification ⌈

	Attribute Existence per Category									
Attribute of ApplicationValueSpecification	VALUE	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP	CUBOID	CUBE_4	CUBE_5
swValueCont	D	D	D	D	D	D	D	D	D	D
swValueCont.unit	O	O	O	O	O	O	O	O	O	O
swValueCont.swValuesPhys	D	D	D	D	D	D	D	D	D	D
swValueCont.swArraysize	N/A	N/A	N/A	D	D	D	D	D	D	D
swAxisCont	N/A	N/A	N/A	N/A	D	D	D	D	D	D
swAxisCont.unit	N/A	N/A	N/A	N/A	O	O	O	O	O	O
swAxisCont.category	N/A	N/A	N/A	N/A	D	D	D	D	D	D
swAxisCont.swAxisIndex	N/A	N/A	N/A	N/A	D	D	D	D	D	D
swAxisCont.swArraysize	N/A	N/A	N/A	N/A	D	D	D	D	D	D
swAxisCont.swValuesPhys	N/A	N/A	N/A	N/A	D	O(1)	O(1)	O(1)	O(1)	O(1)

This rule shall be imposed at the time when the contract phase generation is executed.

⌋()

The following legend applies to the cells in the table contained in [constr_1519]:

D **Define** the attribute.

N/A Attribute is **not applicable** for usage in the scope of this element.

O **Optionally** define the attribute.

In addition to the primary cell legend the following annotations apply to the cells in the table contained in [constr_1375]:

(1) Optional if COM_AXIS or RES_AXIS is used, otherwise attribute shall exist.

[constr_10017] Existence of attribute SwAxisCont.category ⌈For each SwAxisCont, attribute category shall exist **at the time when the contract phase generation is executed.**⌋()

[constr_10040] Value of ApplicationValueSpecification.swAxisCont.category ⌈The value of attribute ApplicationValueSpecification.swAxisCont.category shall not be set to fixAXIS **at the time when the contract phase generation is executed.**⌋()

Class	**SwAxisCont**			
Package	M2::MSR::CalibrationData::CalibrationValue			
Note	This represents the values for the axis of a compound primitive (curve, map). For standard and fix axes, SwAxisCont contains the values of the axis directly. The axis values of SwAxisCont with the category COM_AXIS, RES_AXIS are for display only. For editing and processing, only the values in the related GroupAxis are binding.			
Base	*ARObject*			
Aggregated by	ApplicationValueSpecification.swAxisCont			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
category	CalprmAxisCategoryEnum	0..1	attr	This category specifies the particular axis types: • STD_AXIS • COM_AXIS • RES_AXIS (swArraysize necessary) **Tags:**xml.sequenceOffset=20
swArraysize	ValueList	0..1	aggr	For multidimensional compound primitivies (curve, map ...) it is necessary to know the dimensions.They are specified using swArraySize. • RES_AXIS **Tags:**xml.sequenceOffset=70
swAxisIndex	AxisIndexType	0..1	attr	This property allows to explicitly assign the axis contents to a particular axis. It is specified by numbers where 1 corresponds to the x-axis. It is also possible to derive the axis association from the sequence of the parent. **Tags:**xml.sequenceOffset=50
swValuesPhys	SwValues	0..1	aggr	swValuesPhys represents the values in the physical domain. **Tags:**xml.sequenceOffset=80
unit	Unit	0..1	ref	This represents the physical unit of the provided values. **Tags:**xml.sequenceOffset=30
unitDisplayName	SingleLanguageUnitNames	0..1	aggr	This represents the display name which is used for the physical unit of the axis. **Tags:**xml.sequenceOffset=40

Table 5.125: SwAxisCont

Rationale for the existence of [constr_10040]: the value fixAXIS for category indicates that the respective axis that is calculated out of the value of calibration parameters. Obviously, it does not make sense to initialize an axis of this kind and therefore it is excluded from the usage inside an ApplicationValueSpecification.

[constr_10018] Existence of attribute SwAxisCont.swAxisIndex ⌈For each SwAxisCont, attribute swAxisIndex shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_10019] Existence of attribute SwAxisCont.swValuesPhys ⌈For each SwAxisCont, attribute swValuesPhys shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	<<atpMixed>> **SwValues**			
Package	M2::MSR::CalibrationData::CalibrationValue			
Note	This meta-class represents a list of values. These values can either be the input values of a curve (abscissa values) or the associated values (ordinate values). In case of multidimensional structures, the values are ordered such that the lowest index runs the fastest. In particular for maps and cuboids etc. the resulting long value list can be subsectioned using Value Group. But the processing needs to be done as if vg is not there. Note that numerical values and textual values should not be mixed.			
Base	*ARObject*			
Aggregated by	SwAxisCont.swValuesPhys, SwValueCont.swValuesPhys, ValueGroup.vgContents			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
v	Numerical	0..1	attr	This is a non variant Value. It is provided for sake of Compatibility to ASAM CDF. **Tags:**xml.sequenceOffset=40
vf	Numerical	0..1	attr	This allows to specify the value as VariationPoint. It is distinguished to non variant for sake of compatibility to ASAM CDF 2.0. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
vg	ValueGroup	0..1	aggr	This allows to have intersections in the values in order to support specific rendering (eg. using stylesheets). For tools it is important that the v values are always processed in the same (flattened) order and the tool is able to interpret it without respecting vg. **Tags:**xml.sequenceOffset=50
vt	VerbatimString	0..1	attr	This represents the values of textual data elements (Strings). Note that vt uses the | to separate the values for the different bitfield masks in case that the semantics of the related DataPrototype is described by means of a BITFIELD_TEXTTABLE in the associated CompuMethod. **Tags:**xml.sequenceOffset=30

▽

△

Class	<<atpMixed>> **SwValues**			
vtf	NumericalOrText	0..1	aggr	This aggregation represents the ability to provide a value that is either numerical or text which existence is subject to variability. From the formal point of view, the aggregation needs to have the multiplicity 1 because SwValues is modelled with stereotype <<atpMixed>>. Nevertheless, the existence of vtf is optional and subject to constraints. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=vtf, vtf.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 5.126: SwValues

Class	**ValueGroup**			
Package	M2::MSR::CalibrationData::CalibrationValue			
Note	This element enables values to be grouped. It can be used to perform row and column-orientated groupings, so that these can be rendered properly e.g. as a table.			
Base	*ARObject*			
Aggregated by	SwValues.vg			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
label	MultilanguageLong Name	0..1	aggr	This label allows to give the valueGroup a particular name. It can be used if the Values are rendered as a table. **Tags:**xml.sequenceOffset=20
vgContents	SwValues	0..1	aggr	This represents the contents of the value group. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table 5.127: ValueGroup

Class	<<atpMixed>> **ValueList**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This is a generic list of numerical values.			
Base	*ARObject*			
Aggregated by	RuleBasedAxisCont.swArraysize, RuleBasedValueCont.swArraysize, SwAxisCont.swArraysize, SwServiceArg.swArraysize, SwValueCont.swArraysize			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
v	Numerical	0..1	attr	This is a particular numerical value without variation. **Tags:**xml.sequenceOffset=30

▽

△

Class	<<atpMixed>> **ValueList**			
vf (ordered)	Numerical	*	attr	This is one entry in the list of numerical values **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.roleElement=true xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

Table 5.128: ValueList

5.6.6 Values for TEXTTABLE

The creation of a ValueSpecification applied to a DataPrototype that is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE needs to make sure that the numerical value is represented in the CompuMethod.

This means that in this specific case, the initialization of a DataPrototype is typed by an ImplementationDataType is – contrary to the typical approach for a ValueSpecification applied in the context of an ImplementationDataType – actually done in the **physical domain** and thereby it can be ensured that the value provided in the ValueSpecification is represented in the CompuMethod.

For example, in a specific case the values from 0..10 are covered by corresponding literals of the applicable CompuMethod, what is the consequence if the value of the DataPrototype is set to the **numerical** value 20? This scenario should be avoided, **unless** the ValueSpecification represents an invalidValue (see section 5.4.2, specifically [constr_10196]).

[constr_1225] DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE ⌈If a DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE the applicable ValueSpecification shall be a TextValueSpecification.

In this case the value provided shall match to one of the applicable text values (vt, shortLabel, symbol) defined by the applicable CompuScales.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note that several attributes of meta-class CompuScale can be taken to describe the actual value. It is therefore necessary to clarify what happens if several of these attributes exist within the context of one CompuScale. This clarification can be found in [TPS_SWCT_01696].

5.6.7 Values for BITFIELD_TEXTTABLE

A CompuMethod of category BITFIELD_TEXTTABLE effectively defines a data type where single bits or a set of single bits have an individual meaning.

In other words, the definition of a ValueSpecification according to a CompuMethod of category BITFIELD_TEXTTABLE translates into the statement: "each unique value of attribute CompuScale.mask (because this is how the single bits or sets of single bits are identified) in the context of the enclosing CompuMethod delivers one contribution to the definition of the ValueSpecification".

[TPS_SWCT_01792] Initialization of a DataPrototype associated with a CompuMethod of category BITFIELD_TEXTTABLE ⌈The initialization of a DataPrototype associated with a CompuMethod of category BITFIELD_TEXTTABLE requires the simultaneous initialization of all single bits or sets of bits represented by the respective definition of the bitmask represented by attribute CompuScale.mask.

Access to the elements is possible by using bit-operations, therefore the initialization needs to be defined in a way that bit-operations can be used to apply the provided value.

In other words, the overall value shall be created out of bitwise or-ing (represented by the usage of the "|" symbol) contributions from the individual subsets.⌋*()*

Listing 5.22 shows the definition of an ApplicationValueSpecification that can be taken e.g. for the initialization of a DataPrototype typed by an ApplicationDataType that refers to a CompuMethod of category BITFIELD_TEXTTABLE.

```
<APPLICATION-VALUE-SPECIFICATION>
  <SHORT-LABEL>tirePressureInit</SHORT-LABEL>
  <CATEGORY>VALUE</CATEGORY>
  <SW-VALUE-CONT>
    <SW-VALUES-PHYS>
      <VT>problem_low_Pressure|rearRight_yes|rearLeft_yes|frontRight_no|
          frontLeft_no</VT>
    </SW-VALUES-PHYS>
  </SW-VALUE-CONT>
</APPLICATION-VALUE-SPECIFICATION>
```

Listing 5.22: Example for the definition of an ApplicationValueSpecification for BITFIELD_TEXTTABLE

Listing 5.23 shows the definition of an TextValueSpecification that can be taken e.g. for the initialization of a DataPrototype typed by an ImplementationDataType that refers to a CompuMethod of category BITFIELD_TEXTTABLE.

```
<TEXT-VALUE-SPECIFICATION>
  <SHORT-LABEL>tirePressureInit</SHORT-LABEL>
  <VALUE>problem_low_Pressure|rearRight_yes|rearLeft_yes|frontRight_no|
      frontLeft_no</VALUE>
</TEXT-VALUE-SPECIFICATION>
```

Listing 5.23: Example for the definition of a TextValueSpecification for BITFIELD_TEXTTABLE

Please note that areas of the initialized DataPrototype that are not covered by the bit-masks may contain arbitrary values. If this needs to be avoided, it is necessary to add "dummy" bit-masks to the semantically not relevant parts of the value of the DataPrototype.

5.6.8 Values for Strings

5.6.8.1 String is modeled by ApplicationDataType

[TPS_SWCT_01859] Specification of constant value for a String modeled by ApplicationDataType ⌈The specification of a constant value for a DataPrototype typed by an ApplicationPrimitiveDataType of category STRING shall be done by specifying an ApplicationValueSpecification. Inside the ApplicationValueSpecification, the concrete value shall be provided in ApplicationValueSpecification.swValueCont.swValuesPhys.vt.⌋*()*

An example of how a constant value for a string modeled as ApplicationDataType is contained in Listing 5.1.

5.6.8.2 String is modeled by ImplementationDataType

[TPS_SWCT_01860] Specification of constant value for a String modeled by ImplementationDataType ⌈The specification of a constant value for a DataPrototype typed by an ImplementationDataType that references an SwBaseType with a string encoding (e.g. UTF-8) shall be done on the basis of (a series of) NumericalValueSpecification.⌋*()*

An example of how a constant value for a string modeled by means of an ImplementationDataType is contained in Listing 5.4.

5.6.9 Specification of Values based on Rules

5.6.9.1 Support for primitive Data Types

[TPS_SWCT_01484] Meaning of ApplicationRuleBasedValueSpecification ⌈The purpose of the ApplicationRuleBasedValueSpecification is to provide means for a compact provision of values for DataPrototypes that otherwise would require a high volume (in terms of serialized ARXML) of e.g. initialization data.

ApplicationRuleBasedValueSpecification may be used for ApplicationArrayDataType, and also (if applicable) to the so-called Compound Primitive Data Types.⌋*(RS_SWCT_03260)*

For example, an ApplicationArrayDataType that has 100 elements would need to be initialized such that for each element a dedicated initial value is provided.

In the most prominent cases the majority of these elements are initialized with an identical value (e.g. 0) and only the first few elements differ in terms of initialization values.

Please note that ApplicationRuleBasedValueSpecification applies for arrays typed by a primitive data type. Rule-based value specification of arrays of a composite data type is done by means of the CompositeRuleBasedValueSpecification.

Class	***AbstractRuleBasedValueSpecification*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This represents an abstract base class for all rule-based value specifications.			
Base	*ARObject*, *ValueSpecification*			
Subclasses	ApplicationRuleBasedValueSpecification, CompositeRuleBasedValueSpecification, NumericalRuleBasedValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 5.129: AbstractRuleBasedValueSpecification

[constr_1779] Scope of the definition of an AbstractRuleBasedValueSpecification ⌈An AbstractRuleBasedValueSpecification shall only be defined in the context of an ArrayValueSpecification or a ConstantSpecification. If the AbstractRuleBasedValueSpecification is defined in the context of a ConstantSpecification then a reference to this ConstantSpecification shall only be created in the context of an ArrayValueSpecification.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Class	**ApplicationRuleBasedValueSpecification**
Package	M2::AUTOSARTemplates::CommonStructure::Constants
Note	This meta-class represents rule based values for DataPrototypes typed by ApplicationDataTypes (ApplicationArrayDataType or a compound ApplicationPrimitiveDataType which also boils down to an array-nature).
Base	*ARObject*, *AbstractRuleBasedValueSpecification*, *CompositeRuleBasedValueArgument*, *ValueSpecification*

▽

△

Class	ApplicationRuleBasedValueSpecification			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, CompositeRuleBasedValueSpecification.compoundPrimitiveArgument, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
category	Identifier	0..1	attr	This represents the category of the RuleBasedValueSpecification **Tags:**xml.sequenceOffset=-20
swAxisCont (ordered)	RuleBasedAxisCont	*	aggr	This represents the axis values of a Compound Primitive Data Type (curve or map). The first swAxisCont describes the x-axis, the second swAxisCont describes the y-axis, the third swAxisCont describes the z-axis. In addition to this, the axis can be denoted in swAxisIndex.
swValueCont	RuleBasedValueCont	0..1	aggr	This represents the values of an array or Compound Primitive Data Type.

Table 5.130: ApplicationRuleBasedValueSpecification

[constr_1922] Existence of ApplicationRuleBasedValueSpecification.category ⌈For each ApplicationRuleBasedValueSpecification, attribute category shall exist **at the time when the RTE is generated**.⌋*()*

Please note that attribute ApplicationRuleBasedValueSpecification.category is mentioned in [constr_2058], which would fail if the attribute does not exist.

[constr_10041] Value of ApplicationRuleBasedValueSpecification.swAxisCont.category ⌈The value of ApplicationValueSpecification.swAxisCont.category shall not be set to fixAXIS⌋*()*

Rationale for the existence of [constr_10041]: the value fixAXIS of attribute category indicates that the respective axis that is calculated out of the value of calibration parameters. Obviously, it does not make sense to initialize an axis of this kind and therefore it is excluded from the usage inside an ApplicationRuleBasedValueSpecification.

Class	RuleBasedAxisCont
Package	M2::AUTOSARTemplates::CommonStructure::Constants
Note	This represents the values for the axis of a compound primitive (curve, map). For standard and fix axes, SwAxisCont contains the values of the axis directly. The axis values of SwAxisCont with the category COM_AXIS, RES_AXIS are for display only. For editing and processing, only the values in the related GroupAxis are binding.
Base	*ARObject*

▽

△

Class	RuleBasedAxisCont			
Aggregated by	ApplicationRuleBasedValueSpecification.swAxisCont			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	CalprmAxisCategoryEnum	0..1	attr	This category specifies the particular axis types: • STD_AXIS • COM_AXIS • RES_AXIS (swArraysize necessary) **Tags:**xml.sequenceOffset=20
ruleBasedValues	RuleBasedValueSpecification	0..1	aggr	This represents the rule based value specification for the axis of a compound primitive (curve, map). **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=80 xml.typeWrapperElement=false
swArraysize	ValueList	0..1	aggr	For multidimensional compound primitives (curve, map ...) it is necessary to know the dimensions.They are specified using swArraySize. **Tags:**xml.sequenceOffset=40
swAxisIndex	AxisIndexType	0..1	attr	This property allows to explicitly assign the axis contents to a particular axis. It is specified by numbers where 1 corresponds to the x-axis. It is also possible to derive the axis association from the sequence of the parent. **Tags:**xml.sequenceOffset=50
unit	Unit	0..1	ref	This represents the physical unit of the provided values. **Tags:**xml.sequenceOffset=30

Table 5.131: RuleBasedAxisCont

[constr_1923] Existence of `RuleBasedAxisCont.ruleBasedValues` ⌈For each `RuleBasedAxisCont`, attribute `ruleBasedValues` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

Class	RuleBasedValueCont			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This represents the values of a compound primitive (CURVE, MAP, CUBOID, CUBE_4, CUBE_5, VAL_BLK) or an array.			
Base	*ARObject*			
Aggregated by	ApplicationRuleBasedValueSpecification.swValueCont			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
ruleBasedValues	RuleBasedValueSpecification	0..1	aggr	This represents the rule based value specification for the array or compound primitive (CURVE, MAP, CUBOID, CUBE_4, CUBE_5, VAL_BLK). **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=80 xml.typeWrapperElement=false

▽

△

Class	RuleBasedValueCont			
swArraysize	ValueList	0..1	aggr	This attribute defines the size of each dimension for compound primitives CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, VAL_BLK. For each dimension one value has to be defined, e.g. one in case of COM_AXIS and two or more in case of MAP. **Tags:**xml.sequenceOffset=40
unit	Unit	0..1	ref	This represents the physical unit of the provided values. **Tags:**xml.sequenceOffset=30

Table 5.132: RuleBasedValueCont

[constr_1924] Existence of RuleBasedValueCont.ruleBasedValues ⌈For each RuleBasedValueCont, attribute ruleBasedValues shall exist **at the time when the contract phase generation is executed**.⌋*()*

In case the ApplicationRuleBasedValueSpecification is applied to Compound Primitive Data Types basically the same rules apply for ApplicationRuleBasedValueSpecification as defined for ApplicationValueSpecification.

[constr_2057] Mandatory information of a RuleBasedAxisCont ⌈If the attribute swAxisCont is defined for an ApplicationRuleBasedValueSpecification the RuleBasedAxisCont shall define one swAxisIndex value and one swArraysize value per dimension, even in the case when the owning ApplicationRuleBasedValueSpecification defines only the content of a single dimensional object like a CURVE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_2058] Mandatory information of a RuleBasedValueCont ⌈If the attribute swValueCont is defined for an ApplicationRuleBasedValueSpecification the RuleBasedValueCont shall always define the attribute swArraysize if the ApplicationRuleBasedValueSpecification is of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, or VAL_BLK.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note that the definition of attribute swArraysize is not required for an ApplicationRuleBasedValueSpecification of category ARRAY because the applicable size can typically be derived from the context.

Please note further that for multidimensional Compound Primitive Data Types (e.g. MAP) it is necessary to know the dimensions in order to be able to process the SwValues. [constr_2057] and [constr_2058] shall support a consistent handling of single and multidimensional Compound Primitive Data Types.

If the ApplicationRuleBasedValueSpecification defines values for a Compound Primitive Data Type with more than one input axis the swArraysize gets mandatory to ensure the correct processing of the values calculated by rule.

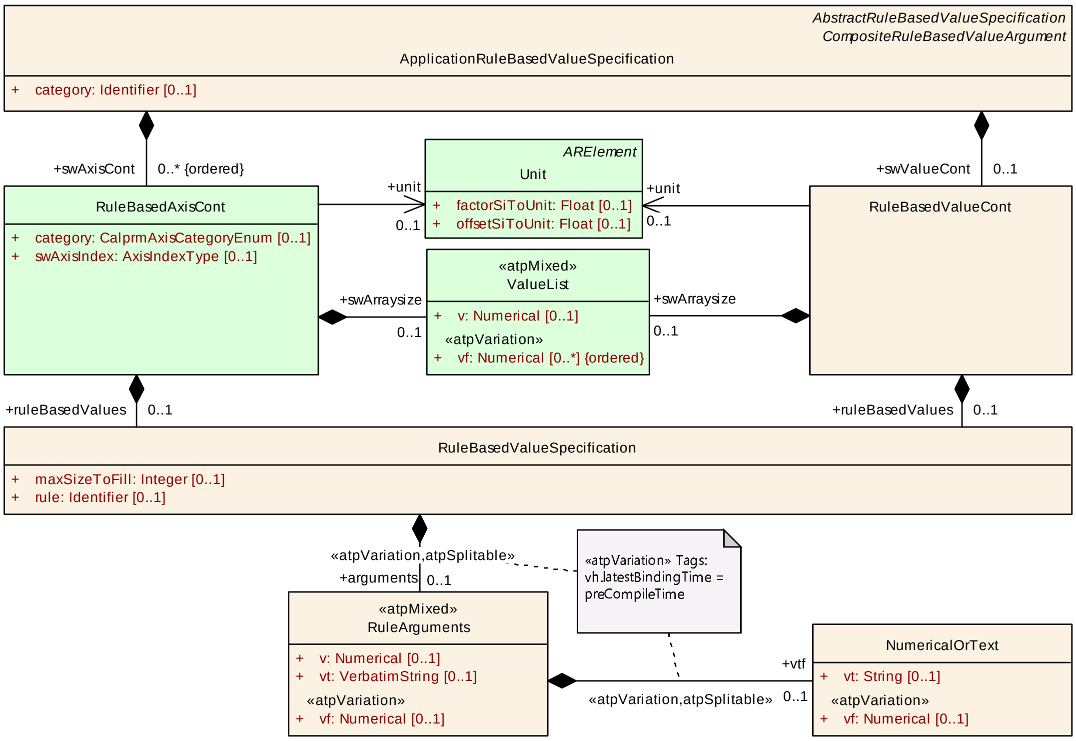


Figure 5.78: Definition of an ApplicationRuleBasedValueSpecification

[TPS_SWCT_02053] Values of RuleBasedAxisCont with the category COM_AXIS, RES_AXIS are for display only ⌈In case of ApplicationRuleBasedValueSpecifications of category MAP, CUBOID, CUBE_4, CUBE_5 or CURVE it is possible that the RuleBasedAxisCont of axes can be omitted if the axis is of category COM_AXIS or RES_AXIS.

If RuleBasedAxisCont values exists in such cases for the axes these are for display purpose only because the related DataPrototype of the MAP or CURVE does not hold the values of such axes. These are properties of the DataPrototype of the COM_AXIS or RES_AXIS.⌋*()*

Hence, values of the COM_AXIS itself are described by RuleBasedValueCont.

[TPS_SWCT_01528] Meaning of NumericalRuleBasedValueSpecification ⌈The purpose of the NumericalRuleBasedValueSpecification is to provide means for a compact provision of values for DataPrototypes that otherwise would require a high volume (in terms of serialized ARXML) of e.g. initialization data. NumericalRuleBasedValueSpecification may used for DataPrototypes typed by ImplementationDataTypes of category ARRAY or Compound Primitive Data Types mapped to ImplementationDataTypes of category ARRAY.⌋*(RS_SWCT_03260)*

Concerning initValues for Compound Primitive Data Types please note as well [TPS_SWCT_01185].

Class	NumericalRuleBasedValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to support a rule-based initialization approach for data types with an array-nature (ImplementationDataType of category ARRAY).			
Base	*ARObject*, *AbstractRuleBasedValueSpecification*, *ValueSpecification*			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
ruleBasedValues	RuleBasedValueSpecification	0..1	aggr	This represents the rule based value specification for the array. **Tags:** xml.roleElement=true xml.roleWrapperElement=false xml.typeWrapperElement=false

Table 5.133: NumericalRuleBasedValueSpecification

[constr_1925] Existence of NumericalRuleBasedValueSpecification.ruleBasedValues ⌈For each NumericalRuleBasedValueSpecification, attribute ruleBasedValues shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01495] Standardized value of RuleBasedValueSpecification.rule ⌈AUTOSAR reserves a dedicated value of RuleBasedValueSpecification.rule and CompositeRuleBasedValueSpecification.rule in a standardized semantics:

- FILL_UNTIL_END
- FILL_UNTIL_MAX_SIZE

The meaning of this value of rule is explained in [TPS_SWCT_01494] and [TPS_SWCT_01609].⌋*(RS_SWCT_03260, RS_SWCT_03181)*

[TPS_SWCT_01485] The order of RuleArguments arguments shall be respected ⌈The order of arguments in RuleArguments corresponds to the order of elements in the array, i.e. the first argument corresponds to the first element of the array, the second argument corresponds to the second element of the array, and so on.⌋*(RS_SWCT_03260)*

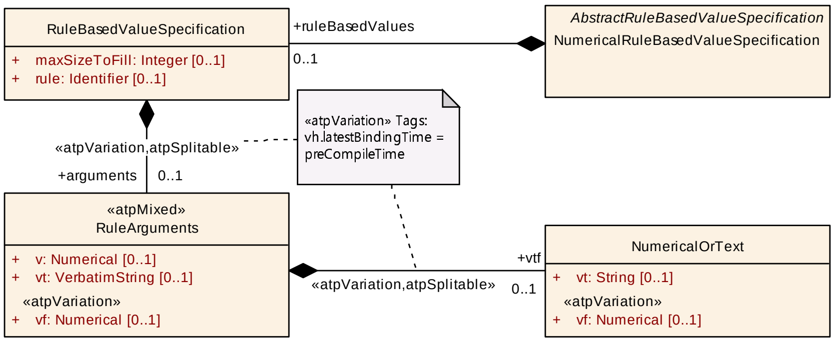


Figure 5.79: Definition of a `NumericalRuleBasedValueSpecification`

Please note that a single argument can be defined by the attributes

- `RuleArguments.v`
- `RuleArguments.vf`
- `RuleArguments.vt`
- `RuleArguments.vtf.vf`
- `RuleArguments.vtf.vt`

[TPS_SWCT_01493] The number of `RuleBasedValueSpecification.arguments` shall not exceed the array size ⌈If the number of `RuleBasedValueSpecification.arguments` exceeds the number of elements of an array that it is applied to then the `RuleBasedValueSpecification.arguments` that go beyond the last element of the array shall be ignored.⌋*(RS_SWCT_03260)*

[TPS_SWCT_01494] A `RuleBasedValueSpecification` of `rule` `FILL_UNTIL_END` shall fill the value of the last `RuleBasedValueSpecification.arguments` until the last element of the array ⌈The following rule applies to `RuleBasedValueSpecification`s of `rule FILL_UNTIL_END`:

If the number of `RuleBasedValueSpecification.arguments` is smaller than the number of elements of the array it is applied to then the value of the last `RuleBasedValueSpecification.arguments` shall be applied to any following element of the array until the last element of the array.⌋*(RS_SWCT_03260)*

[TPS_SWCT_01609] A `RuleBasedValueSpecification` of `rule` `FILL_UNTIL_MAX_SIZE` shall fill the value of the last `RuleBasedValueSpecification.arguments` until the number of elements specified in `maxSizeToFill` ⌈The following rule applies to `RuleBasedValueSpecification`s of `rule FILL_UNTIL_MAX_SIZE`:

If the number of `RuleBasedValueSpecification.arguments` is smaller than the number of elements of the array it is applied to and smaller than `maxSizeToFill`, then the value of the last `RuleBasedValueSpecification.arguments` shall be applied to so many of the following elements that the first `maxSizeToFill` elements of the array are filled.⌋*(RS_SWCT_03260)*

Class	RuleBasedValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to support a rule-based initialization approach for data types with an array-nature (ApplicationArrayDataType and ImplementationDataType of category ARRAY) or a compound Application PrimitiveDataType (which also boils down to an array-nature).			
Base	*ARObject*			
Aggregated by	NumericalRuleBasedValueSpecification.ruleBasedValues, RuleBasedAxisCont.ruleBasedValues, Rule BasedValueCont.ruleBasedValues			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arguments	RuleArguments	0..1	aggr	This represents the arguments for the RuleBasedValue Specification. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arguments, arguments.variationPoint.short Label vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
maxSizeToFill	Integer	0..1	attr	If a rule is chosen which does not fill until the end, this determines until which size the rule shall fill the values. **Tags:**xml.sequenceOffset=40
rule	Identifier	0..1	attr	This denotes the name of the rule of the RuleBasedValue Specification. The rule determines the calculation specification according which the arguments are used to calculated the values. **Tags:**xml.sequenceOffset=20

Table 5.134: RuleBasedValueSpecification

[constr_1926] Existence of RuleBasedValueSpecification.rule ⌈For each RuleBasedValueSpecification, attribute rule shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1927] Existence of RuleBasedValueSpecification.arguments ⌈For each RuleBasedValueSpecification, the aggregation of RuleArguments in the role arguments shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	<<atpMixed>> **RuleArguments**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This represents the arguments for a rule-based value specification.			
Base	*ARObject*			
Aggregated by	RuleBasedValueSpecification.arguments			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
v	Numerical	0..1	attr	This represents a numerical value for the RuleBased ValueSpecification.
vf	Numerical	0..1	attr	This represents a numerical value for the RuleBased ValueSpecification which may subject to variability. The latest binding time of the VariationPoint shall be pre CompileTime. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

▽

△

Class	<<atpMixed>> **RuleArguments**			
vt	VerbatimString	0..1	attr	This represents a textual value for the RuleBasedValue Specification.
vtf	NumericalOrText	0..1	aggr	This aggregation represents the ability to provide a value that is either numerical or text which existence is subject to variability. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=vtf, vtf.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 5.135: RuleArguments

As an example for the application of a RuleBasedValueSpecification, consider the following scenario:

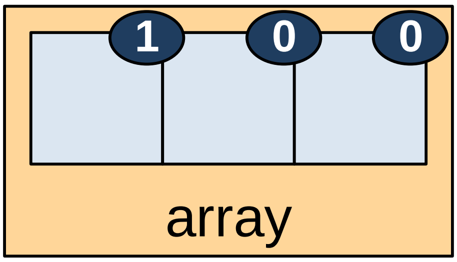


Figure 5.80: Value specification for a simple array

The sketched array depicted in Figure 5.80 corresponds to the modeling exemplified in Listing 5.24.

```
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
      <SW-VALUE-CONT>
        <RULE-BASED-VALUES>
          <RULE>FILL_UNTIL_END</RULE>
          <ARGUMENTSS>
            <RULE-ARGUMENTS>
              <V>1</V>
              <V>0</V>
            </RULE-ARGUMENTS>
          </ARGUMENTSS>
        </RULE-BASED-VALUES>
      </SW-VALUE-CONT>
    </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
```

Listing 5.24: Value specification for a simple array

Please note that it is not foreseen that an ArrayValueSpecification consist of a collection of ApplicationRuleBasedValueSpecifications or even a mixture of ApplicationRuleBasedValueSpecification with another kind of ValueSpecification.

[constr_10009] Aggregation of ApplicationRuleBasedValueSpecification
⌈Each ArrayValueSpecification shall only aggregate at most one ApplicationRuleBasedValueSpecification in the role element.

If one ApplicationRuleBasedValueSpecification is aggregated then it shall be the only aggregated element, i.e. no further ValueSpecification shall exist in the same aggregation where an ApplicationRuleBasedValueSpecification is aggregated.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note that it is not foreseen that an ArrayValueSpecification consist of a collection of NumericalRuleBasedValueSpecification or even a mixture of NumericalRuleBasedValueSpecification with another kind of ValueSpecification.

[constr_1754] Aggregation of NumericalRuleBasedValueSpecification
⌈Each ArrayValueSpecification shall only aggregate at most one NumericalRuleBasedValueSpecification in the role element.

If one NumericalRuleBasedValueSpecification is aggregated then it shall be the only aggregated element, i.e. no further ValueSpecification shall exist in the same aggregation where an NumericalRuleBasedValueSpecification is aggregated.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

5.6.9.2 Support for composite Data Types

[TPS_SWCT_01692] Meaning of CompositeRuleBasedValueSpecification
⌈The rule-based initialization of arrays of a composite data type is modeled by means of the CompositeRuleBasedValueSpecification.⌋*(RS_SWCT_03260)*

Class	**CompositeRuleBasedValueSpecification**
Package	M2::AUTOSARTemplates::CommonStructure::Constants
Note	This meta-class represents rule-based values for DataPrototypes typed by composite AutosarDataTypes.
Base	*ARObject*, *AbstractRuleBasedValueSpecification*, *ValueSpecification*

▽

△

Class	**CompositeRuleBasedValueSpecification**			
Aggregated by	ApplicationAssocMapElementValueSpecification.key, ApplicationAssocMapElementValueSpecification.value, ArrayValueSpecification.element, CalibrationParameterValue.applInitValue, CalibrationParameterValue.implInitValue, ConstantSpecification.valueSpec, CryptoServiceKey.developmentValue, DiagnosticEnvDataCondition.compareValue, DiagnosticEnvDataElementCondition.compareValue, FieldSenderComSpec.initValue, ISignal.initValue, ISignal.timeoutSubstitutionValue, NonqueuedReceiverComSpec.initValue, NonqueuedReceiverComSpec.timeoutSubstitutionValue, NonqueuedSenderComSpec.initValue, NvProvideComSpec.ramBlockInitValue, NvProvideComSpec.romBlockInitValue, NvRequireComSpec.initValue, ParameterDataPrototype.initValue, ParameterProvideComSpec.initValue, ParameterRequireComSpec.initValue, PersistencyDataRequiredComSpec.initValue, PersistencyKeyValuePair.initValue, PortDefinedArgumentValue.value, PortPrototypeBlueprintInitValue.value, RecordValueSpecification.field, *StateManagementCompareCondition*.compareValue, SwDataDefProps.invalidValue, VariableDataPrototype.initValue			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
argument (ordered)	CompositeValueSpecification	*	aggr	This represents the collection of aggregated ValueSpecifications. The last ValueSpecification in the collection shall be taken to execute the filling rule. **Tags:**xml.sequenceOffset=30
compoundPrimitiveArgument (ordered)	CompositeRuleBasedValueArgument	*	aggr	This represents the collection of aggregated ValueSpecifications for compound primitive data type. The last ValueSpecification in the collection shall be taken to execute the filling rule. **Tags:**xml.sequenceOffset=35
maxSizeToFill	PositiveInteger	0..1	attr	If a rule is chosen which does not fill until the end, this determines until which size the rule shall fill the values. **Tags:**xml.sequenceOffset=40
rule	Identifier	0..1	attr	This denotes the name of the rule of the RuleBasedValueSpecification. The rule determines the calculation specification according which the arguments are used to calculated the values. **Tags:**xml.sequenceOffset=20

Table 5.136: CompositeRuleBasedValueSpecification

[constr_1928] Existence of CompositeRuleBasedValueSpecification.rule ⌈For each CompositeRuleBasedValueSpecification, attribute rule shall exist **at the time when the contract phase generation is executed.**⌋*()*

[constr_1929] Existence of CompositeRuleBasedValueSpecification.argument ⌈For each CompositeRuleBasedValueSpecification, the aggregation of CompositeValueSpecification in the role argument shall exist **at the time when the contract phase generation is executed.**⌋*()*

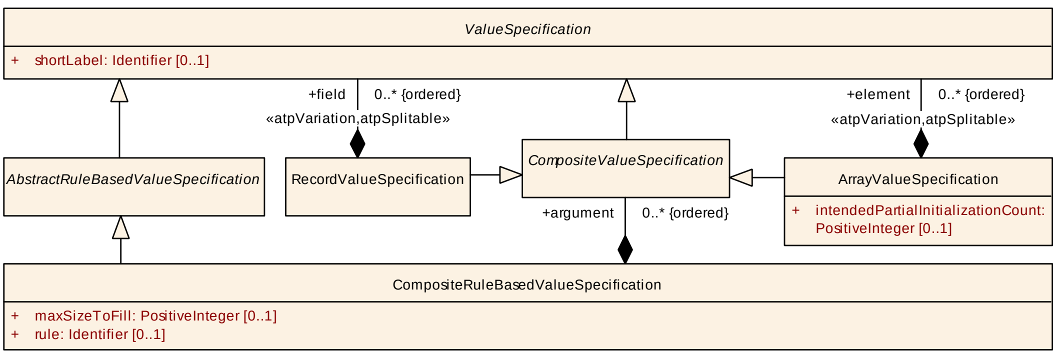


Figure 5.81: Rule-based value specification of arrays of a composite data type

As an example of how the rule-based initialization of composite data structures works, please consider the composite structure sketched in Figure 5.82. In simple terms, it describes an array consisting of elements that themselves are typed by a structure of two primitive elements.

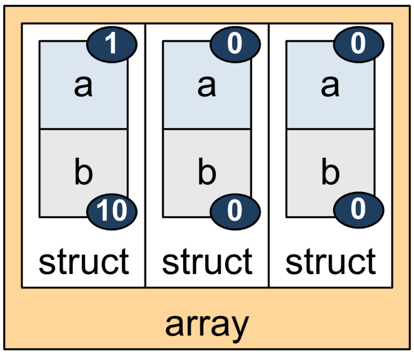


Figure 5.82: Example for the explanation of rule-based composite value-specification

In this example, the element “a” of the first structure shall be initialized with the value 1, the corresponding “b” element shall be assigned a 10. All other values in all following elements shall be set to 0. This is also indicated by the numbers in ellipses in Figure 5.82.

The implementation of the example in ARXML is illustrated in Listing 5.25. As already explained before, the last (in the order of appearance) ValueSpecification in the context of an AbstractRuleBasedValueSpecification is taken to execute the rule (as described above).

```
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
      <RULE>FILL_UNTIL_END</RULE>
      <ARGUMENTS>
        <RECORD-VALUE-SPECIFICATION>
          <FIELDS>
            <APPLICATION-VALUE-SPECIFICATION>
              <SW-VALUE-CONT>
                <SW-VALUES-PHYS>
                  <V>1</V>
                </SW-VALUES-PHYS>
              </SW-VALUE-CONT>
            </APPLICATION-VALUE-SPECIFICATION>
            <APPLICATION-VALUE-SPECIFICATION>
              <SW-VALUE-CONT>
                <SW-VALUES-PHYS>
                  <V>10</V>
                </SW-VALUES-PHYS>
              </SW-VALUE-CONT>
            </APPLICATION-VALUE-SPECIFICATION>
          </FIELDS>
        </RECORD-VALUE-SPECIFICATION>
        <RECORD-VALUE-SPECIFICATION>
          <FIELDS>
```

```
            <APPLICATION-VALUE-SPECIFICATION>
              <SW-VALUE-CONT>
                <SW-VALUES-PHYS>
                  <V>0</V>
                </SW-VALUES-PHYS>
              </SW-VALUE-CONT>
            </APPLICATION-VALUE-SPECIFICATION>
            <APPLICATION-VALUE-SPECIFICATION>
              <SW-VALUE-CONT>
                <SW-VALUES-PHYS>
                  <V>0</V>
                </SW-VALUES-PHYS>
              </SW-VALUE-CONT>
            </APPLICATION-VALUE-SPECIFICATION>
          </FIELDS>
        </RECORD-VALUE-SPECIFICATION>
      </ARGUMENTS>
    </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
```

Listing 5.25: Example for composite rule-based value specification

A more complicated example is sketched in Figure 5.83. Here, a deeply nested composite data structure is described: an array of structures that in turn contain an array.

To keep the ARXML listing as simple as possible, the example assumes that **all** (as opposed to the initialization of the first, and then of all other elements) "struct" elements shall be initialized with the same value.

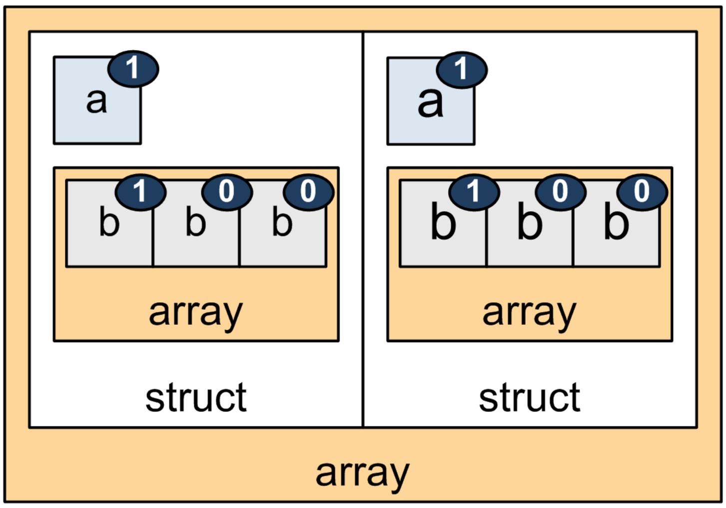


Figure 5.83: Value specification for a deeply nested array

The deeply nested "array" is initialized by means of an ApplicationRuleBased-ValueSpecification.

```
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
```

```
      <RULE>FILL_UNTIL_END</RULE>
      <ARGUMENTS>
        <RECORD-VALUE-SPECIFICATION>
          <FIELDS>
            <APPLICATION-VALUE-SPECIFICATION>
              <SW-VALUE-CONT>
                <SW-VALUES-PHYS>
                  <V>1</V>
                </SW-VALUES-PHYS>
              </SW-VALUE-CONT>
            </APPLICATION-VALUE-SPECIFICATION>
            <ARRAY-VALUE-SPECIFICATION>
              <ELEMENTS>
                <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
                  <SW-VALUE-CONT>
                    <RULE-BASED-VALUES>
                      <RULE>FILL_UNTIL_END</RULE>
                      <ARGUMENTSS>
                        <RULE-ARGUMENTS>
                          <V>1</V>
                          <V>0</V>
                        </RULE-ARGUMENTS>
                      </ARGUMENTSS>
                    </RULE-BASED-VALUES>
                  </SW-VALUE-CONT>
                </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
              </ELEMENTS>
            </ARRAY-VALUE-SPECIFICATION>
          </FIELDS>
        </RECORD-VALUE-SPECIFICATION>
      </ARGUMENTS>
    </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
```

Listing 5.26: Value specification for a deeply nested array

Please note that it is not foreseen that an ArrayValueSpecification consist of a collection of CompositeRuleBasedValueSpecification or even a mixture of CompositeRuleBasedValueSpecification with another kind of ValueSpecification.

[constr_1755] Aggregation of CompositeRuleBasedValueSpecification ⌈Each ArrayValueSpecification shall only aggregate at most one CompositeRuleBasedValueSpecification in the role element.

If one CompositeRuleBasedValueSpecification is aggregated then it shall be the only aggregated element, i.e. no further ValueSpecification shall exist in the same aggregation where an CompositeRuleBasedValueSpecification is aggregated.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

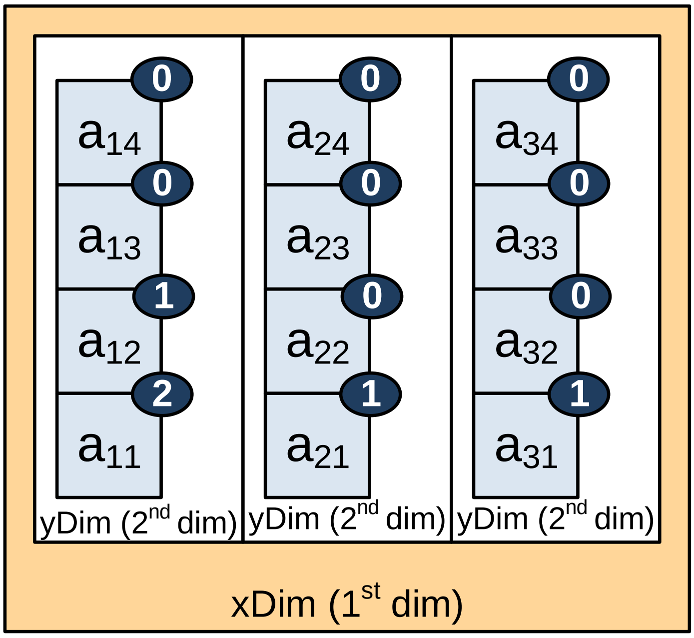


Figure 5.84: Rule-based Value specification for a 2-dimensional array

As another variation of the theme, the rule-based initialization of multi-dimensional arrays shall be discussed.

Consequently, the following example extends the one-dimensional rule-based Value-Specification (see Figure 5.80) to a second dimension. The general layout is sketched in the following figure:

For the sake of clarity, the picture has been drawn to align the first dimension (the *x-axis*) of the two-dimensional array with the horizontal direction an the second dimension (the *y-axis*) with the vertical direction.

The direction index values of each array element are visible as subscript on the bottom right of the element, i.e. a_{12} indicates that the element is part of the first element on the *x-axis* and represents the second element of the *y-axis*. The initial value of element a_{12} shall be 1.

As indicated by the sketch in Figure 5.84, the second element (i.e. the "vertical" array, i.e. everything from a_{21} to a_{24}) and all following (i.e. everything from a_{31} to a_{34}) shall have the identical initial values. The first element deviates from the second in terms of initial values.

The creation of the `ArrayValueSpecification` for this example is based on the definition of a `CompositeRuleBasedValueSpecification` with two `argument`s:

- an `ArrayValueSpecification` that carries an `ApplicationRuleBased-ValueSpecification` for the first element (that itself is an array) on the x-dimension and

- an ArrayValueSpecification that carries an ApplicationRuleBasedValueSpecification for each of the remaining elements (that itself are arrays) on the x-dimension.

```
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
      <RULE>FILL_UNTIL_END</RULE>
      <ARGUMENTS>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
              <CATEGORY>ARRAY</CATEGORY>
              <SW-VALUE-CONT>
                <RULE-BASED-VALUES>
                  <RULE>FILL_UNTIL_END</RULE>
                  <ARGUMENTSS>
                    <RULE-ARGUMENTS>
                      <V>2</V>
                      <V>1</V>
                      <V>0</V>
                    </RULE-ARGUMENTS>
                  </ARGUMENTSS>
                </RULE-BASED-VALUES>
              </SW-VALUE-CONT>
            </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
          </ELEMENTS>
        </ARRAY-VALUE-SPECIFICATION>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
              <CATEGORY>ARRAY</CATEGORY>
              <SW-VALUE-CONT>
                <RULE-BASED-VALUES>
                  <RULE>FILL_UNITL_END</RULE>
                  <ARGUMENTSS>
                    <RULE-ARGUMENTS>
                      <V>1</V>
                      <V>0</V>
                    </RULE-ARGUMENTS>
                  </ARGUMENTSS>
                </RULE-BASED-VALUES>
              </SW-VALUE-CONT>
            </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
          </ELEMENTS>
        </ARRAY-VALUE-SPECIFICATION>
      </ARGUMENTS>
    </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
```

Listing 5.27: Value specification for a 2-dimensional array

The next example adds one dimension to the array structure, i.e. it describes a three-dimensional array, as sketched in Figure 5.85.

In this case the *x-axis* has again been aligned in the horizontal direction, why the *y-axis* is drawn vertically. The *z-axis*, finally, goes horizontal again.

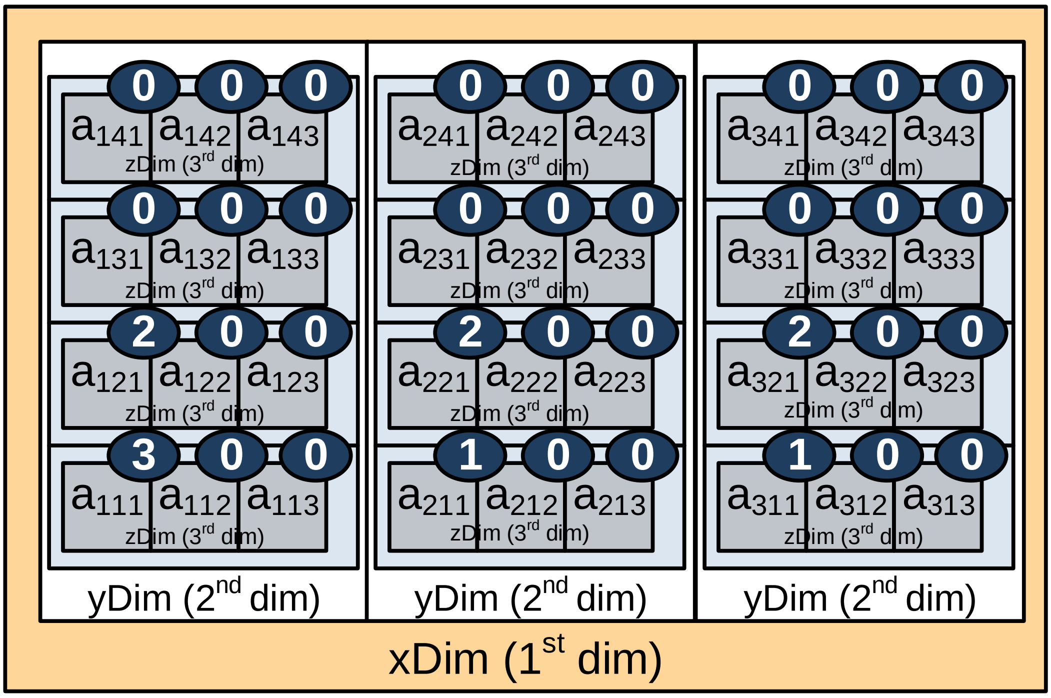


Figure 5.85: Rule-based Value specification for a 3-dimensional array

The principal structure of this example is similar to the previous example.

The first array element in the x-dimension (which includes everything from a_{111} to a_{143}) defines a different initial value than the other elements.

Please note that the initial values of the second and third element in x-direction are identical.

```
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
      <RULE>FILL_UNTIL_END</RULE>
      <ARGUMENTS>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
              <RULE>FILL_UNTIL_END</RULE>
              <ARGUMENTS>
                <ARRAY-VALUE-SPECIFICATION>
                  <ELEMENTS>
                    <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
                      <CATEGORY>ARRAY</CATEGORY>
                      <SW-VALUE-CONT>
                        <RULE-BASED-VALUES>
                          <RULE>FILL_UNTIL_END</RULE>
                          <ARGUMENTSS>
                            <RULE-ARGUMENTS>
```

```
                      <V>3</V>
                      <V>0</V>
                    </RULE-ARGUMENTS>
                  </ARGUMENTSS>
                </RULE-BASED-VALUES>
              </SW-VALUE-CONT>
            </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
          </ELEMENTS>
        </ARRAY-VALUE-SPECIFICATION>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
              <CATEGORY>ARRAY</CATEGORY>
              <SW-VALUE-CONT>
                <RULE-BASED-VALUES>
                  <RULE>FILL_UNTIL_END</RULE>
                  <ARGUMENTSS>
                    <RULE-ARGUMENTS>
                      <V>2</V>
                      <V>0</V>
                    </RULE-ARGUMENTS>
                  </ARGUMENTSS>
                </RULE-BASED-VALUES>
              </SW-VALUE-CONT>
            </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
          </ELEMENTS>
        </ARRAY-VALUE-SPECIFICATION>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
              <CATEGORY>ARRAY</CATEGORY>
              <SW-VALUE-CONT>
                <RULE-BASED-VALUES>
                  <RULE>FILL_UNTIL_END</RULE>
                  <ARGUMENTSS>
                    <RULE-ARGUMENTS>
                      <V>0</V>
                    </RULE-ARGUMENTS>
                  </ARGUMENTSS>
                </RULE-BASED-VALUES>
              </SW-VALUE-CONT>
            </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
          </ELEMENTS>
        </ARRAY-VALUE-SPECIFICATION>
      </ARGUMENTS>
    </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
<ARRAY-VALUE-SPECIFICATION>
  <ELEMENTS>
    <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
      <RULE>FILL_UNTIL_END</RULE>
      <ARGUMENTS>
        <ARRAY-VALUE-SPECIFICATION>
          <ELEMENTS>
            <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
```

```
                <CATEGORY>ARRAY</CATEGORY>
                <SW-VALUE-CONT>
                  <RULE-BASED-VALUES>
                    <RULE>FILL_UNTIL_END</RULE>
                    <ARGUMENTSS>
                      <RULE-ARGUMENTS>
                        <V>1</V>
                        <V>0</V>
                      </RULE-ARGUMENTS>
                    </ARGUMENTSS>
                  </RULE-BASED-VALUES>
                </SW-VALUE-CONT>
              </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
            </ELEMENTS>
          </ARRAY-VALUE-SPECIFICATION>
          <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
              <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
                <CATEGORY>ARRAY</CATEGORY>
                <SW-VALUE-CONT>
                  <RULE-BASED-VALUES>
                    <RULE>FILL_UNTIL_END</RULE>
                    <ARGUMENTSS>
                      <RULE-ARGUMENTS>
                        <V>2</V>
                        <V>0</V>
                      </RULE-ARGUMENTS>
                    </ARGUMENTSS>
                  </RULE-BASED-VALUES>
                </SW-VALUE-CONT>
              </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
            </ELEMENTS>
          </ARRAY-VALUE-SPECIFICATION>
          <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
              <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
                <CATEGORY>ARRAY</CATEGORY>
                <SW-VALUE-CONT>
                  <RULE-BASED-VALUES>
                    <RULE>FILL_UNTIL_END</RULE>
                    <ARGUMENTSS>
                      <RULE-ARGUMENTS>
                        <V>0</V>
                      </RULE-ARGUMENTS>
                    </ARGUMENTSS>
                  </RULE-BASED-VALUES>
                </SW-VALUE-CONT>
              </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
            </ELEMENTS>
          </ARRAY-VALUE-SPECIFICATION>
        </ARGUMENTS>
      </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
    </ELEMENTS>
  </ARRAY-VALUE-SPECIFICATION>
</ARGUMENTS>
</COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
```

```
  </ELEMENTS>
</ARRAY-VALUE-SPECIFICATION>
```

Listing 5.28: Value specification for a 3-dimensional array

5.6.9.3 Support for compound primitive Data Types

Please note that the CompositeRuleBasedValueSpecification also supports the rule-based initialization of arrays[21] that are typed by a Compound Primitive Data Type, e.g. of category CURVE.

[TPS_SWCT_01836] Attributes of CompositeRuleBasedValueSpecification ⌈Meta-class CompositeRuleBasedValueSpecification can be used to fulfill two distinct use cases:

- Definition of a rule-based value specification for an array of composite data type based on the aggregation of CompositeValueSpecification in the role argument.
- Definition of a rule-based value specification for an array of Compound Primitive Data Type based on the aggregation of CompositeRuleBasedValueArgument in the role compoundPrimitiveArgument.

⌋*()*

[constr_10075] Existence of CompositeRuleBasedValueSpecification.argument vs. compoundPrimitiveArgument ⌈For every CompositeRuleBasedValueSpecification, at most one of the aggregations

- argument
- compoundPrimitiveArgument

shall exist **at the time when the contract phase generation is executed**.⌋*()*

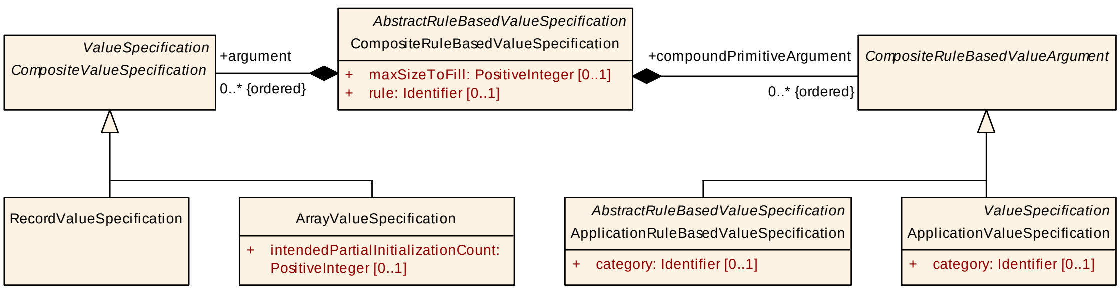


Figure 5.86: Rule-based value specification for compound primitive data objects

[21] Of course, this capability is restricted to ApplicationArrayDataType. On the level of ImplementationDataType, ValueSpecifications that reflect the structure of the respective ImplementationDataType are used.

Class	***CompositeRuleBasedValueArgument*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class has the ability to serve as the abstract base class for ValueSpecifications that can be used for compound primitive data types.			
Base	*ARObject*			
Subclasses	ApplicationRuleBasedValueSpecification, ApplicationValueSpecification			
Aggregated by	CompositeRuleBasedValueSpecification.compoundPrimitiveArgument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 5.137: CompositeRuleBasedValueArgument

Please find a simplified example for the rule-based initialization of an array of an `ApplicationPrimitiveDataType` of `category` `CURVE` with an integrated axis in section 5.6.10.5.

5.6.10 Examples

5.6.10.1 Example for Constant Specification for CURVE

The following example illustrates how a `ConstantSpecification` is specified for a `CURVE`. Please note, that in this example the `vf` attribute is used for the `swArraysize` as well as for the `swValuesPhys`.

The basic intention of `vf` is the usage for variant rich models but it is valid as well if `vf` contains invariant values.

```
<CONSTANT-SPECIFICATION>
  <SHORT-NAME>PhysInitValuesOfCurve</SHORT-NAME>
  <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a CURVE where
     the axis is a STD_AXIS</L-2>
  </DESC>
  <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
    <CATEGORY>CURVE</CATEGORY>
    <SW-AXIS-CONTS>
    <SW-AXIS-CONT>
      <CATEGORY>STD_AXIS</CATEGORY>
      <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
      <SW-ARRAYSIZE>
      <VF>4</VF>
      </SW-ARRAYSIZE>
      <SW-VALUES-PHYS>
      <VF>0</VF>
      <VF>1</VF>
      <VF>2</VF>
      <VF>3</VF>
      </SW-VALUES-PHYS>
    </SW-AXIS-CONT>
    </SW-AXIS-CONTS>
    <SW-VALUE-CONT>
```

```
    <UNIT-REF DEST="UNIT">/Units/NwtMtr</UNIT-REF>
    <SW-ARRAYSIZE>
      <VF>4</VF>
    </SW-ARRAYSIZE>
    <SW-VALUES-PHYS>
      <VF>00.000</VF>
      <VF>10.000</VF>
      <VF>20.000</VF>
      <VF>30.000</VF>
    </SW-VALUES-PHYS>
    </SW-VALUE-CONT>
  </APPLICATION-VALUE-SPECIFICATION>
  </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

Listing 5.29: Example for Constant Specification for CURVE

5.6.10.2 Example for Constant Specification for MAP

The following example illustrates how an ConstantSpecification is specified for a MAP. In this case one axis of the MAP is a STD_AXIS and the second one is a COM_AXIS.

Please note that in this example the v attribute is used for the swArraysize as well as for the swValuesPhys.

This is possible because the example contains only invariant values.

```
<CONSTANT-SPECIFICATION>
  <SHORT-NAME>PhysInitValuesOfMap</SHORT-NAME>
  <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a MAP where
      the first axis is a STD_AXIS and the second axis is a COM_AXIS</L-2>
  </DESC>
  <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
    <CATEGORY>MAP</CATEGORY>
    <SW-AXIS-CONTS>
    <SW-AXIS-CONT>
      <CATEGORY>STD_AXIS</CATEGORY>
      <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
      <SW-ARRAYSIZE>
      <V>4</V>
      </SW-ARRAYSIZE>
      <SW-VALUES-PHYS>
      <V>0</V>
      <V>1</V>
      <V>2</V>
      <V>3</V>
      </SW-VALUES-PHYS>
    </SW-AXIS-CONT>
    </SW-AXIS-CONTS>
    <SW-VALUE-CONT>
    <UNIT-REF DEST="UNIT">/Units/NwtMtr</UNIT-REF>
```

```
      <SW-ARRAYSIZE>
        <V>4</V>
        <V>2</V>
      </SW-ARRAYSIZE>
      <SW-VALUES-PHYS>
        <VG>
        <LABEL>
          <L-4 L="EN">Values for axis index 2 equals 0</L-4>
        </LABEL>
        <V>00</V>
        <V>10</V>
        <V>20</V>
        <V>30</V>
        </VG>
        <VG>
        <LABEL>
          <L-4 L="EN">Values for axis index 2 equals 1</L-4>
        </LABEL>
        <V>01</V>
        <V>11</V>
        <V>21</V>
        <V>31</V>
        </VG>
      </SW-VALUES-PHYS>
      </SW-VALUE-CONT>
    </APPLICATION-VALUE-SPECIFICATION>
    </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

Listing 5.30: Example for Constant Specification for MAP

5.6.10.3 Example for Constant Specification for MAP with two STD_AXIS

The example contained in this sub-chapter illustrates the creation of the ConstantSpecification for a MAP that (in contrast to the previous example sketched in Listing 5.30) consists of two STD_AXIS.

Like in the previous example, the v attribute is used for the swArraysize as well as for the swValuesPhys.

```
<CONSTANT-SPECIFICATION>
    <SHORT-NAME>MapExample</SHORT-NAME>
    <VALUE-SPEC>
    <APPLICATION-VALUE-SPECIFICATION>
      <CATEGORY>MAP</CATEGORY>
      <SW-AXIS-CONTS>
      <SW-AXIS-CONT>
        <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
        <SW-ARRAYSIZE>
        <V>4</V>
        </SW-ARRAYSIZE>
        <SW-VALUES-PHYS>
        <V>1</V>
        <V>2</V>
```

```
        <V>3</V>
        <V>4</V>
        </SW-VALUES-PHYS>
      </SW-AXIS-CONT>
      <SW-AXIS-CONT>
        <SW-AXIS-INDEX>2</SW-AXIS-INDEX>
        <SW-ARRAYSIZE>
        <V>2</V>
        </SW-ARRAYSIZE>
        <SW-VALUES-PHYS>
        <V>10</V>
        <V>11</V>
        </SW-VALUES-PHYS>
      </SW-AXIS-CONT>
      </SW-AXIS-CONTS>
      <SW-VALUE-CONT>
      <SW-ARRAYSIZE>
        <V>4</V>
        <V>2</V>
      </SW-ARRAYSIZE>
      <SW-VALUES-PHYS>
        <VG>
        <LABEL>
          <L-4 L="EN">Values for 10</L-4>
        </LABEL>
        <V>110</V>
        <V>210</V>
        <V>310</V>
        <V>410</V>
        </VG>
        <VG>
        <LABEL>
          <L-4 L="EN">Values for 11</L-4>
        </LABEL>
        <V>111</V>
        <V>211</V>
        <V>311</V>
        <V>411</V>
        </VG>
      </SW-VALUES-PHYS>
      </SW-VALUE-CONT>
    </APPLICATION-VALUE-SPECIFICATION>
    </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

Listing 5.31: Example for Constant Specification for STD_AXIS

5.6.10.4 Example for Constant Specification for COM_AXIS

The following example illustrates how an ConstantSpecification is specified for a COM_AXIS.

```
<CONSTANT-SPECIFICATION>
  <SHORT-NAME>PhysInitValuesOfComAxis</SHORT-NAME>
```

```
  <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a COM_AXIS</L
      -2>
  </DESC>
  <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
    <CATEGORY>COM_AXIS</CATEGORY>
    <SW-VALUE-CONT>
    <UNIT-REF DEST="UNIT">/Units/Rpm</UNIT-REF>
    <SW-ARRAYSIZE>
      <V>6</V>
    </SW-ARRAYSIZE>
    <SW-VALUES-PHYS>
      <V>0</V>
      <V>500</V>
      <V>1000</V>
      <V>1500</V>
      <V>3000</V>
      <V>5000</V>
    </SW-VALUES-PHYS>
    </SW-VALUE-CONT>
  </APPLICATION-VALUE-SPECIFICATION>
  </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

Listing 5.32: Example for Constant Specification for COM_AXIS

5.6.10.5 Example for Constant Specification for an Array of compound primitive Objects

The example starts with the definition of the data type for the input value for the curve see Listing 5.33. This data type is used for the definition of the actual curve data type in Listing 5.35.

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>axisInputType</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.33: Definition of curve input value data type

The next step is the definition of the data type for the result of the interpolation. This part is sketched in Listing 5.34. This data type is used for the definition of the actual curve data type in Listing 5.35.

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>curveType</SHORT-NAME>
```

```
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.34: Definition of data type for the result of the interpolation

Finally, the data type for the actual curve is defined, see Listing 5.35.

```
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>MyTable</SHORT-NAME>
  <CATEGORY>CURVE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        <SW-CALPRM-AXIS-SET>
          <SW-CALPRM-AXIS>
            <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
            <CATEGORY>STD_AXIS</CATEGORY>
            <SW-AXIS-INDIVIDUAL>
              <INPUT-VARIABLE-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-
                  TYPE">/ApplicationDataTypes/axisInputType</INPUT-VARIABLE-
                  TYPE-REF>
              <SW-MAX-AXIS-POINTS>10</SW-MAX-AXIS-POINTS>
              <SW-MIN-AXIS-POINTS>0</SW-MIN-AXIS-POINTS>
            </SW-AXIS-INDIVIDUAL>
            <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          </SW-CALPRM-AXIS>
        </SW-CALPRM-AXIS-SET>
        <VALUE-AXIS-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/
            ApplicationDataTypes/curveType</VALUE-AXIS-DATA-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</APPLICATION-PRIMITIVE-DATA-TYPE>
```

Listing 5.35: Definition of curve data type

The ApplicationPrimitiveDataType of category CURVE defined in the listing above may be used as the element type (ApplicationArrayElement.type) of an ApplicationArrayDataType. The definition of this array data type is not part of the example.

The initialization of a DataPrototype typed by such an ApplicationArrayDataType is sketched in the listing below:

```
<CONSTANT-SPECIFICATION>
  <SHORT-NAME>MyConst</SHORT-NAME>
  <VALUE-SPEC>
    <ARRAY-VALUE-SPECIFICATION>
```

```
        <ELEMENTS>
          <COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
            <RULE>FILL_UNTIL_END</RULE>
            <COMPOUND-PRIMITIVE-ARGUMENTS>
              <APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
                <SW-AXIS-CONTS>
                  <RULE-BASED-AXIS-CONT>
                    <SW-ARRAYSIZE>
                      <V>10</V>
                    </SW-ARRAYSIZE>
                    <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
                    <RULE-BASED-VALUES>
                      <ARGUMENTSS>
                        <RULE-ARGUMENTS>
                          <V>1</V>
                          <V>2</V>
                          <V>3</V>
                          <V>4</V>
                          <V>5</V>
                          <V>6</V>
                          <V>7</V>
                          <V>8</V>
                          <V>9</V>
                          <V>10</V>
                        </RULE-ARGUMENTS>
                      </ARGUMENTSS>
                    </RULE-BASED-VALUES>
                  </RULE-BASED-AXIS-CONT>
                </SW-AXIS-CONTS>
                <SW-VALUE-CONT>
                  <SW-ARRAYSIZE>
                    <V>10</V>
                  </SW-ARRAYSIZE>
                  <RULE-BASED-VALUES>
                    <RULE>FILL_UNTIL_END</RULE>
                    <ARGUMENTSS>
                      <RULE-ARGUMENTS>
                        <V>0</V>
                      </RULE-ARGUMENTS>
                    </ARGUMENTSS>
                  </RULE-BASED-VALUES>
                </SW-VALUE-CONT>
              </APPLICATION-RULE-BASED-VALUE-SPECIFICATION>
            </COMPOUND-PRIMITIVE-ARGUMENTS>
          </COMPOSITE-RULE-BASED-VALUE-SPECIFICATION>
        </ELEMENTS>
      </ARRAY-VALUE-SPECIFICATION>
    </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

Listing 5.36: Definition of constant to initialize an array of curve data type

5.7 Initial Values

5.7.1 Overview

[TPS_SWCT_01301] Importance of initial values ⌈If the value of a VariableDataPrototype/ParameterDataPrototype has not properly been set by a piece of software, it can still happen that another piece of software tries to access the value of the VariableDataPrototype/ParameterDataPrototype.

For various reasons it is therefore advised to be able 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.⌋*()*

Therefore, the AUTOSAR standard foresees means for defining initial values for VariableDataPrototypes/ParameterDataPrototypes on different conceptual levels.

That is, although defined for the same VariableDataPrototype/ParameterDataPrototype, an initial value defined on one conceptual level can "supersede" the definition of another initial value on a different conceptual level provided that the priority of the first is higher than the priority of the latter.

The meaning of "supersede" in this context is that the definition of an initial value on a specific conceptual level is the only relevant definition of an initial value on that level.

[TPS_SWCT_01518] Priority of initial value definition with respect to conceptual levels ⌈Any initial value defined in the context of a conceptual level of lower priority is ignored!⌋*()*

[TPS_SWCT_01855] Definition of initial value for data transmission ⌈Initial values for sender/receiver-based data transmission (i.e. intra-Ecu as well as inter-Ecu) shall only be defined by means of NonqueuedSenderComSpec.initValue resp. NonqueuedReceiverComSpec.initValue. Any definition of an initValue defined in the context of VariableDataPrototype shall be ignored.⌋*()*

[TPS_SWCT_01182] Conceptual levels for the definition of initial values ⌈The following conceptual levels for the definition of initial values exist:

1. It is possible to aggregate an initValue directly at the definition of any VariableDataPrototype/ParameterDataPrototype. A restriction applies, please consult with [TPS_SWCT_01855].
2. It is possible to aggregate an initValue at the level of a ComSpec, namely:
   - NonqueuedSenderComSpec
   - NonqueuedReceiverComSpec
   - ParameterProvideComSpec
   - ParameterRequireComSpec

- NvRequireComSpec
- NvProvideComSpec

3. It is possible to aggregate a implInitValue and an applInitValue at the definition of a CalibrationParameterValue.

The priority of one definition of an initial value over another is reflected by the numerical order of the above enumeration, e.g. a definition on level 3 supersedes a definition on level 2.⌋*()*

5.7.2 Initial Value Representation

[TPS_SWCT_01183] Actual value of an initValue shall be interpreted according to the AutosarDataType ⌈A DataPrototype can be typed by either an ApplicationDataType or else an ImplementationDataType. Therefore, the actual value of an initValue shall be interpreted according to the AutosarDataType that types the DataPrototype.

That is, if the DataPrototype is typed by an ApplicationDataType, the value shall be interpreted as a physical value while, if the DataPrototype is typed by an ImplementationDataType, the value is to be interpreted as the direct numerical representation.⌋*(RS_SWCT_03216, RS_SWCT_03217)*

[TPS_SWCT_01184] ApplicationPrimitiveDataTypes with category VALUE ⌈In case of ApplicationPrimitiveDataTypes with category VALUE it is sufficient if the initValues are provided as physical values only because the RTE Generator should be able to evaluate the related CompuMethod appropriately.⌋*(RS_SWCT_03216, RS_SWCT_03217)*

Please note that DataPrototypes that refer to CompuMethods of category SCALE_LINEAR_AND_TEXTTABLE (or similar) shall be initialized by means of the definition of several ApplicationValueSpecification.swValueCont.swValuesPhys.vtf.

Depending on the evaluation of the binding expression either a numerical value or a string is taken to initialize the DataPrototype.

[TPS_SWCT_01185] initValues for Compound Primitive Data Types ⌈The definition of initValues in the numerical representation for Compound Primitive Data Type is done such that the initValues have to be provided as a RecordValueSpecification respectively an ArrayValueSpecification or NumericalRuleBasedValueSpecification matching to the related ImplementationDataType. The additional representation can be provided and associated by means of a ConstantSpecificationMapping.⌋*(RS_SWCT_03216)*

Please note that the definition of Compound Primitive Data Type can be found in section 5.6.

[constr_1221] DataPrototype is typed by an ApplicationPrimitiveDataType ⌈If a DataPrototype is typed by an ApplicationPrimitiveDataType, its initValue shall be provided by an ApplicationValueSpecification.

If the underlying ApplicationPrimitiveDataType represents an enumeration, the value provided shall match to one of the applicable text values (vt, shortLabel, symbol) defined by the applicable CompuScales.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note that several attributes of meta-class CompuScale can be taken to describe the actual value. It is therefore necessary to clarify what happens if several of these attributes exist within the context of one CompuScale. This clarification can be found in [TPS_SWCT_01696].

[constr_1385] DataPrototype is typed by an ImplementationDataType ⌈If a DataPrototype is typed by an ImplementationDataType, its initValue shall **not** be provided by an ApplicationValueSpecification.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1222] category of an AutosarDataType used to type a DataPrototype is set to STRING ⌈If the category of an AutosarDataType used to type a DataPrototype is set to STRING, the ApplicationValueSpecification used to initialize the DataPrototype shall be of category STRING.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1223] DataPrototype is typed by an ApplicationRecordDataType ⌈If a DataPrototype is typed by an ApplicationRecordDataType, the corresponding initValue shall be provided by a RecordValueSpecification.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1224] DataPrototype is typed by an ApplicationArrayDataType ⌈If a DataPrototype is typed by an ApplicationArrayDataType, the corresponding initValue shall be provided by an ArrayValueSpecification (that may contain an ApplicationRuleBasedValueSpecification).

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

5.7.3 Initial Values For CalibrationParameters

[TPS_SWCT_01188] Definition of calibration data sets through RTE-generator and compiler ⌈It is possible to provide sets of initial values for calibration parameters which are instance specific, thus overriding any initial values predefined by a ParameterDataPrototype, ParameterRequireComSpec or a ParameterProvideComSpec.

This allows to create the calibration data sets through RTE-generator and compiler. These initial values are specified in CalibrationParameterValueSet and CalibrationParameterValue. The latter aggregates a ValueSpecification in two different roles:

- applInitValue for data structured according to ApplicationDataType. In this case the values are defined in the physical domain.
- implInitValue for data structured according to ImplementationDataType. In this case the values are defined in the numerical domain.

⌋*(RS_SWCT_03175)*

Anyhow, these initial values can be imported from e.g. an ASAM CDF file.

Class	**CalibrationParameterValueSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::CalibrationParameter Values			
Note	Specification of a constant that can be part of a package, i.e. it can be defined stand-alone. **Tags:**atp.recommendedPackage=CalibrationParameterValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
calibration ParameterValue	CalibrationParameter Value	*	aggr	This represents single CalibrationParameterValues in the CalibrationParameterValueSet. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=calibrationParameterValue, calibration ParameterValue.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 5.138: CalibrationParameterValueSet

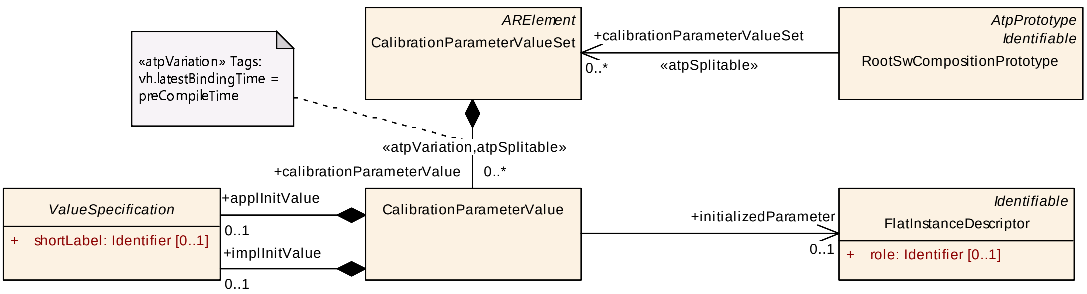


Figure 5.87: Calibration Parameter Values

Class	CalibrationParameterValue			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::CalibrationParameterValues			
Note	Specifies instance specific calibration parameter values used to initialize the memory objects implementing calibration parameters in the generated RTE code. RTE generator will use the implInitValue to override the initial values specified for the DataPrototypes of a component type. The applInitValue is used to exchange init values with the component vendor not publishing the transformation algorithm between ApplicationDataTypes and ImplementationDataTypes or defining an instance specific initialization of components which are only defined with ApplicationDataTypes. Note: If both representations of init values are available these need to represent the same content. Note further that in this case an explicit mapping of ValueSpecification is not implemented because calibration parameters are delivered back after the calibration phase.			
Base	*ARObject*			
Aggregated by	CalibrationParameterValueSet.calibrationParameterValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
applInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ApplicationDataType
implInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ImplementationDataType
initialized Parameter	FlatInstanceDescriptor	0..1	ref	This represents the parameter that is initialized by the CalibrationParameterValue.

Table 5.139: CalibrationParameterValue

[constr_1933] Existence of `CalibrationParameterValue.initializedParameter` ⌈For each `CalibrationParameterValue`, the reference to meta-class `ConstantSpecification` in the role `initializedParameter` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

5.7.4 Initial Value for optional Element

The existence of optional elements in a given `AutosarDataType` needs to be properly considered for the definition of an initial value.

5.7.4.1 Initial Value for optional ApplicationRecordElement

The initial value for a given `DataPrototype` typed by an `ApplicationRecordDataType` is defined by means of a `RecordValueSpecification`.

[TPS_SWCT_01823] Definition of `ValueSpecification` for an `ApplicationRecordDataType` with unavailable optional elements ⌈If an `ApplicationRecordDataType` contains `ApplicationRecordElement`s where attribute `isOptional` is set to `true`, it is still necessary that the corresponding `RecordValueSpecification`s fulfill [constr_1271], i.e. a `ValueSpecification` shall be provided for optional elements that are unavailable as far as the `ValueSpecification` is concerned.

The canonical approach to fulfill [constr_1271] is to fill the place in the structure that represents a non-available optional element of the structure with a special kind of ValueSpecification (the NotAvailableValueSpecification) in order to unambiguously convey the information that this element is not available for the specific DataPrototype, as far as the initial value is concerned.⌋*(RS_SWCT_03320)*

[TPS_SWCT_01821] Semantics of attribute NotAvailableValueSpecification.defaultPattern ⌈The usage of NotAvailableValueSpecification explicitly defines that no specific initialization value shall be defined for an optional element.

The memory area occupied by the NotAvailableValueSpecification shall however be filled with a deterministic pattern during initialization.

The content of defaultPattern shall be used to fill the gap in the memory occupied by an optional element with NotAvailableValueSpecification initialization.⌋*(RS_SWCT_03320)*

[TPS_SWCT_01822] Application of attribute NotAvailableValueSpecification.defaultPattern happens only during initialization ⌈The value of NotAvailableValueSpecification.defaultPattern is applied only during the initialization of the respective DataPrototype.

Therefore, if the optional element is not received during a specific reception, the memory area is untouched. It is the duty of the application to check whether that optional element has actually been received.⌋*(RS_SWCT_03320)*

[constr_10005] Existence of attribute NotAvailableValueSpecification.defaultPattern ⌈For each NotAvailableValueSpecification, attribute defaultPattern shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_10006] Valid interval of attribute NotAvailableValueSpecification.defaultPattern ⌈The valid interval for attribute NotAvailableValueSpecification.defaultPattern **at the time when the contract phase generation is executed** is 0..255.⌋*()*

5.7.4.2 Initial Value for optional ImplementationDataTypeElement

[TPS_SWCT_01785] Initial value for ImplementationDataType of category STRUCTURE where attribute isStructWithOptionalElement set to the value true ⌈If an initial value is to be provided for an ImplementationDataType of category STRUCTURE where attribute isStructWithOptionalElement set to the value true then an initial value shall be defined for all ImplementationDataTypeElements including the first ImplementationDataTypeElement where the shortName is set to the value availabilityBitfield.⌋*(RS_SWCT_03320)*

[TPS_SWCT_01786] Initial value for the ImplementationDataTypeElement where the shortName is set to the value availabilityBitfield ⌈The initial value for the ImplementationDataTypeElement where the shortName is set to the value availabilityBitfield shall be defined in a way that the bit that represents the existence of a given element is set to true if the element shall initially be available.

If the corresponding element shall not be initially available then the respective bit shall be set to false.⌋*(RS_SWCT_03320)*

[TPS_SWCT_01787] Initialization of not-available ImplementationDataType-Element ⌈If a given ImplementationDataTypeElement is not available in the context of the definition of an initial value, then a "dummy" initial value shall be defined anyway for the element in order to not break [constr_1272].

The provided ValueSpecification shall be considered as "don't care".⌋*()*

From the perspective of performance, it is recommended to use the value 0 (on the level of a leaf element) for an initialization according to [TPS_SWCT_01787] of non-available ImplementationDataTypeElement.

6 Compatibility

6.1 Introduction

In order to connect PortPrototypes of SwComponentTypes, the compatibility of PortPrototypes needs to be verified. This section defines the basic rules for formal compatibility of PortPrototypes.

Compatibility will be defined bottom-up, i.e. first the rules for compatible AutosarDataTypes are set up, then the rules for the different types of PortInterfaces are derived.

Another aspect of compatibility is whether two model-elements (e.g. ApplicationDataType vs. ImplementationDataType) can be mapped to each other.

For the compatibility of PortInterfaces basically two options apply:

1. finding of matching pairs of elements of PortInterfaces is based on matching shortName plus the application of compatibility rules for their attributes.
2. a PortInterfaceMapping can be taken to declare two elements of PortPrototypes as compatible without applying further formal checks.

6.2 Compatibility of Data Types

The AUTOSAR meta-model defines a number of meta-classes (e.g. ApplicationPrimitiveDataType) that eventually refer to a set of attributes (e.g. a lower boundary for its values) relevant for compatibility checking.

Instantiating a data-type related meta-class defines a data type on M1 level (e.g. *temperatureType*). In other words: ApplicationPrimitiveDataType is an M2 artifact; it is taken as the template for creating a corresponding M1 artifact *temperatureType*.

In this context, the issue of compatibility refers to the M1 objects, i.e. the instances of sub-classes of AutosarDataType need to be considered. For this purpose the relevant part of the AUTOSAR meta-model need to be fully explored with respect to compatibility.

6.2.1 ApplicationDataType

6.2.1.1 ApplicationPrimitiveDataType

[constr_1047] Compatibility of ApplicationPrimitiveDataTypes ⌈Instances of ApplicationPrimitiveDataType are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All the following sub conditions apply:

   (a) They have the same category.

   (b) The swDataDefProps (after consideration of [constr_1015]) attached to the M1 data types are compatible.

2. In the context of using the ApplicationPrimitiveDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by one of the ApplicationPrimitiveDataTypes in the role firstDataPrototype and to another DataPrototype typed by the other ApplicationPrimitiveDataType in the role secondDataPrototype.

3. In the context of using the ApplicationPrimitiveDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by the ApplicationPrimitiveDataType in the role secondDataPrototype and to another DataPrototype typed by an ApplicationCompositeDataType in the role firstDataPrototype and additionally for the side of the ApplicationCompositeDataType a corresponding ApplicationCompositeDataTypeSubElementRef exists in the role firstElement that in turn references an ApplicationCompositeElementDataPrototype.

⌋*()*

Please note that the meaning of “swDataDefProps attached to the M1 data types are compatible” is explained in section 6.2.4.

Please note further that it is **not** required that the shortNames of two data types shall be identical in order to consider the two data types as compatible.

6.2.1.2 ApplicationCompositeDataType

An instance of an ApplicationRecordDataType is never compatible to an instance of an ApplicationArrayDataType **unless** a PortInterfaceMapping exists that details the terms of compatibility (see [TPS_SWCT_01543]).

[constr_1048] Compatibility of ApplicationRecordDataTypes ⌈Instances of ApplicationRecordDataTypes are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All elements *at the same record position* are of compatible AutosarDataTypes (either ApplicationCompositeDataTypes or ApplicationPrimitiveDataTypes).

2. For each ApplicationRecordDataType.element, the attribute isOptional shall either

   - not exist on both sides or
   - be set to the value false if it only exists on one side or
   - have the identical value on both sides.

3. In the context of a DataPrototypeMapping, for each ApplicationRecordElement of the required ApplicationRecordDataType a SubElementMapping exists such that a ApplicationCompositeDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ApplicationRecordElement **and** a corresponding ApplicationCompositeDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ApplicationRecordElement of the provided ApplicationRecordDataType.

⌋*()*

[constr_1049] Compatibility of ApplicationArrayDataTypes ⌈Instances of ApplicationArrayDataType are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All the following sub conditions apply:

   (a) Their elements are of a compatible AutosarDataTypes (either ApplicationCompositeDataTypes or ApplicationPrimitiveDataTypes).

   (b) The attributes maxNumberOfElements and arraySizeSemantics (given the existence) have identical values.

2. In the context of a DataPrototypeMapping, for the ApplicationArrayElement of the required ApplicationArrayDataType a SubElementMapping exists such that a ApplicationCompositeDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ApplicationArrayElement **and** a corresponding ApplicationCompositeDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ApplicationArrayElement of the provided ApplicationArrayDataType.

⌋*()*

6.2.2 ImplementationDataType

[constr_1050] Compatibility of ImplementationDataTypes ⌈Instances of ImplementationDataType are compatible **at the time when the RTE is generated** if and only if after all type-references are resolved one of the following rules apply:

1. All the following sub conditions apply:

   (a) They have the same category.

   (b) They have the identical structure (this refers to ImplementationDataTypeElement and their subElements).

(c) The attributes arraySize and arraySizeSemantics have (given the existence) identical values.

(d) For each ImplementationDataType.subElement, the attribute isOptional shall either

- not exist on both sides or
- be set to the value false if it only exists on one side or
- have the identical value on both sides.

(e) The swDataDefProps (after consideration of [constr_1015]) attached to the M1 data types are compatible.

2. In the context of using the ImplementationDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by one of the ImplementationDataTypes in the role firstDataPrototype and to another DataPrototype typed by the other ImplementationDataType in the role secondDataPrototype.

3. In the context of using the ImplementationDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by the ImplementationDataTypes in the role secondDataPrototype and to another DataPrototype typed by an ImplementationDataType with a subElement in the role firstDataPrototype and additionally for the side of the ImplementationDataType with a subElement a corresponding ImplementationDataTypeSubElementRef exists in the role firstElement that in turn references an ImplementationDataTypeElement.

⌋*()*

Please note that the meaning of "swDataDefProps attached to the M1 data types are compatible" is explained in section 6.2.4.

Please note that it is **not** required that the shortNames of two data types shall be identical in order to consider the two data types as compatible.

The following constraint applies for the case that mode manager and mode user are using different ImplementationDataTypes. From the point of view of the RTE there is only the necessity that all possible numbers used to represent ModeDeclarations of the mode manager has to fit into the range of the data type used for the mode user.

[constr_1168] Compatibility of ImplementationDataTypes used in the ModeRequestTypeMap ⌈Both ImplementationDataTypes shall fulfill [constr_1167].

In addition to that, the possible numbers used for representing ModeDeclarations on the side of the mode manager shall match the supported range of the ImplementationDataType used for representing ModeDeclarations on the side of the mode user (see [constr_1075]).

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

6.2.3 Compatibility of SwBaseType

[constr_1220] Compatibility of SwBaseType ⌈Two SwBaseTypes are compatible **at the time when the RTE is generated** if and only if attributes

- baseTypeSize respectively
- byteOrder,
- memAlignment,
- baseTypeEncoding, and
- nativeDeclaration

have identical values **at the time when the RTE is generated**.⌋*()*

6.2.4 Compatibility of SwDataDefProps

[constr_1051] Compatibility of SwDataDefProps ⌈SwDataDefProps are compatible **at the time when the RTE is generated** if and only if:

1. They refer to compatible Unit definitions, or neither of them has an associated Unit.
2. They refer to compatible conversion methods or neither of them associates such a method.
3. They both aggregate a ValueSpecification in the role invalidValue or neither of them aggregates a ValueSpecification in the role invalidValue.
4. If existent (see previous condition), one of the following conditions apply to ValueSpecifications aggregated in the role invalidValue for being considered compatible (after following and resolving indirections created by ConstantReference):
   (a) both are ApplicationValueSpecifications and the values are compatible according to [TPS_GST_02501].
   (b) both are NumericalValueSpecifications and the values are compatible according to [TPS_GST_02501].
   (c) both are TextValueSpecifications and the values are identical.
   (d) both are ArrayValueSpecifications and the values are effectively identical, e.g. if one ArrayValueSpecification specifies all values explicitly and the other ArrayValueSpecification specifies values based on a rule then the yield of both ArrayValueSpecifications (i.e. element for element) shall be identical.
   (e) both are RecordValueSpecifications and the values are identical.

(f) if one is a NumericalValueSpecification and the other one is an ApplicationValueSpecification then the check for compatibility shall apply the CompuMethod on the physical value such that a comparison on the implementation level becomes possible. [TPS_GST_02501] applies[1].

5. They refer to compatible data constraints dataConstr.

6. They refer to compatible swRecordLayouts

All other attributes (e.g. swCalibrationAccess do not affect compatibility).⌋*()*

Please note that compatible conversion methods are described in chapter 6.2.4.5.

6.2.4.1 Compatibility of Units

[constr_1052] Compatibility of Units ⌈Two Unit definitions are compatible **at the time when the RTE is generated** if and only if:

1. They have compatible (see [TPS_GST_02501]) values of attributes factorSiToUnit and offsetSiToUnit.

2. One of the following conditions is fulfilled:

   - They refer to compatible definitions of PhysicalDimension.
   - Neither of them associates a PhysicalDimension.
   - One Unit refers to a PhysicalDimension with shortName NoDimension where all exponents are set to 0 and the other Unit does not refer to a PhysicalDimension.

⌋*()*

Please note that the case of two referenced PhysicalDimensions with the shortName NoDimension is handled by [constr_1053].

Please note further that it is **not** required that the shortNames of two Units shall be identical in order to consider the two units as compatible.

[TPS_SWCT_01492] Default values for factorSiToUnit and offsetSiToUnit ⌈The default value of attribute Unit.factorSiToUnit is 1.

The default value of attribute Unit.offsetSiToUnit is 0.⌋*()*

Further constraints apply specifically for the handling of Units in the context of assigning a ValueSpecification to a given AutosarDataPrototype:

[constr_1391] Compatibility of Units in the context of assignment using an ApplicationValueSpecification ⌈If an ApplicationValueSpecification is

[1] if one is a NumericalValueSpecification and the other one is an ApplicationValueSpecification and the application of the CompuMethod on the side of the ApplicationValueSpecification does not yield a valid number a comparison is not possible.

used in the context of an assignment to an AutosarDataPrototype, then the ApplicationValueSpecification.swValueCont.unit shall be compatible to the Unit used in the definition of the given AutosarDataPrototype, i.e. AutosarDataType.swDataDefProps.unit.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1392] Compatibility of Units in the context of assignment using an ApplicationRuleBasedValueSpecification ⌈If an ApplicationRuleBasedValueSpecification is used in the context of an assignment to an AutosarDataPrototype then the ApplicationRuleBasedValueSpecification.swValueCont.unit shall be compatible to the Unit used in the definition of the given AutosarDataPrototype, i.e. AutosarDataType.swDataDefProps.unit.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[constr_1393] Existence of RuleBasedValueCont.unit ⌈For every RuleBasedValueCont, the reference unit shall exist **at the time when the contract phase generation is executed**.⌋()

[constr_1771] Existence of SwValueCont.unit ⌈For every SwValueCont, the reference unit shall exist **at the time when the contract phase generation is executed**.⌋()

6.2.4.2 Compatibility of PhysicalDimensions

[constr_1053] Compatibility of PhysicalDimensions ⌈Two PhysicalDimension definitions are compatible if and only if the values of

- lengthExp
- massExp
- timeExp
- currentExp
- temperatureExp
- molarAmountExp
- luminousIntensityExp

are identical and **either**

- the shortNames are identical **or**
- a PhysicalDimensionMapping exists that maps one of the PhysicalDimensions in the role firstPhysicalDimension and the other PhysicalDimension in the role secondPhysicalDimension.

The imposition time of this constraint depends on the context:

- If the compatibility of PhysicalDimensions is evaluated in the context of the creation of a SwConnector, then the rule shall be imposed **at the time when the RTE is generated**.
- If the context is the creation of an ApplicationValueSpecification, then the rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

For clarification, there are some physical dimensions around that share the identical values for the exponents but still have a completely different meaning and shall therefore not be considered compatible. For precisely this reason [constr_1053] **requires** the shortNames of two PhysicalDimensions to be identical as a prerequisite for compatibility.

For example, there are at least two physical dimensions that share the values of

- lengthExp = 2
- massExp = 1
- timeExp = -2
- currentExp = 0
- temperatureExp = 0
- molarAmountExp = 0
- luminousIntensityExp = 0

The unit described by this set of exponents is usually referred to as “Nm” for *newton-meter* and it can be used for *torque* just as well as for *energy*. Obviously, two Units shall never be considered compatible if one refers to *torque* and the other one refers to *energy*.

6.2.4.3 Compatibility of Data Constraints

The compatibility of two DataConstrs depends on the context in which the owning data elements are connected:

[constr_1126] Compatibility of DataConstrs ⌈The DataConstr (e.g. the limits) defined by the type of the providing data element shall be within the constraints defined by the type of the requiring data element.

For client-server communication, the following rules apply:

- For arguments with attribute direction set to the value in, the client shall take the role of the *provider* and the server shall take the role of the *requiring side*.
- For arguments with attribute direction set to the value inoutthe DataConstr shall be equal on both sides.
- For arguments with attribute direction set to the value out, the server shall take the role of the *provider* and the client shall take the role of the *requiring side*.

This rule shall be applied **at the time when the RTE is generated**.⌋*()*

In addition, it is always allowed that the requiring element defines no constraints.

[constr_1278] PhysConstrs references a Unit ⌈DataConstrs are only compatible if the DataConstr.dataConstrRule.physConstrs.unit are compatible or neither DataConstr.dataConstrRule.physConstrs.unit exist **at the time when the RTE is generated**.⌋*()*

[constr_1054] No DataConstr available at the provider ⌈If the provider defines no constraints, it is only compatible with a receiver which also defines no constraints at all **at the time when the RTE is generated**.⌋*()*

In other words, this is not a compatibility rule for the types but for the data prototypes.

6.2.4.4 Compatibility in case of ImplementationDataType

If the SwDataDefProps are owned by an ImplementationDataType, further conditions shall be met to ensure compatibility.

Note that depending on the category of the ImplementationDataType, at most one of these four constraints is actually relevant:

1. category **VALUE**:

   [constr_1055] ImplementationDataType has category VALUE ⌈The attributes baseType shall refer to a compatible SwBaseType **at the time when the contract phase generation is executed**.⌋*()* (see explanation in the following rule). The rules regarding the compatibility of SwBaseTypes are covered by [constr_1220].

2. category **TYPE_REFERENCE**:

   [constr_1056] ImplementationDataType has category TYPE_REFERENCE ⌈The ImplementationDataTypes referenced by the attributes SwDataDefProps.implementationDataType shall be compatible **at the time when the contract phase generation is executed**.⌋*()*

3. category **DATA_REFERENCE**:

[constr_1057] **ImplementationDataType** **has** **category DATA_REFERENCE** ⌈The attributes SwDataDefProps.swPointerTargetProps shall have identical targetCategory and shall refer to SwDataDefProps where all attributes are identical **at the time when the contract phase generation is executed**.⌋*()* (in other words, the target types of the pointers shall be identical, not only compatible).

4. category **FUNCTION_REFERENCE**:

   [constr_1058] **ImplementationDataType** **has** **category FUNCTION_REFERENCE** ⌈The attributes SwDataDefProps.swPointerTargetProps.functionPointerSignature shall refer to BswModuleEntrys which each resolve to the **same function signature** **at the time when the contract phase generation is executed**.⌋*()*

   Please note that the term “same signature” refers to the following predicates:

   - same number of arguments
   - return values and arguments shall have **identical** - not only *compatible* - data types

Two SwBaseTypes are compatible (in the sense of allowing a connection of ports via the RTE) if a simple conversion rule exists between the two types in the underlying programming language.

Admittedly, this is a rather weak condition. But because the definition of SwBaseTypes can contain a nativeDeclaration it is not possible to state this rule more specifically.

However, conversion between base types is considered as a less common use case than the simple case that the connected types just contain two identical SwBaseTypes (which is of course included in the rule).

Please note that, in addition, the existence of ApplicationDataTypes also constraints the possible SwBaseTypes via the compatibility rules for the mapping between ApplicationDataTypes and ImplementationDataType as will be explained in more detail in chapter 6.2.5.

6.2.4.5 Compatibility of CompuMethods

[constr_1163] **Compatibility of** **CompuMethods** ⌈Two CompuMethod definitions are compatible **at the time when the RTE is generated** if and only if all attributes **except**

- shortName
- desc
- introduction

- longName
- adminData
- annotation
- displayFormat

are **identical and** the compuScales and units are compatible.⌋()

[constr_1153] Applicability of compatibility requirements for CompuScales ⌈Compatibility requirements for CompuScales shall only apply for CompuScales where the category of the enclosing CompuMethod is one of the following:

- TEXTTABLE
- SCALE_LINEAR_AND_TEXTTABLE
- SCALE_RATIONAL_AND_TEXTTABLE
- TAB_NOINTP
- BITFIELD_TEXTTABLE
- LINEAR
- RAT_FUNC
- IDENTICAL

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1154] Compatibility of CompuScales for sender-receiver communication and similar use cases ⌈For sender-receiver communication and similar use cases, it is required that the set of CompuScales defined in the CompuMethod of the provider of the communication (i.e. on the side of the PPortPrototype) shall be a subset of the set of CompuScales defined in the CompuMethod on the required side (i.e. on the side of the RPortPrototype) **at the time when the RTE is generated**.⌋()

[constr_1155] Compatibility of CompuScales for client-server communication ⌈For client-server communication, the following rules apply **at the time when the RTE is generated**:

For arguments of direction IN the CompuScales defined in the CompuMethod of the client (i.e. on the side of the RPortPrototype) shall be a subset of the set of CompuScales defined in the CompuMethod supported at the server (i.e. on the side of the PPortPrototype).

For arguments of the direction OUT the set of CompuScales defined in the CompuMethod of the server (i.e. on the side of the PPortPrototype) shall be a subset of the set of CompuScales defined in the CompuMethod supported at the client (i.e. on the side of the RPortPrototype).

For arguments of direction INOUT the set of CompuScales defined in the CompuMethod of server and client shall be identical.⌋()

[constr_1156] Relevance of "names" of CompuScales ⌈CompuScales which contribute to tabular conversion by having a compuConst are compatible **if and only if** the "names" of the compuScales, (namely shortLabel, vt and symbol, according to the priority rules communicated in [TPS_SWCT_01431]) are equal.

If the scale has no compuConst, "names" of CompuScales are not relevant for compatibility.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01842] Applicability of constraints of CompuScales ⌈The constraints [constr_1154], [constr_1155], and [constr_1156] shall **only** apply in the absence of a TextTableMapping which shall take precedence regarding the compatibility if it exists.⌋*()*

[constr_1176] Compatibility of CompuScales of category LINEAR and RAT_FUNC ⌈CompuScales of category LINEAR and RAT_FUNC are considered compatible **at the time when the RTE is generated** if they yield the same conversion.⌋*()*

For example, $\frac{n_0+n_1*phys}{d_0+d_1*phys}$ is compatible to $\frac{N_0+N_1*phys}{D_0}$ if $n_0 \sim N_0$ && $n_1 \sim N_1$ && $d_0 \sim D_0$ && $d_1 \sim 0$.

Note that $\sim$ indicates compatibility of numerical values according to [TPS_GST_-02501].

[constr_1192] Compatibility of "IDENTICAL" to "RAT_FUNC" or "LINEAR" ⌈Similar to [constr_1176], a CompuScale where the category of the enclosing CompuMethod is set to IDENTICAL is – **at the time when the RTE is generated** – considered compatible to a CompuScale where the category of the enclosing CompuMethod is set to RAT_FUNC or LINEAR if the following rule applies:

$$int = \frac{N_0+N_1*phys+N_i*phys^i}{D_0+D_1*phys+D_i*phys^i} = phys$$

⌋*()*

For example, this is the case for

$N_0 \sim 0$ && $D_0 \sim 1$ && $N_1 \sim 1$ && $D_1 \sim 0$ && $N_i \sim D_i \sim 0\ \forall i > 1$.

Please note that the compatibility does not depend on the direction (compuInternalToPhys vs. compuPhysToInternal) of CompuMethods of category LINEAR.

6.2.5 Compatibility of ApplicationDataType and ImplementationDataType

The usage of ApplicationDataTypes implies that also a corresponding ImplementationDataType exists at a certain point in time. The ImplementationDataType is required as the basis for configuring and generating the RTE and/or contract phase header files.

[TPS_SWCT_01461] Existence of ImplementationDataType ⌈The existence of ImplementationDataTypes is **not** required until the methodology step of generating an RTE or executing the RTE contract phase. Before arriving at this step in the methodology, it is perfectly feasible to use only ApplicationDataTypes for describing the semantics of software-components.⌋ *()*

As a consequence, it is necessary to define compatibility rules that unambiguously clarify the conformance of an ApplicationDataType with an ImplementationDataType and vice versa.

Please note that this kind of compatibility also supports situations where e.g. a dataElement typed by an ApplicationDataType without a corresponding ImplementationDataType in a PPortPrototype should be connected to a dataElement typed by an ImplementationDataType in an RPortPrototype.

In general, the compatibility rules for allowing a data type mapping are the same as the rules for connections. Exceptions are explicitly stated in the rules below.

Several rules depend on the category of the data types.

6.2.5.1 General Rule

As a general rule, if an ImplementationDataType of category TYPE_REFERENCE is targeted by a type mapping or port connection, all the rules given below apply to the ImplementationDataType which is finally valid after resolving all such references.

This is not repeated in all rules. As an example, if we say that something can be mapped/connected to an ImplementationDataType of category VALUE, then this shall include the possibility of mapping/connecting to an ImplementationDataType of category TYPE_REFERENCE which refers to another ImplementationDataType of category VALUE.

6.2.5.2 Category VALUE

[constr_1059] Compatibility of data types with category VALUE ⌈An ApplicationDataType of category VALUE shall (after all indirections created by ImplementationDataTypes of category TYPE_REFERENCE are resolved) only be mapped/connected to an ImplementationDataType which also has category VALUE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋ *()*

In this case, the ImplementationDataType.baseType shall be able to express all the numerical values required by the ApplicationDataType, see Figure 5.30.

This condition is fulfilled if the numerical range which can be expressed by the SwBaseType at least covers the range defined by the limits in ApplicationDataType.swDataDefProps.dataConstr (which are either internal limits or physical limits to be converted via the CompuMethod which also has to be provided by the ApplicationDataType).

Note that for sender-receiver communication of a data element via a network there is the possibility to reduce the numerical range against what has been defined via the corresponding data type. However, this is not achieved via mapping to another ImplementationDataType at the data element itself but via the networkRepresentation of the ComSpec (for further explanation of this aspect see section 4.5.1).

6.2.5.3 Category ARRAY, VAL_BLK

[constr_1060] Compatibility of data types with category ARRAY, VAL_BLK ⌈

ApplicationDataType	ImplementationDataType: Array of uint8	ImplementationDataType: Array of other
ApplicationArrayDataType of category ARRAY, VAL_BLK, arraySizeSemantics = fixedSize	ImplementationDataType of category ARRAY, with ImplementationDataTypeElement with arraySizeSemantics = fixedSize	ImplementationDataType of category ARRAY, with ImplementationDataTypeElement with arraySizeSemantics = fixedSize
ApplicationArrayDataType of category ARRAY, VAL_BLK, arraySizeSemantics = variableSize	ImplementationDataType of category ARRAY, with ImplementationDataTypeElement with arraySizeSemantics = variableSize or Variable-Size Array Data Type	Variable-Size Array Data Type

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

In this case, the array size, the arraySizeSemantics (given that it exists) and the type of the array elements of the ImplementationDataType shall be such that they can be mapped/transferred 1:1 by order to the corresponding application data and vice versa.

Note that in case of mapping between arrays it is not required that a DataTypeMap exists between the data types of the array elements or that the respective shortNames are identical.

6.2.5.4 Category STRUCTURE

[constr_1061] Compatibility of data types with category STRUCTURE ⌈An ApplicationDataType of category STRUCTURE shall (after all indirections created by ImplementationDataTypes of category TYPE_REFERENCE are resolved) only be mapped/connected to an ImplementationDataType of category STRUCTURE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

This means, that the corresponding pairs of elements shall also have compatible types. Note that it is not required that the data types of the single elements have identical shortNames or that a DataTypeMap exists for each pair of single element.

[constr_1662] Compatibility of ApplicationRecordDataType and ImplementationDataType that both represent an Optional Element Structure ⌈An ApplicationRecordDataType that represents an Optional Element Structure shall (after all indirections created by ImplementationDataTypes of category TYPE_REFERENCE are resolved) only be mapped/connected to an ImplementationDataType of category STRUCTURE that represents an Optional Element Structure if corresponding pairs of elements have the same value of the attribute isOptional.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

6.2.5.5 Category BOOLEAN

[constr_1063] Compatibility of data types with category BOOLEAN ⌈An ApplicationDataType of category BOOLEAN shall (after all indirections created by ImplementationDataTypes of category TYPE_REFERENCE are resolved) only be mapped/connected to an ImplementationDataType of category VALUE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

6.2.5.6 Category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5

[constr_1064] Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 ⌈An ApplicationDataType of category

- COM_AXIS,
- RES_AXIS,
- CURVE,
- MAP,
- CUBOID,

- CUBE_4, or
- CUBE_5

shall (after all indirections created by ImplementationDataTypes of category TYPE_REFERENCE are resolved) only be mapped/connected to an ImplementationDataType of category

- STRUCTURE or
- ARRAY.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

There are several possibilities how to express these types via plain or nested arrays and/or structures on implementation level.

Some examples are given in 5.4.4. In any case, the primitive elements of the implementation type shall fit (by their order in memory) to the corresponding SwRecordLayout.

It is not required, to define DataTypeMaps for the sub-elements or both representations.

6.2.5.7 Forbidden Mappings

[constr_1066] Forbidden mappings to ImplementationDataType ⌈An ApplicationDataType shall never be mapped to

- an ImplementationDataType of category
  - UNION,
  - DATA_REFERENCE, or
  - FUNCTION_REFERENCE,
- or to an ImplementationDataType that contains subElements of category
  - UNION,
  - DATA_REFERENCE, or
  - FUNCTION_REFERENCE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Concerning the SwDataDefProps of an ApplicationDataType instance or an ImplementationDataType instance which shall be mapped/connected on M1, see [constr_1015]. The following rules apply:

1. The cases where the ImplementationDataType is not allowed to set a property but only "inherits" it from the ApplicationDataType are not relevant for

compatibility. These attributes are simply not allowed in the ImplementationDataType.

2. In case that only the ImplementationDataType may “define” the property this definition shall fit into the semantical requirements given by the ApplicationDataType in order to make the two types compatible.

   This is namely important for the attribute baseType and is explained above in the rule for types of category VALUE.

3. In case the ImplementationDataType may “add” a property it may only add but not change a property defined by the ApplicationDataType (namely note, displayFormat, and swImplPolicy) in order to be compatible.

   This means that the respective computation methods can be defined in only one of the types in order to be compatible. In all other cases, only the ApplicationDataType may define the computation method.

4. For the compatibility with respect to connectors there are some additional rules for the values of the attribute swImplPolicy which are considered general rules on the level of DataPrototypes and PortInterfaces.

   Therefore, these additional rules are explained in chapter 6.3 and chapter 6.4.4.

5. The case that an ImplementationDataType may “redefine” a property which is already set by the ApplicationDataType is not considered as relevant for the compatibility with respect to mapping of the types in general but of course there may be project specific rules as to which redefinition is allowed (e.g. for swAddrMethod or dataConstr). See also 5.5.3 about data constraints.

6. For the compatibility with respect to connectors the attribute dataConstr shall be treated in the same way as for compatibility of data types in general, for more details please refer to 6.2.4.

6.3 Compatibility of Variable Data Prototypes and Parameter Data Prototypes

[constr_1068] Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types ⌈Two VariableDataPrototypes or ParameterDataPrototypes of ApplicationPrimitiveDataTypes or ImplementationDataTypes of category VALUE, BOOLEAN, or STRING are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All the following subconditions apply:

   (a) They are typed by (read “refer to”) compatible AutosarDataTypes

   (b) The two VariableDataPrototypes or ParameterDataPrototypes have identical shortNames. This is required to map VariableDataPrototypes in unordered SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces.

   (c) The attribute swImplPolicy is either set to queued for both or none of the VariableDataPrototypes.

2. In the context of a DataPrototypeMapping, one of the applicable VariableDataPrototypes or ParameterDataPrototypes is referenced by the DataPrototypeMapping in the role firstDataPrototype and the other VariableDataPrototypes or ParameterDataPrototypes is referenced by the same DataPrototypeMapping in the role secondDataPrototype.

⌋*()*

[constr_1187] Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types ⌈

DataPrototypes of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY are compatible **at the time when the RTE is generated** if one of the following conditions evaluates to true:

1. The underlying ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY are identical

2. The underlying ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY fulfill the following condition:

   - They consist of the same number of elements **and**

   - They are composed of compatible AutosarDataTypes (either ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY **OR** ApplicationPrimitiveDataTypes or ImplementationDataTypes of category VALUE, BOOLEAN, or STRING) in *the same order* **and**

   - All attributes match exactly, except for the shortName of the M1 AutosarDataType.

3. In the context of a DataPrototypeMapping, for each ApplicationCompositeElementDataPrototype of the required DataPrototype a SubElementMapping exists such that a ApplicationCompositeDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ApplicationCompositeElementDataPrototype **and** a corresponding ApplicationCompositeDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ApplicationCompositeElementDataPrototype of the provided ApplicationCompositeDataType.

4. If and only if the DataPrototype is **not** typed by an ApplicationDataType but by an ImplementationDataType: in the context of a DataPrototypeMapping, for each ImplementationDataTypeElement of the required DataPrototype a SubElementMapping exists such that a ImplementationDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ImplementationDataTypeElement **and** a corresponding ImplementationDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ImplementationDataTypeElement of the provided ImplementationDataType.

⌋*()*

6.4 Compatibility of Sender Receiver Interfaces, Parameter Interfaces and Non Volatile Data Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a receiver shall process a certain data value to correctly interpret the following values).

6.4.1 Connection of Required and Provided Port via AssemblySwConnector

The compatibility of SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces are considered for connecting of PortPrototypes with an AssemblySwConnector.

[constr_1069] Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors ⌈PortPrototypes of different DataInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:

   (a) For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required PortPrototype a compatible (see [constr_1068]) VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the provided PortPrototype.

   The shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

   (b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

ii. It references one of the two VariableDataPrototypes or ParameterDataPrototypes in the role firstDataPrototype and the other in the role secondDataPrototype.

2. For each such pair, the values of their isService attributes are identical.

⌋()

The constraint [constr_1071] defines which PortInterface elements are compatible depending on the PortInterface type and the swImplPolicy attributes of the PortInterface elements.

6.4.2 Connection of Inner and Outer Port via DelegationSwConnector

The compatibility of SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces is considered for connecting of PortPrototypes with a DelegationSwConnector.

[constr_1070] Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors ⌈PortPrototypes of different DataInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:

   (a) For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required inner PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the required outer PortPrototype.

   The shortName of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

   [constr_1071] defines which PortInterface elements are compatible depending on the PortInterface type and the swImplPolicy attributes of the PortInterface elements.

   (b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two VariableDataPrototypes or ParameterDataPrototypes in the role firstDataPrototype and the other in the role secondDataPrototype.

2. One of the following conditions applies:

(a) For at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided inner PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided outer PortPrototype.

The shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

[constr_1071] defines which PortInterface elements are compatible depending on the PortInterface type and the swImplPolicy attributes of the PortInterface elements.

(b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

i. It is (if a corresponding SwConnector already exists) referenced by the corresponding SwConnector.

ii. It references one of the two VariableDataPrototypes or ParameterDataPrototypes in the role firstDataPrototype and the other in the role secondDataPrototype.

3. For each such pair, the values of their isService attributes are identical.

⌋*()*

6.4.3 Connection of Required and Provided Port via PassThroughSwConnector

[constr_1248] Compatibility of PortPrototypes of different DataInterfaces in the context of a PassThroughSwConnector ⌈PortPrototypes of different DataInterfaces are considered compatible **at the time when the RTE is generated** if and only if

1. For **at least one** VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required outer PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the provided outer PortPrototype.

   Either the shortName of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair **or** a PortInterfaceMapping exists that defines which differently named elements of PortInterfaces correlate with each other.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋*()*

The constraint [constr_1071] defines which elements of PortInterface are considered compatible depending on the type of PortInterface as well as the attribute swImplPolicy of the elements of PortInterfaces.

6.4.4 Compatibility of ParameterDataPrototype and VariableDataPrototype depending on PortInterface Type

Table [constr_1071] contains a comprehensive description of which combinations of ParameterDataPrototype and VariableDataPrototype used in PortPrototypes typed by various kinds of PortInterfaces are considered compatible.

[constr_1071] compatibility of ParameterDataPrototype and VariableDataPrototype ⌈

Provided Port Required Outer Port Provided Inner Port Required Outer Port			Required Port / Required Inner Port / Provided Outer Port / Provided Outer Port					
PortInterface			Prm			S/R		NvD
Interface Element			PDP			VDP		VDP
		SwImplPolicyEnum	fixed	const	standard	standard	queued	standard
Prm	PDP	fixed	yes	yes	yes	yes	no	yes
		const	no	yes	yes	yes	no	yes
		standard	no	no	yes	yes	no	yes
S/R	VDP	standard	no	no	no	yes	no	yes
		queued	no	no	no	no	yes	no
NvD	VDP	standard	no	no	no	yes	no	yes

This rule shall be imposed **at the time when the RTE is generated**.

⌋()

The following legend applies for the abbreviations used in the table contained in [constr_1071]:

Interface Element i.e. elements of PortInterface

PDP ParameterDataPrototype

VDP VariableDataPrototype

Port Interface i.e. kind of PortInterface

Prm ParameterInterface

S/R SenderReceiverInterface

NvD NvDataInterface

[constr_1071] defines which PortInterface elements are compatible depending on the kind of PortInterface and the swImplPolicy attributes of the PortInterface elements.

[constr_1287] Compatibility of SenderReceiverInterfaces with respect to invalidationPolicy ⌈VariableDataPrototypes defined in the context of the SenderReceiverInterface are only compatible if the invalidationPolicys have the same value.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01567] Default behavior for invalidationPolicy ⌈For VariableDataPrototypes and ParameterDataPrototypes in the context of NvDataInterface respectively ParameterInterface, the invalidationPolicy is treated like “Invalidation is switched off” (dontInvalidate).⌋*(RS_SWCT_00200)*

6.5 Compatibility of Mode Switch Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a receiver shall process a certain data value to correctly interpret the following values).

Note that concerning the compatibility of ModeSwitchInterfaces it is necessary to distinguish between the context of an AssemblySwConnector, the context of an DelegationSwConnector, and the context of a PassThroughSwConnector.

6.5.1 Connection of Required and Provided Port via AssemblySwConnector

Here, the compatibility of ModeSwitchInterfaces is considered for the context of an AssemblySwConnector.

[constr_1072] Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector ⌈PortPrototypes of different ModeSwitchInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:
    (a) For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the required PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the provided PortPrototype.
    (b) A ModeInterfaceMapping.modeMapping exists for which the following conditions apply:
        i. It is referenced by the corresponding SwConnector.

ii. It references one of the two ModeDeclarationGroupPrototypes in the role firstModeGroup and the other in the role secondModeGroup.

2. For each such pair, the values of their isService attributes are identical.

⌋()

6.5.2 Connection of Inner and Outer Port via DelegationSwConnector

Here, the compatibility of ModeSwitchInterfaces is considered for the context of a DelegationSwConnector.

[constr_1073] Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector ⌈PortPrototypes of different ModeSwitchInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:

   (a) For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the inner PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the outer PortPrototype.

   (b) A ModeInterfaceMapping.modeMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two ModeDeclarationGroupPrototypes in the role firstModeGroup and the other in the role secondModeGroup.

2. For each such pair, the values of their isService attributes are identical.

⌋()

6.5.3 Connection of Outer and Outer Port via PassThroughSwConnector

[constr_1249] Compatibility of ModeSwitchInterfaces in the context of a PassThroughSwConnector ⌈PortPrototypes of different ModeSwitchInterfaces are considered compatible **at the time when the RTE is generated** if and only if

1. For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the required outer PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the provided outer PortPrototype.

Either the shortNames of the ModeDeclarationGroupPrototypes are used to identify the pair **or** a ModeInterfaceMapping exists that maps the corresponding ModeDeclarationGroupPrototypes.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋()

6.6 Compatibility of Mode Declaration Group Prototypes

[constr_1074] Compatibility of ModeDeclarationGroupPrototypes ⌈

ModeDeclarationGroupPrototypes are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. They are typed by (read "refer to") compatible ModeDeclarationGroups.
2. A ModeDeclarationGroupPrototypeMapping exists that identifies the differently named ModeDeclarationGroupPrototypes that correlate with each other. [constr_1210] applies.

⌋()

6.7 Compatibility of Mode Declaration Groups

[constr_1075] Compatibility of ModeDeclarationGroups ⌈ModeDeclarationGroups are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All the following subconditions apply:
   (a) They define an identical number of ModeDeclarations.
   (b) Each ModeDeclaration on the required side corresponds to a ModeDeclaration on the provided side with an identical shortName.
   (c) The initialModes on both sides refer to ModeDeclarations with identical shortNames.
   (d) The attribute ModeDeclarationGroup.modeUserErrorBehavior.errorReactionPolicy has identical values on both sides.
   (e) The attribute ModeDeclarationGroup.modeManagerErrorBehavior.errorReactionPolicy has identical values on both sides.
   (f) The attribute ModeDeclarationGroup.modeUserErrorBehavior.defaultMode either does not exist on both sides or refers on both sides to ModeDeclarations with identical shortNames.

(g) The attribute ModeDeclarationGroup.modeManagerErrorBehavior.defaultMode either does not exist on both sides or refers on both sides to ModeDeclarations with identical shortNames.

(h) one of the following subconditions applies:

- the attribute category has the value ALPHABETIC_ORDER on both sides.
- the attribute category has the value EXPLICIT_ORDER on both sides **and** the matching ModeDeclarations according to 1(b) have the identical values of the attributes ModeDeclaration.value **and** also the value of ModeDeclarationGroup.onTransitionValue matches on both sides.

2. A ModeDeclarationMapping is applied which identifies the corresponding ModeDeclarations.

In addition, the compatibility of corresponding ModeTransitions shall be checked, i.e. [constr_1194] and [constr_1245] apply.⌋*()*

[constr_1245] Consideration of ModeTransitions for the compatibility of ModeDeclarationGroups ⌈One of the following conditions for the consideration of ModeTransitions for the compatibility of ModeDeclarationGroups shall apply **at the time when the RTE is generated**:

- **Either** the mode provider **or** the mode user define ModeTransitions.
- The ModeTransitions defined in the context of the mode provider are **identical** to the ModeTransitions defined in the context of the mode user **or** a ModeDeclarationMapping mapping is applied.

⌋*()*

[constr_1194] Identical ModeTransitions ⌈Two ModeDeclarationGroups contain identical modeTransitions **at the time when the RTE is generated** if and only if

1. For each ModeTransition defined in the context of the mode provider one ModeTransition with the same shortName is defined in the context of the mode user.
2. Each pair of ModeTransitions in both ModeDeclarationGroups identified by their respective shortName have identical targets (in terms of the shortName of the referenced ModeDeclaration) of the references enteredMode and exitedMode.

⌋*()*

6.8 Compatibility of Argument Prototypes

[constr_1076] Compatibility of ArgumentDataPrototypes ⌈Two ArgumentDataPrototypes are compatible **at the time when the RTE is generated** if and only if

1. They are typed by compatible AutosarDataTypes **or** a ClientServerOperationMapping.argumentMapping exists that references one ArgumentDataPrototype in the role firstDataPrototype and the other ArgumentDataPrototype in the role secondDataPrototype.

2. They have the same value of the argument direction (in, out or inout), i.e. [constr_1268] applies.

⌋*()*

6.9 Compatibility of Application Errors

[constr_1077] Compatibility of ApplicationErrors ⌈Two ApplicationErrors are compatible **at the time when the RTE is generated** if and only if one of the following conditions applies:

1. All the following subconditions apply:

   (a) They have the same shortName.

   (b) They have the same attributes. Especially the errorCode shall be identical in both ApplicationErrors.

2. A ClientServerInterfaceMapping.errorMapping exists that references one of the ApplicationErrors in the role firstApplicationError and the other ApplicationErrors in the role secondApplicationError.

⌋*()*

6.10 Compatibility of Client/Server Operations

[constr_1078] Compatibility of ClientServerOperations ⌈Two ClientServerOperations are considered compatible **at the time when the RTE is generated** if their signatures match. In particular, they are compatible if and only if

1. They have the same number of ArgumentDataPrototypes.

2. The n-th arguments of both ClientServerOperations are compatible. This implies ordering of ArgumentDataPrototypes.

3. They have identical values of attribute diagArgIntegrity or the attribute diagArgIntegrity does not exist on both sides.

4. They have the same shortName (again allows for mapping in PortInterfaces).
5. The required ClientServerOperation specifies a compatible ApplicationError for each ApplicationError that is possibly raised by the provided ClientServerOperation, maybe more. Thereby, ClientServerOperations that refer to a possibleError that represents the value E_OK are compatible to ClientServerOperations that do refer to possibleErrors where none of them represents the value E_OK.

⌋()

6.11 Compatibility of Client Server Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a client shall call a certain operation to allow the server to work correctly).

6.11.1 Connection of Required and Provided Port via AssemblySwConnector

[constr_1079] Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector ⌈ClientServerInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:
   (a) For each ClientServerOperation defined in the context of the ClientServerInterface of the required PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.
   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:
      i. It is referenced by the corresponding SwConnector.
      ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.
2. For each such pair, the values of their isService attributes are identical.

⌋()

6.11.2 Connection of Inner and Outer Port via DelegationSwConnector

[constr_1080] Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector ⌈ClientServerInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:

   (a) For each ClientServerOperation defined in the context of the ClientServerInterface of the required inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the required outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.

   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.

2. One of the following conditions applies:

   (a) For at least one ClientServerOperation defined in the context of the ClientServerInterface of the provided inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.

   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.

3. For each such pair, the values of their isService attributes are identical.

⌋*()*

6.11.3 Connection of Outer and Outer Port via PassThroughSwConnector

[constr_1250] Compatibility of ClientServerInterfaces in the context of a PassThroughSwConnector ⌈PortPrototypes of different ClientServerInterfaces are considered compatible **at the time when the RTE is generated** if and only if

1. For **at least one** ClientServerOperation defined in the context of the ClientServerInterface of the provided outer PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the required outer PortPrototype.

   Either the shortNames of the ClientServerOperations are used to identify the pair **or** a ClientServerInterfaceMapping exists that maps the corresponding ClientServerOperations.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋*()*

6.12 Compatibility of Trigger Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a client shall call a certain operation to allow the server to work correctly).

6.12.1 Connection of Required and Provided Port via AssemblySwConnector

[constr_1081] Compatibility of TriggerInterfaces in the context of an AssemblySwConnector ⌈TriggerInterfaces are compatible **at the time when the RTE is generated** if and only if

1. One of the following conditions applies:

   (a) For each Trigger defined in the context of the TriggerInterface of the required PortPrototype a compatible Trigger exists in the TriggerInterface of the provided PortPrototype. The shortNames of Trigger are used to identify the pair.

   (b) A TriggerInterfaceMapping.triggerMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two Triggers in the role firstTrigger and the other in the role secondTrigger.

2. For each such pair, the values of their isService attributes are identical.

⌋*()*

6.12.2 Connection of Inner and Outer Port via DelegationSwConnector

[constr_1082] Compatibility of TriggerInterfaces in the context of an DelegationSwConnector ⌈TriggerInterfaces are compatible **at the time when the RTE is generated** if and only if all the following conditions apply:

1. One of the following subconditions applies:

   (a) For each Trigger defined in the context of the TriggerInterface of the **required** inner PortPrototype a compatible Trigger exists in the TriggerInterface of the **required** outer PortPrototype. The shortNames of Trigger are used to identify the pair.

   (b) For at least one Trigger defined in the context of the TriggerInterface of the **provided** outer PortPrototype a compatible Trigger exists in the TriggerInterface of the **provided** inner PortPrototype. The shortNames of Trigger are used to identify the pair.

   (c) A TriggerInterfaceMapping.triggerMapping exists for which all the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two Triggers in the role firstTrigger and the other in the role secondTrigger.

2. For each such pair, the values of their isService attributes are identical.

⌋*()*

6.12.3 Connection of Outer and Outer Port via PassThroughSwConnector

[constr_1251] Compatibility of PortPrototypes of TriggerInterfaces in the context of a PassThroughSwConnector ⌈PortPrototypes of different TriggerInterfaces are considered compatible **at the time when the RTE is generated** if and only if

1. For **at least one** Trigger defined in the context of the TriggerInterface of the required outer PortPrototype a compatible Trigger exists in the TriggerInterface of the provided outer PortPrototype.

   Either the shortName of Triggers are used to identify the pair **or** a TriggerInterfaceMapping exists that refers to one of the Triggers in the role firstTrigger and to the other in the role secondTrigger.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋*()*

6.13 Compatibility of Trigger

[constr_1083] Compatibility of Triggers ⌈Triggers are compatible **at the time when the RTE is generated** if one of the following conditions is fulfilled:

- They have an identical shortName.
- A TriggerMapping exists that references one of the Triggers in the role firstTrigger and the other Trigger in the role secondTrigger.

⌋*()*

6.14 Entire Delegation of a Provided Port Prototype

[constr_1084] delegation of a provided outer PortPrototype ⌈The delegation of a provided outer PortPrototype is properly defined if the following criteria are fulfilled:

1. For each VariableDataPrototype or ParameterDataPrototype present in the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the provided outer PortPrototype at least one connection via DelegationSwConnector to a provided inner PortPrototype **or** PassThroughSwConnector to a required outer PortPrototype with a compatible VariableDataPrototype or ParameterDataPrototype in the SenderReceiverInterface NvDataInterface or ParameterInterface of the provided inner PortPrototype **or** required outer PortPrototype exists.

   Either the shortNames of VariableDataPrototypes or ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

2. For each VariableDataPrototype provided by a PRPortPrototype that is typed by a SenderReceiverInterface or NvDataInterface and that is referenced in the role outerPort by a DelegationSwConnector a corresponding VariableDataPrototype owned by an innerPort shall be provided by either a PPortPrototype or a PRPortPrototype.

   Either the shortNames of VariableDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

3. For the ModeDeclarationGroupPrototype present in the ModeSwitchInterface of the provided outer PortPrototype exactly one connection via DelegationSwConnector to a provided inner PortPrototype **or** PassThroughSwConnector to a required outer PortPrototype with a compatible ModeDeclarationGroupPrototype in the ModeSwitchInterface

of the provided inner PortPrototype **or** required outer PortPrototype exists.

Either the shortNames of ModeDeclarationGroupPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

4. For each ClientServerOperation present in the ClientServerInterface of the provided outer PortPrototype exactly one connection via DelegationSwConnector to a provided inner PortPrototype **or** PassThroughSwConnector to a required outer PortPrototype with a compatible ClientServerOperation in the ClientServerInterface of the provided inner PortPrototype **or** required outer PortPrototype exists.

   Either the shortNames of ClientServerOperations are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

5. For each Trigger present in the TriggerInterface of the provided outer PortPrototype exactly one connection via DelegationSwConnector to a provided inner PortPrototype **or** PassThroughSwConnector to a required outer PortPrototype with a compatible Trigger in the TriggerInterface of the provided inner PortPrototype **or** required outer PortPrototype exists.

   Either the shortNames of Triggers are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

This constraint is associated with two possible imposition times:

- **at the time when the RTE is generated**
- **at the time when the creation of the CompositionSwComponentType is finished**

⌋*()*

The constraint [constr_1071] defines which PortInterface elements are compatible depending on the kind of PortInterface and the swImplPolicy attributes of the PortInterface elements.

6.14.1 Split and Merge of PortInterface Elements

With the definition of compatibility rules in chapter 6.4, 6.11, and 6.12, it is possible to split and distribute elements of a PortPrototype typed by a PortInterface containing a superset of PortInterface elements to PortPrototypes of type of PortInterfaces containing subsets of PortInterface elements.

Please find examples that explain the usage of splitting and merging in section 6.16.2.

6.15 Compatibility in Case of a Flat ECU Extract

Please note that in the case of a flat ECU extract of software-components specific compatibility rules apply. To some extent, these rules contradict the rules existing for the pure VFB approach (see chapter 6). That is, if the split-and-merge pattern has been applied on the creation of DelegationSwConnectors it might happen that compatibility rules defined in chapter 6 are violated.

However, given that the flattened ECU extract has been created out of a valid CompositionSwComponentType the flattened ECU extract does not become invalid in this case. In other words, the transformation does not create an invalid model out of a valid model.

However, to support this statement it is necessary to define additional compatibility rules that properly cover this case and allow for a successful validation of the flattened ECU extract.

For the flat ECU extract the compatibility of SenderReceiverInterfaces, NvDataInterfaces, and ParameterInterfaces is considered for connecting of PortPrototypes with a DelegationSwConnector.

[constr_1085] Compatibility in the case of a flat ECU extract ⌈PortPrototypes of different SenderReceiverInterfaces, NvDataInterfaces, and ParameterInterfaces are compatible if and only if for at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the RPortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the provided PortPrototype.

The compatibility of PortInterface elements depends on the kind of PortInterface and the swImplPolicy attributes of the PortInterface elements.

Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.⌋*()*

For clarification, [constr_1071] defines which PortInterface elements are compatible depending on the kind of PortInterface and the swImplPolicy attributes of the PortInterface elements.

Please note that in case of the flat ECU extract it might happen that AssemblySwConnectors that connect to a specific RPortPrototype also connect to PPortPrototypes that do not fulfill the compatibility rule specified in 6.4.1.

In particular, the dataElements might correspond to dataElements defined in the scope of different PPortPrototypes. In other words, in the flat ECU extract it is possible to merge dataElements from different providers.

6.16 Compatibility Examples

This section provides some examples that may explain the compatibility of PortPrototypes.

6.16.1 Compatibility on Assembly Level

The rules for compatibility with respect to the connection of dataElements by means of AssemblySwConnectors are perhaps easier to digest than the delegation case but nonetheless it seems appropriate to provide a set of examples that illustrate the compatibility issue.

6.16.1.1 Legal Use

One of the less trivial examples of this kind is the case of sender/receiver n:1 communication. Figure 6.1 sketches a case where both sender software-components provide the full set of dataElements that are required by the RPortPrototype of the receiving software-component.

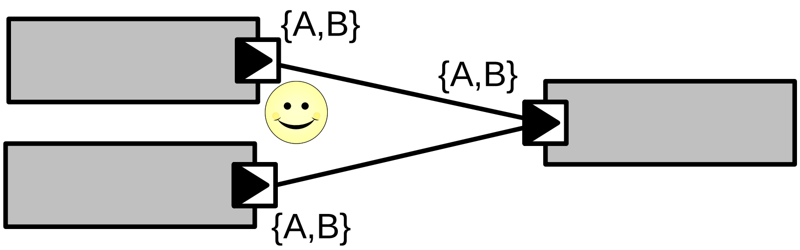


Figure 6.1: legal n:1 communication

The next case (exemplified by Figure 6.2) implements a situation where one sender provides two dataElements {A, B} while the other sender provides only as subset of these, i.e. {B}.

As the RPortPrototype of the receiving software-component requires only the dataElement {B} compatibility issues will not occur because for every required dataElement a compatible dataElement is provided.

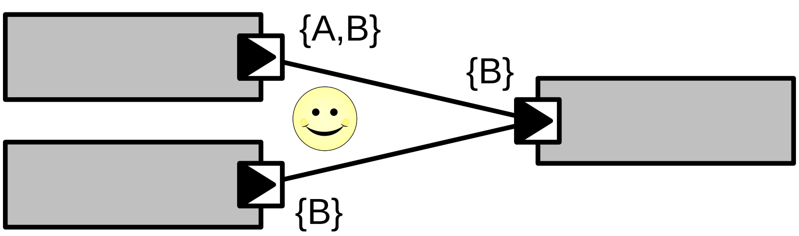


Figure 6.2: legal n:1 communication

6.16.1.2 Illegal Use

On possible example for an illegal configuration of a sender/receiver communication is the scenario sketched in Figure 6.3. Although the sender software-components in total provide the set of required dataElements the *individual* AssemblySwConnectors create incompatible connections between sender and receiver.

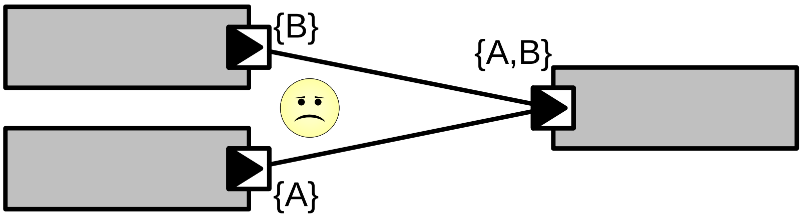


Figure 6.3: illegal n:1 communication

6.16.2 Compatibility on Delegation Level

The rules for compatibility with respect to the delegation of dataElements perhaps require some explanation in terms of examples. The first example 6.4 describes a legal situation where two DelegationSwConnectors split the dataElements contained in the RPortPrototype owned by a CompositionSwComponentType.

6.16.2.1 Legal Use

The examples explain the usage of DelegationSwConnectors in different configurations and different values of DelegatedPortAnnotation. Please note that the DelegatedPortAnnotation is usually defined before the internal structure of a CompositionSwComponentType is fully clarified.

At a later point in time it has to be consistent or can be removed. Decorating the example with applicable values of DelegatedPortAnnotation should facilitate the understanding of the meaning of the DelegatedPortAnnotation.

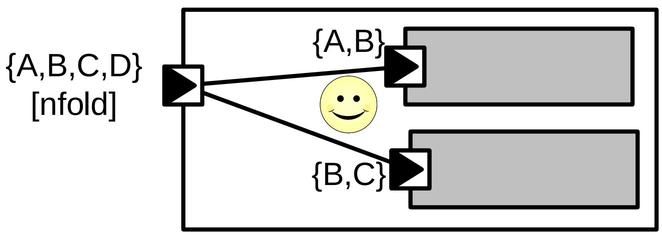


Figure 6.4: Legal split of delegation connector

All required dataElements are provided by the DelegationSwConnectors attached to the delegation RPortPrototype. The fact that dataElement D is not conveyed to any of the RPortPrototypes owned by the SwComponentPrototypes does not have any impact on the compatibility.

In other words: the RPortPrototype at the CompositionSwComponentType actually contains the superset of dataElements {A, B, C, D}. The two required inner PortPrototypes of the SwComponentPrototypes contain the subsets of VariableDataPrototypes {A, B} and {B, C}. In this case the resulting communication pattern on the VFB for B would be 1:n.

This requires the value of the attribute signalFan of DelegatedPortAnnotation to be set to the value nfold.

In the next example, the RPortPrototype of the CompositionSwComponentType contains the superset of dataElements {A, B}. The two RPortPrototypes of the SwComponentPrototypes contain *different* subsets, i.e. {A} and {B}.

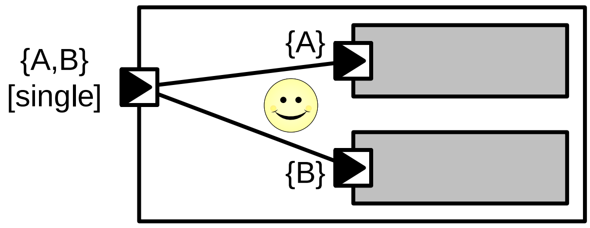


Figure 6.5: Legal split of delegation connector

In this case, the resulting communication pattern on the VFB would be n:1. In this case the value of the attribute signalFan of DelegatedPortAnnotation should be set to single.

The next example is about the merge of DelegationSwConnectors. The PPortPrototype owned by the CompositionSwComponentType contains a superset of dataElements {A, B}. The two PPortPrototypes of the SwComponentPrototypes contain a *disjoint* subset each, i.e. {A} and {B}.

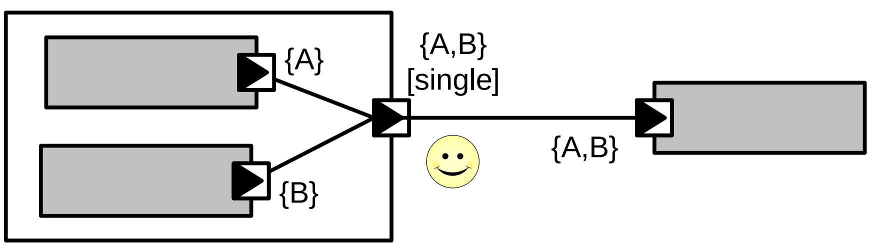


Figure 6.6: Legal merge of delegation connector

In this case, the resulting communication pattern on the VFB would be 1:x, with x taking values between 0 and n. In this case the value of the attribute signalFan of DelegatedPortAnnotation should be set to single. All VariableDataPrototypes of the provided outer PortPrototypes are provided by exactly one provided inner PortPrototype.

As a variation of this theme, the next example features a PPortPrototype owned by a CompositionSwComponentType that contains the superset of dataElements {A, B, C}.

The PPortPrototypes of the SwComponentPrototypes in turn contain subsets of dataElements, i.e. {A, B} and {B, C}. In this case the resulting communication pattern on the VFB for {B} would be n:1.

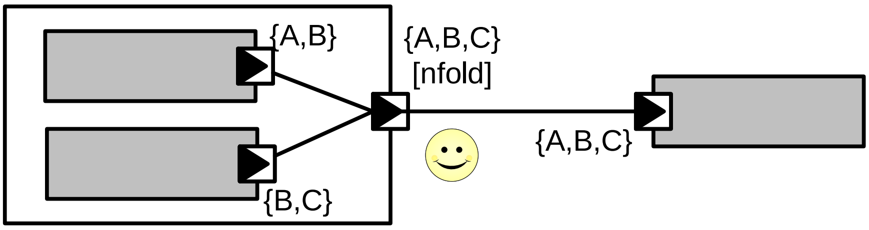


Figure 6.7: Legal merge of delegation connector

This would require the value of the attribute signalFan of DelegatedPortAnnotation to be set to nfold. All dataElements of the delegation PPortPrototype are provided by at least one PPortPrototype of the SwComponentPrototypes. Therefore, the criteria of entire delegation defined in chapter 6.14 are fulfilled.

The next example looks very similar. However, the subtle difference is that the second SwComponentPrototype provides dataElements {C, D} rather than {B, C}.

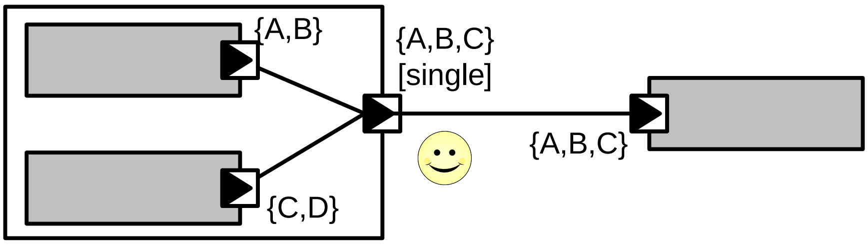


Figure 6.8: Legal merge of delegation connector

Although dataElement {D} does not appear in the delegation PPortPrototype, the compatibility rules are fully satisfied with this scenario.

The next example shows a valid delegation of SwConnectors that goes end-to-end via CompositionSwComponentTypes to included SwComponentPrototypes.

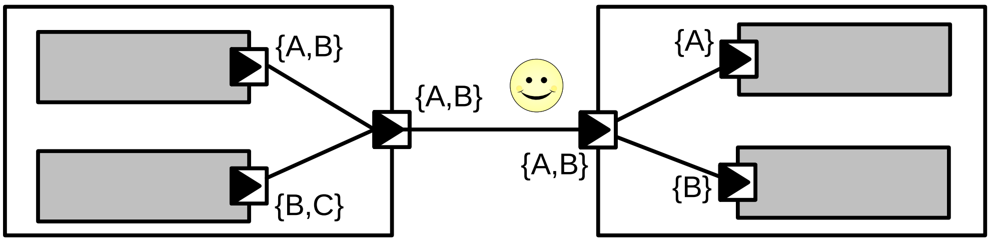


Figure 6.9: Valid delegation of SwConnectors that goes end-to-end

6.16.2.2 Illegal Use

The first example for an illegal use of splitting of dataElements suffers from the fact that not all dataElements owned by the RPortPrototypes of the SwComponentPrototypes are available from the connected RPortPrototypes owned by the CompositionSwComponentType.

Although dataElements the connections in total match ({A} and {B} are connected to a PortPrototype requiring {A, B}) the compatibility rules are not fulfilled because they apply separately for *each* SwConnector

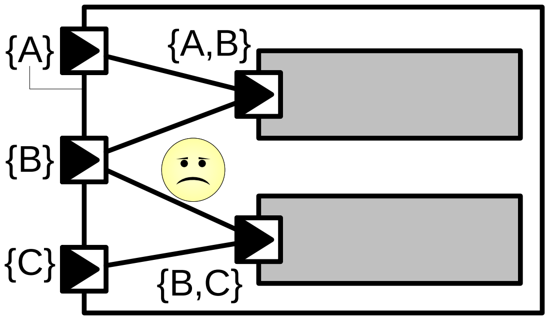


Figure 6.10: Illegal split of delegation connector

In the next example compatibility is also not fulfilled because the required dataElement {E} is not provided by the delegation RPortPrototype.

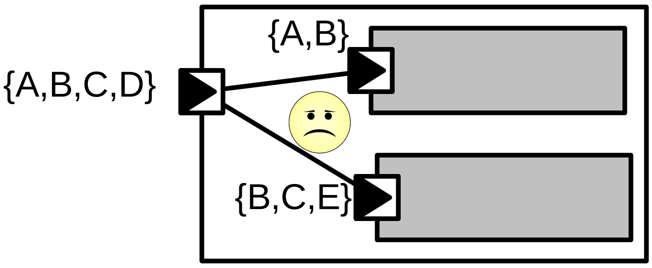


Figure 6.11: Illegal split of delegation connector

An incompatible merge of DelegationSwConnectors is sketched in Figure 6.12. In this case the dataElement {E} is *not* provided by one of the PPortPrototypes owned by the SwComponentPrototypes inside the CompositionSwComponentType.

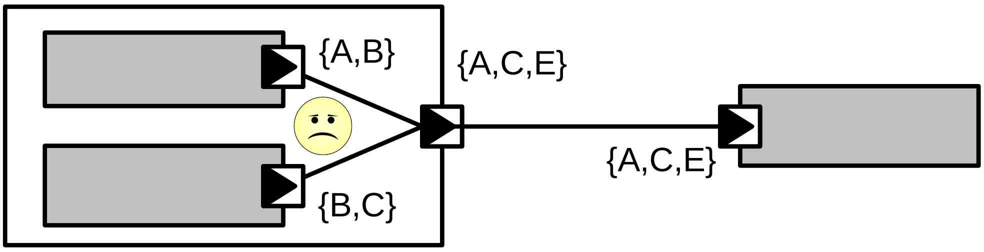


Figure 6.12: Illegal merge of delegation connector

The next example shows an invalid delegation of SwConnectors that goes end-to-end via CompositionSwComponentTypes to included SwComponentPrototypes.

Similar to the example sketched in Figure 6.12, the dataElement {E} is *not* provided by one of the PPortPrototypes owned by the SwComponentPrototypes inside the CompositionSwComponentType.

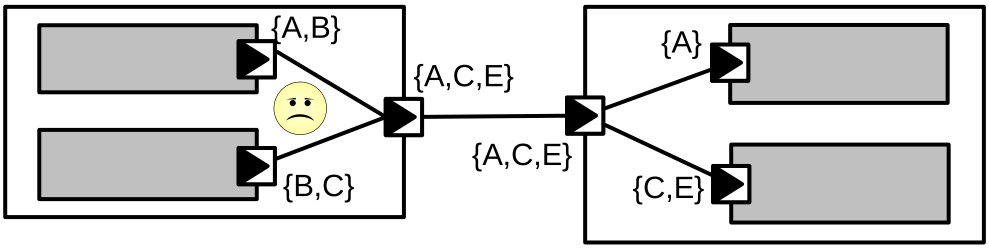


Figure 6.13: Invalid delegation of SwConnectors that goes end-to-end

7 Internal Behavior

7.1 Introduction

[TPS_SWCT_01075] SwcInternalBehavior ⌈SwcInternalBehavior provides means for formally defining the behavior of an AtomicSwComponentType.⌋*(RS_-SWCT_03040)*

This chapter focuses on the description of the SwcInternalBehavior meta-class and the various meta-classes it aggregates. An overview of the meta-class is sketched in Figure 7.3. Please note that SwcInternalBehavior inherits from InternalBehavior.

The role of SwcInternalBehavior in the context of an AUTOSAR software-component is depicted in Figure 7.1. As mentioned in section 3.2, the reason to make the aggregation of SwcInternalBehavior to AtomicSwComponentType ≪atpSplitable≫ is to allow for the development of SwcInternalBehavior in a later process step (e.g. after the VFB view has been completed).

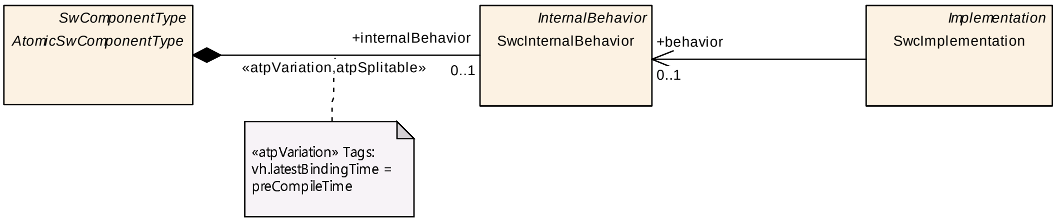


Figure 7.1: The "big picture" of SwcInternalBehavior

Class	***InternalBehavior*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswInternalBehavior, SwcInternalBehavior			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	***InternalBehavior*** (abstract)			
constant Memory	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this Internal Behavior. The shortName of ParameterDataPrototype has to be equal to the "C' identifier of the described constant. The characteristic value(s) might be shared between Sw ComponentPrototypes of the same SwComponentType. The aggregation of constantMemory is subject to variability with the purpose to support variability in the software component or module implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=constantMemory.shortName, constant Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
exclusiveArea	ExclusiveArea	*	aggr	This specifies an ExclusiveArea for this InternalBehavior. The exclusiveArea is local to the component resp. module. The aggregation of ExclusiveAreas is subject to variability. Note: the number of ExclusiveAreas might vary due to the conditional existence of RunnableEntities or BswModuleEntities. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveArea.shortName, exclusive Area.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea NestingOrder	ExclusiveAreaNesting Order	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaNestingOrder.shortName, exclusiveAreaNestingOrder.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**_InternalBehavior_** (abstract)			
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurerment variables implemented by this software component. The term "static" is used in the meaning of "non-temporary" and does not necessarily specify a linker encapsulation. This kind of memory is only supported if supportsMultipleInstantiation is FALSE. The shortName of the VariableDataPrototype has to be equal with the "C' identifier of the described variable. The aggregation of staticMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=staticMemory.shortName, static Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 7.1: InternalBehavior

Class	**SwcInternalBehavior**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	_ARObject_, _AtpClassifier_, _AtpFeature_, _AtpStructureElement_, _Identifiable_, _InternalBehavior_, _Multilanguage Referrable_, _Referrable_			
Aggregated by	_AtomicSwComponentType_.internalBehavior, _AtpClassifier_.atpFeature			
Attribute	**_Type_**	**_Mult._**	**_Kind_**	**_Note_**
arTypedPer Instance Memory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is set to "true" or if the component defines NVRAM access via permanent blocks. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory.shortName, ar TypedPerInstanceMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
event	RTEEvent	*	aggr	This is a RTEEvent specified for the particular Swc InternalBehavior. The aggregation of RTEEvent is subject to variability with the purpose to support the conditional existence of RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using Data ReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.short Label vh.latestBindingTime=preCompileTime
exclusiveArea Policy	SwcExclusiveArea Policy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveArea Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
explicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=explicitInterRunnableVariable.shortName, explicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
handle TerminationAnd Restart	HandleTerminationAnd RestartEnum	0..1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSw ComponentType may either not support stop and restart, or support only stop, or support both stop and restart.
implicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implicitInterRunnableVariable.shortName, implicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedDataTypeSet

▽

△

Class	**SwcInternalBehavior**			
includedMode Declaration GroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedModeDeclarationGroupSet
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of Port Prototypes and component local memories like "per InstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Memory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstance Memory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceMemory.shortName, perInstance Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is set to "true". The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceParameter.shortName, per InstanceParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portAPIOption, portAPIOption.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
runnable	RunnableEntity	*	aggr	This is a RunnableEntity specified for the particular Swc InternalBehavior. The aggregation of RunnableEntity is subject to variability with the purpose to support the conditional existence of RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of Port Prototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnable.shortName, runnable.variation Point.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	SwcService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternal Behavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency.shortName, service Dependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
shared Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same Sw ComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sharedParameter.shortName, shared Parameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
supports Multiple Instantiation	Boolean	0..1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=variationPointProxy.shortName

Table 7.2: SwcInternalBehavior

[constr_1934] Existence of attribute `SwcInternalBehavior.handleTerminationAndRestart` ⌈For each `SwcInternalBehavior`, attribute `handleTerminationAndRestart` shall exist **at the time when the RTE is generated.**⌋*()*

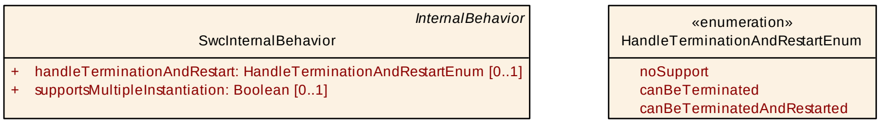


Figure 7.2: Termination and restart

[constr_1935] Existence of attribute SwcInternalBehavior.supportsMultipleInstantiation ⌈For each SwcInternalBehavior, attribute supportsMultipleInstantiation shall exist **at the time when the contract phase generation is executed**.⌋*()*

Enumeration	**HandleTerminationAndRestartEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior
Note	Controls the behavior of an AtomicSwComponentType with respect to stop and restart.
Aggregated by	SwcInternalBehavior.handleTerminationAndRestart
Literal	***Description***
canBeTerminated	Supports termination. **Tags:**atp.EnumerationLiteralIndex=0
canBeTerminatedAndRestarted	Supports termination and restarting. **Tags:**atp.EnumerationLiteralIndex=1
noSupport	Stop and restart is not supported at all. **Tags:**atp.EnumerationLiteralIndex=2

Table 7.3: HandleTerminationAndRestartEnum

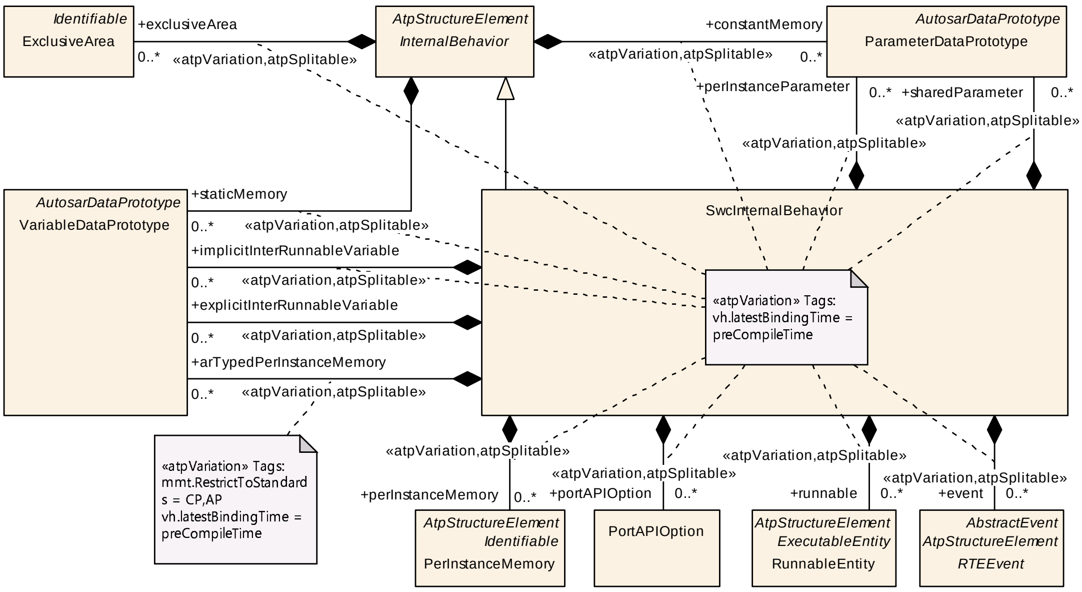


Figure 7.3: SwcInternalBehavior

7.2 Runnable Entity

The concept of RunnableEntity (more details can be found in Figure 7.4) is defined in the specification of the Virtual Function Bus [3].

[TPS_SWCT_01030] RunnableEntity ⌈RunnableEntitys are the smallest code-fragments that are provided by a software-component and are (at least indirectly) a subject for scheduling by the underlying operating system or else (in rare cases) for execution in ISR context.⌋*(RS_SWCT_03040, RS_SWCT_00070, RS_SWCT_00090, RS_SWCT_03050)*

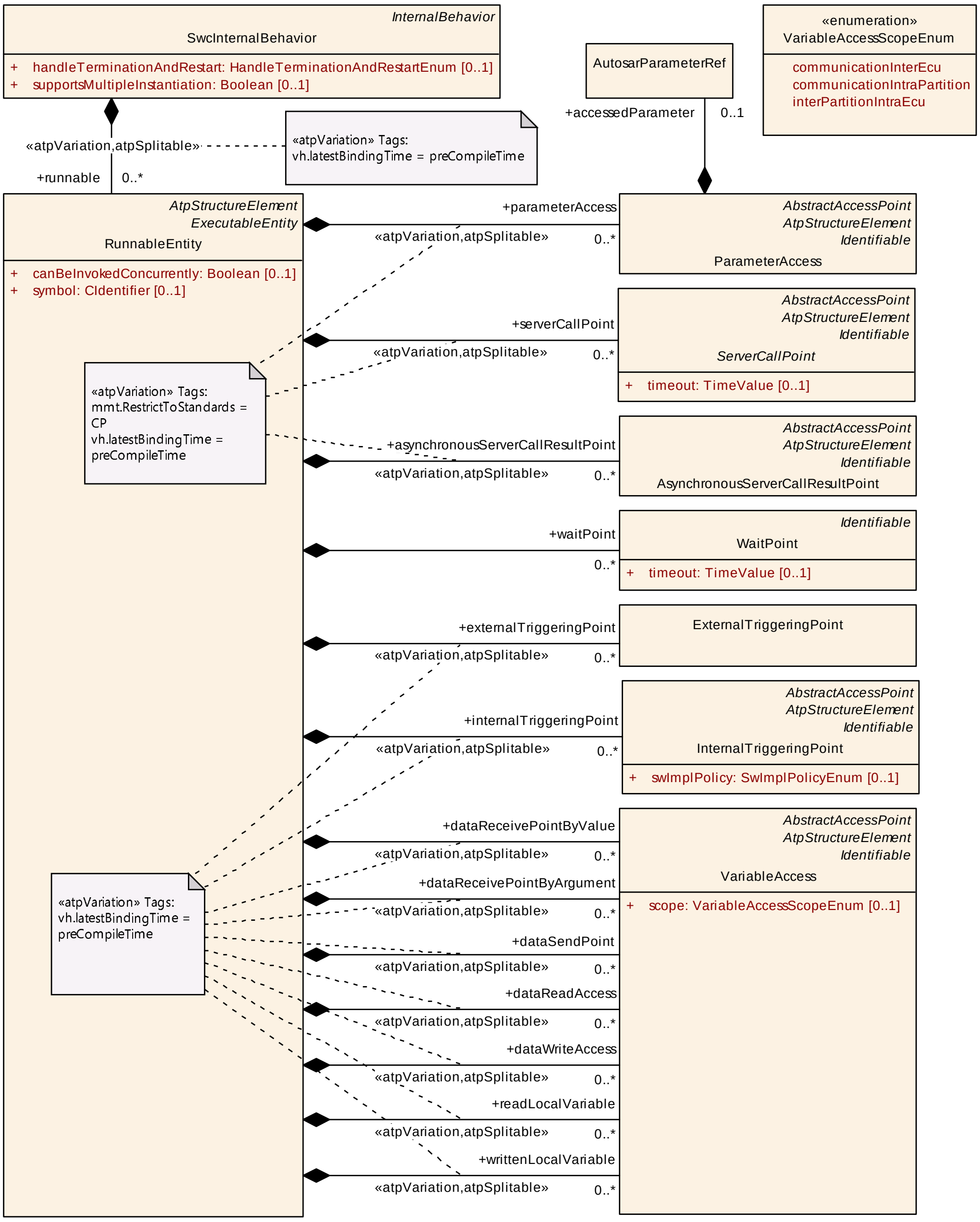


Figure 7.4: Details of RunnableEntity

[TPS_SWCT_01097] CompositionSwComponentType cannot have RunnableEntitys ⌈It is intentionally not possible for CompositionSwComponentType to define a SwcInternalBehavior. Consequently, CompositionSwComponentTypes don't have RunnableEntitys by themselves.⌋*(RS_SWCT_00070, RS_SWCT_00090, RS_SWCT_03050)*

[TPS_SWCT_01098] Only AtomicSwComponentType can have RunnableEntitys ⌈Only the AtomicSwComponentType that are populating a CompositionSwComponentType as SwComponentPrototypes may have RunnableEntitys.⌋*(RS_SWCT_03040, RS_SWCT_00070, RS_SWCT_00090, RS_SWCT_03050)*

This correlation is depicted in Figure 7.5.

Please note that RunnableEntitys exist in several categories that have different properties. Please find more explanation about categories of RunnableEntitys in section 7.2.4.4.

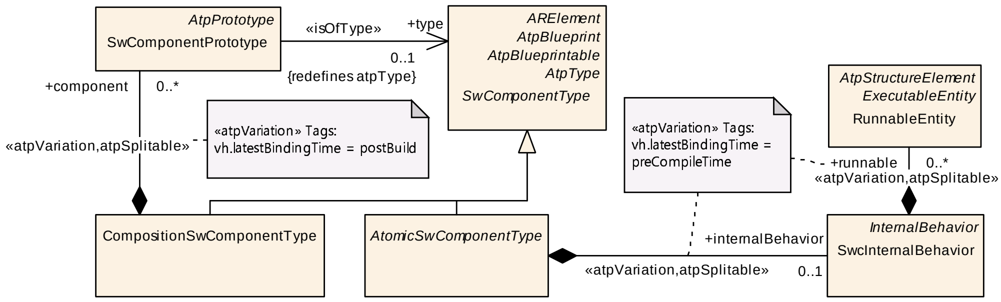


Figure 7.5: Only AtomicSwComponentTypes may have RunnableEntitys

[TPS_SWCT_01302] Semantics of minimumStartInterval ⌈The attribute ExecutableEntity.minimumStartInterval defines the time interval that the RTE will guarantee to not go below between scheduling two consecutive executions of the corresponding RunnableEntity.⌋*(RS_SWCT_03040)*

[constr_1936] Existence of attribute RunnableEntity.symbol ⌈For each RunnableEntity, attribute symbol shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01303] symbol attribute describes the RunnableEntity's entry point ⌈The RunnableEntity.symbol attribute is describing the RunnableEntity's entry point.⌋*(RS_SWCT_03040)*

The implication RunnableEntity.symbol on the uniqueness of symbols in the scope of one EcuInstance is described in [constr_2025] [10].

A RunnableEntity inherits several attributes from its base class ExecutableEntity due to the fact that these are also used in the Basic Software Module Description Template [6]. Here the following constraint applies:

[constr_4082] RunnableEntity.reentrancyLevel shall not be set. ⌈The optional attribute reentrancyLevel shall not be set for a RunnableEntity. This attribute would define more specific reentrancy features than the mandatory attribute canBeInvokedConcurrently. These features are currently only supported for Basic Software.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

Class	RunnableEntity			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponentType and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.runnable			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
argument (ordered)	RunnableEntityArgument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.
asynchronousServerCallResultPoint	AsynchronousServerCallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
canBeInvokedConcurrently	Boolean	0..1	attr	If the value of this attribute is set to "true" the enclosing RunnableEntity can be invoked concurrently (even for one instance of the corresponding AtomicSwComponentType). This implies that it is the responsibility of the implementation of the RunnableEntity to take care of this form of concurrency.
dataReadAccess	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
dataReceivePointByArgument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceivePointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointByValue is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue.shortName, dataReceivePointByValue.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataSendPoint.shortName, dataSendPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWriteAccess	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataWriteAccess.shortName, dataWriteAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
externalTriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint.ident.shortName, externalTriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internalTriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint.shortName, internalTriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeAccessPoint	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeAccessPoint.ident.shortName, modeAccessPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeSwitchPoint	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSwitchPoint.shortName, modeSwitchPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameterAccess	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a ParameterDataPrototype which may either be local or within a PortPrototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of ParameterAccess (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterAccess.shortName, parameterAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=readLocalVariable.shortName, readLocal Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serverCallPoint.shortName, serverCall Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbol	CIdentifier	0..1	attr	The symbol describing this RunnableEntity's entry point. This is considered the API of the RunnableEntity and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The WaitPoint associated with the RunnableEntity.
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=writtenLocalVariable.shortName, written LocalVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 7.4: RunnableEntity

Please note that the formal definition of the semantics of a `RunnableEntity` has strong relations to the specification of the AUTOSAR RTE [2]. The definition of the RTE semantics, however, is not in the scope of this document.

However, the formal definition requires some background discussion that can't be completely left out of this document. Otherwise, the meaning of specific model elements could not be understood properly.

Please note further that there are legitimate use cases for software-components without any `RunnableEntity`s, e.g. in following situations:

- An NvBlockSwComponentType does not require any RunnableEntity if there is no need to proxy any PortPrototype typed by either of the ClientServerInterfaces NvMService or NvMAdmin.
- A ServiceSwComponentType runs in a reduced configuration and does not have to offer any PortPrototype to any service-using application software-component.
- A software-component is configured in a reduced configuration where none of the functionality is selected. In this case, it's simpler to keep the empty software-component instead of adding further VariationPoints at many other elements, e.g. SwComponentPrototype.

  On top of that, a variation-based approach would require the conditional existence of other ARElements which are not yet supported, e.g. a SwcImplementation that references the AtomicSwComponentType.

7.2.1 Concurrency and Reentrancy of a RunnableEntity that cannot be Invoked Concurrently

This section applies to the case that the value of the attribute canBeInvokedConcurrently is false. During runtime, each RunnableEntity of each instance of an AtomicSwComponentType is in a specific run-time state.

The details of the definition and semantics of run-time states can be found in [2]. Nevertheless, this chapter contains a brief description of the fundamental concepts in order to properly being able to discuss the formal modeling of RunnableEntitys.

[TPS_SWCT_01313] Conditions for a transition from suspended to to be started ⌈The SwcInternalBehavior describes for each RunnableEntity the conditions for a transition from suspended to to be started should occur. This is done using the concept of an RTEEvent.⌋*(RS_SWCT_03040)*

When a RunnableEntity is in state to be started, the RTE can decide to start running the RunnableEntity. The delay between entering the state to be started (e.g. a message has been received in response to which the RunnableEntity should run) and moving into the state running (the first instruction of the RunnableEntity has been executed) depends on the scheduling strategy of the RTE, i.e. the mapping of RunnableEntitys on AUTOSAR OS tasks.

The transition from the state running into the state suspended is in the hands of the RunnableEntity: the transition occurs when the RunnableEntity returns (thereby handing over control to the AUTOSAR OS [33]). Some RunnableEntitys (like cat. 2 RunnableEntitys) might never return to the suspended state once they entered the running state.

They might enter the preempted state when being preempted. The same applies if a RunnableEntity needs to wait for a WaitPoint to be unblocked.

[TPS_SWCT_01304] Cat. 1A and 1B RunnableEntitys will eventually terminate ⌈Cat. 1A and 1B RunnableEntitys will eventually return after having executed a specific finite algorithm (the execution time of which might be provided).⌋*(RS_SWCT_-03040)*

[TPS_SWCT_01305] RunnableEntity as one that cannot be invoked concurrently ⌈In case the SwcInternalBehavior defines a RunnableEntity as one that cannot be invoked concurrently it is the responsibility of the RTE to make sure that the RunnableEntity is never started concurrently (for example, in two different AUTOSAR OS tasks). This implies that the implementation of the AtomicSwComponentType does not need to worry about concurrency issues.⌋*(RS_SWCT_03040)*

For example: The internal behavior of an AtomicSwComponentType *MyComponentType* describes a RunnableEntity *R1* which should be enabled when an operation on a client-server PPortPrototype of the AtomicSwComponentType is invoked. The AtomicSwComponentType specifies that the RunnableEntity *R1* cannot be invoked concurrently.

The AtomicSwComponentType *MyComponentType* is instantiated on an ECU. When a call of the operation is received, the corresponding instance of the RunnableEntity *R1* is enabled and the RTE will start executing the RunnableEntity (the RunnableEntity is in state running) in a task eventually managed by the AUTOSAR OS.

If another call of the operation is received while the RunnableEntity is in state running, it is not allowed that the RTE runs the RunnableEntity again in a second task. Rather, the RTE has to wait (and maybe queue the second incoming request) until the RunnableEntity has returned and has moved to the suspended state.

7.2.2 Concurrency and Reentrancy of a RunnableEntity that can be Invoked Concurrently

This section applies to the case that the value of the attribute canBeInvokedConcurrently is set to true.

In this case, it is allowed that the same RunnableEntity is running several times concurrently in different AUTOSAR OS tasks. This implies that the state machine defined in [2] is not the state of the RunnableEntity anymore, but can be cloned an arbitrary number of times.

[TPS_SWCT_01306] Software-component description itself does not put any bounds on the number of concurrent invocations of a RunnableEntity ⌈The software-component description itself does not put any bounds on the number of concurrent invocations of the RunnableEntity that are allowed.

The software-component description only specifies whether the RunnableEntity can be invoked concurrently or not.

Allowing concurrent invocation of a RunnableEntity implies that the implementation of the AtomicSwComponentType needs to take care of this additional form of concurrency.⌋*(RS_SWCT_03040)*

For example: The SwcInternalBehavior of a component-type *MyComponentType* describes a RunnableEntity *R1* which should be enabled when a ClientServerOperation on a PPortPrototype typed by a ClientServerInterface of the AtomicSwComponentType is invoked.

The AtomicSwComponentType specifies that the RunnableEntity *R1* can be invoked concurrently. The AtomicSwComponentType *MyComponentType* is instantiated on an ECU.

When a call of the ClientServerOperation is received the corresponding instance of the RunnableEntity *R1* is enabled and the RTE will start executing the RunnableEntity (the RunnableEntity is in state running) in a task eventually managed by the AUTOSAR OS.

If another call of the ClientServerOperation is received, it is allowed that the same RunnableEntity is started again in a different task.

A typical use-case of concurrent RunnableEntitys is the implementation of AUTOSAR services. The AUTOSAR services will typically take care of concurrency internally: several software-components can directly use the services in parallel.

The ECU-integrator could then decide that the RunnableEntity implementing the AUTOSAR service runs directly in the context (in the task) of the AtomicSwComponentType invoking the service.

This is a very efficient and direct coupling between the client and the server: the connector between the client and the server is reduced to a local function-call.

7.2.3 Timed Activation of Runnable Entities

In many cases, RunnableEntitys need to be activated in response to timing events rather than related to communication (e.g. the reception of a response to an asynchronous operation invocation). Many RunnableEntitys will need to run cyclically with a fixed rate.

The approach taken in the software-component description is to define so-called TimingEvents (please find more details in Figure 7.6) as special kinds of RTEEvents. So far, only one kind of timing-related RTEEvent has been defined: a simple periodic TimingEvent.

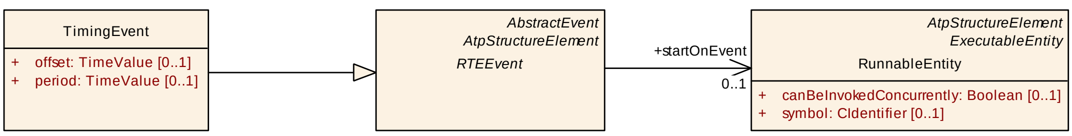


Figure 7.6: Periodic activation of RunnableEntities

[TPS_SWCT_01519] RTE executes certain RunnableEntity periodically ⌈If the SwcInternalBehavior of an AtomicSwComponentType requires that the RTE executes certain RunnableEntitys periodically, the description needs to define a TimingEvent with the desired period.

This TimingEvent then contains a reference to the RunnableEntity that needs to be executed with this period.⌋*(RS_SWCT_03040)*

[constr_2031] Value of TimingEvent.period shall be greater than 0 ⌈Attribute TimingEvent.period shall exist and its value shall be greater than 0 **at the time when the RTE is generated**.⌋*()*

Note that it is possible to override the attribute period on the level of instantiation. See [TPS_SWCT_02507] for more details.

Class	**TimingEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is used to start RunnableEntities that shall be executed periodically.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
offset	TimeValue	0..1	attr	The value makes an assumption about the time offset of the first activation of the RunnableEntity triggered by the mapped TimingEvent relative to the periodic activation of the time base of this TimingEvent. Unit: second.
period	TimeValue	0..1	attr	Period of timing event in seconds. The value of this attribute shall be greater than zero.

Table 7.5: TimingEvent

[constr_1622] Value of TimingEvent.offset vs. TimingEvent.period ⌈If a value is defined for attribute TimingEvent.offset then this value shall be greater than 0 and less or equal than the value of attribute TimingEvent.period of the respective TimingEvent **at the time when the RTE is generated**.⌋*()*

The motivation for the existence of [constr_1622] is that the mapped TimingEvent could not be implemented with the given period if the activation offset is greater than the period of the TimingEvent.

7.2.4 Additional Remarks and Clarifications

7.2.4.1 Reentrancy and Multiple Instantiation

This chapter is emphasizing on the specific meanings of combinations of the attributes SwcInternalBehavior.supportsMultipleInstantiation and RunnableEntity.canBeInvokedConcurrently.

[TPS_SWCT_01307] supportsMultipleInstantiation vs. canBeInvokedConcurrently ⌈The semantics of combining the attributes supportsMultipleInstantiation and canBeInvokedConcurrently is summarized in Table 7.6.⌋ *(RS_SWCT_03040)*

In case the implementation of a AtomicSwComponentType decides to map several RunnableEntitys to the same symbol there are reentrancy problems to be sorted out. However, this scenario is not supported by RTE [2] anyway and shall therefore be avoided.

supportsMultipleInstantiation	*canBeInvokedConcurrently*	*Implication for an implementation of a RunnableEntity*
false	false	This implies that the implementation of the RunnableEntity will never be invoked concurrently from several tasks. The implementation does not need to care about reentrancy issues and can typically use static variables to store state.
true	false	In case there are several instances of the same AtomicSwComponentType on the local ECU, the implementation of the RunnableEntity can still be invoked concurrently from several tasks. However, there will be no concurrent invocations of the implementation with the same instance handle. To ensure that this is safe, the implementation will typically use per-instance memory.
true	true	In this case the RunnableEntity can be invoked concurrently from several tasks, even with the same instance handle.

Table 7.6: supportsMultipleInstantiation vs. canBeInvokedConcurrently

7.2.4.2 Reentrancy and "Library Functions"

Note that all code that is called by different RunnableEntitys (like e.g. library routines, etc.) shall obviously be reentrant. A filter algorithm implemented in C, for example, is not allowed to store values from previous runs by means of static variables or variables with external binding.

7.2.4.3 Compatibility of ClientServerOperations triggering the same RunnableEntity

[TPS_SWCT_01309] signature of a RunnableEntity depends on the connected RTEEvent ⌈The signature of a RunnableEntity depends on the connected RTEEvent.

Multiple OperationInvokedEvents that trigger the same RunnableEntity are only supported if all referenced ClientServerOperations would result in the same RunnableEntity signature for the server RunnableEntity.⌋ *(RS_SWCT_03040)*

[constr_2000] Compatibility of ClientServerOperations triggering the same RunnableEntity ⌈The ClientServerOperations are considered compatible **at the time when the contract phase generation is executed** if

- the number of arguments (which can be ArgumentDataPrototypes or related PortDefinedArgumentValues) is equal and
- the corresponding arguments (i.e. first argument on both sides, second argument on both sides, etc.) are compatible or both are typed by "new-world" Variable-Size Array Data Types where the data types of the array elements are compatible (but the array sizes may differ).
- and the respective values of PortAPIOption.errorHandling are identical.

In particular, this means that:

- for combinations of ArgumentDataPrototypes and ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

  In case of data types of category STRUCTURE all by order matching ImplementationDataTypeElements shall be named equally.

- for combinations of PortDefinedArgumentValues and ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

  In case of ImplementationDataTypeElements of category STRUCTURE all by order matching ImplementationDataTypeElements of the structure shall be named equally.

- for ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useVoid an arbitrary ImplementationDataType is referred to.

In addition, it is required that the **return value defined on both sides shall match** (in terms of Std_ReturnType vs. void) and also the possibleErrors are compatible.⌋*()*

[TPS_SWCT_01520] Implication of the existence of possibleError on compatibility of ClientServerOperations ⌈An implication of [constr_2000] is that a ClientServerOperation that defines **any** possibleError is **not** compatible with a ClientServerOperation that defines **no** possibleError at all because this configuration leads to different data type of the return value of the C function that implements the applicable RunnableEntity.⌋*(RS_SWCT_03040)*

7.2.4.4 Categories of Runnable Entities

[TPS_SWCT_01310] Categories of RunnableEntitys ⌈RunnableEntitys are subdivided into the following categories:

Category 1 RunnableEntitys of Category 1 do not have WaitPoints and are required to terminate in a finite amount of time. Category 1 is divided into two subcategories: Category 1A and Category 1B.

Category 1A RunnableEntitys are only allowed to use implicit APIs.

Category 1B RunnableEntitys are additionally allowed to invoke a server, to use explicit APIs, to issue triggers, to switch modes and to use ExclusiveAreas.

Category 2 In contrast to Category 1, RunnableEntitys, RunnableEntitys of Category 2 always aggregate at least one WaitPoint[1]. Typically, such a RunnableEntity implements an internal loop where one iteration through the loop is triggered whenever a WaitPoint is resolved.

⌋*(RS_SWCT_03040)*

For more details regarding details of the modeling of meta-class RunnableEntity, please refer to Figure 7.4.

The resource need of a RunnableEntity in its later integration usually depends on the used features of the Runtime Environment.

In the AUTOSAR Methodology, the ECU integrator is required to map various RunnableEntitys to a limited amount of OS Tasks in a specific order.

The RunnableEntity categories are useful to indicate the later integration effort and resource need on the basis of the RunnableEntity's design.

Thereby Category 1A RunnableEntitys do not utilize RTE features which are blocking or delaying the execution of the RunnableEntity. As long as the RunnableEntity implementation guaranties stable execution times it's rather simple and reliable to integrate them in a calculation chain.

As opposed to Category 1A, the scheduling behavior of Category 2 RunnableEntitys at runtime depends on the interaction with the interfaces of the enclosing software component.

For instance, the suspend-times of the OS Task, where the RunnableEntity is mapped to, may depend on actual data reception and/or occurrence of timeouts.

Moreover, it's usually not possible to map more than one RunnableEntity to an OS Task when the RunnableEntity implements an infinite internal loop, triggered whenever a WaitPoint is resolved.

In case of Category 1B, additional side conditions impact the schedule behavior and required OS features. For instance, a server call might be simply implemented as direct function call in case of intra partition communication **or** might require a more complex implementation in case of inter-ECU communication.

[1] Category 2 RunnableEntitys usually have to be mapped to *Extended Tasks*, because only extended tasks provide the task state WAITING.

7.2.4.5 Arguments of a Runnable Entity

In many cases an RTE generator will be able to figure out not only the number and data type of arguments to a RunnableEntity but also the name of the arguments. In some cases, however, formal support from the upstream templates is required to facilitate this task.

[TPS_SWCT_01311] Name of an operation argument ⌈This support is available by means of the meta-class RunnableEntityArgument that contributes the name of the argument by means of the value of the attribute symbol.

As a RunnableEntity might need to define many arguments the aggregation of RunnableEntityArgument at RunnableEntity in the role argument has the multiplicity 0..* and as the order of these arguments is significant the meta-model defines the aggregation as ordered[2].⌋ *()*

[constr_1164] Number of arguments owned by a RunnableEntity ⌈If a given RunnableEntity owns RunnableEntityArguments in the role argument, then the number of these RunnableEntityArguments shall be identical to the number of applicable portArgValues of the PortAPIOption that references the PortPrototype that in turn is referenced by the OperationInvokedEvent that references the RunnableEntity **plus** the number of ArgumentDataPrototypes aggregated in the role argument by the ClientServerOperation referenced by said OperationInvokedEvent **at the time when the contract phase generation is executed**.⌋ *()*

[constr_1165] Applicability of RunnableEntityArgument ⌈The existence of a RunnableEntityArgument is limited to RunnableEntitys triggered by a ClientServerOperation.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋ *()*

[TPS_SWCT_01312] RunnableEntity has a mapping to BswModuleEntry ⌈The existence of RunnableEntityArguments in the role argument owned by a RunnableEntity shall be **ignored** by an RTE generator if a mapping to a BswModuleEntry exists.

In this case the name of arguments to the RunnableEntity shall be derived from the applicable SwServiceArgs owned by the mapped BswModuleEntry.⌋ *(RS_SWCT_03040)*

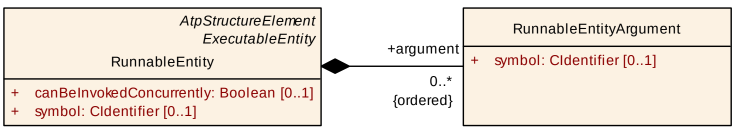


Figure 7.7: Arguments of a RunnableEntity

[2]as the arguments are **ordered** they do not need to be Referrable in order to be able to identify individual arguments

Class	**RunnableEntityArgument**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RunnableEntity			
Note	This meta-class represents the ability to provide specific information regarding the arguments to a RunnableEntity.			
Base	*ARObject*			
Aggregated by	RunnableEntity.argument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
symbol	CIdentifier	0..1	attr	This represents the symbol to be generated into the actual signature on the level of the C programming language.

Table 7.7: RunnableEntityArgument

[constr_1938] Existence of attribute RunnableEntityArgument.symbol ⌈For each RunnableEntityArgument, attribute symbol shall exist **at the time when the contract phase generation is executed**.⌋*()*

7.2.5 Activation Reason of a Runnable Entity

It is feasible to activate a given RunnableEntity by means of several RTEEvents. In many cases, it is therefore necessary to retrieve the information about the activating RTEEvent from within the implementation of the RunnableEntity.

As a typical use case, consider a RunnableEntity that is cyclically activated (by means of a TimingEvent) and in addition it shall also be executed sporadically, e.g. in response to the reception (DataReceivedEvent) of a dataElement.

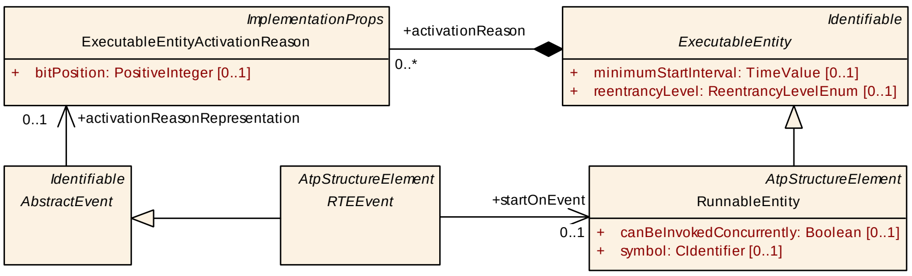


Figure 7.8: ExecutableEntityActivationReason and RunnableEntity

[TPS_SWCT_01469] RTE API for retrieving the current activation reason ⌈The aggregation of a ExecutableEntityActivationReason allows for the RTE generator to create an RTE API for retrieving the current activation reason.⌋*(RS_SWCT_03040, RS_SWCT_03045)*

For details about the implementation of this feature, please refer to the specification of the RTE [2]

[constr_1226] Applicable range for ExecutableEntityActivationReason.bitPosition ⌈The value of attribute ExecutableEntityActivationReason.bitPosition shall be in the range of 0 .. 31 **at the time when the contract phase generation is executed**.⌋*()*

[constr_1227] Value of attribute ExecutableEntityActivationReason.bitPosition shall be unique ⌈The value of attributes ExecutableEntityActivationReason.bitPosition and ExecutableEntityActivationReason.symbol shall be unique in the context of the enclosing RunnableEntity **at the time when the contract phase generation is executed.**⌋*()*

[constr_1228] RTEEvent that is referenced by a WaitPoint in the role trigger shall not reference ExecutableEntityActivationReason ⌈An RTEEvent that is referenced by a WaitPoint in the role trigger shall not reference ExecutableEntityActivationReason in the role activationReasonRepresentation **at the time when the RTE is generated.**⌋*()*

The rationale for the existence of [constr_1228] is obviously that in the described situation the RunnableEntity is already activated and therefore the mentioned RTEEvent does not deliver any information related to the activation reason of said RunnableEntity.

Class	***ExecutableEntity*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Abstraction of executable code.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*BswModuleEntity*, RunnableEntity			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
activationReason	ExecutableEntityActivationReason	*	aggr	If the ExecutableEntity provides at least one activation Reason element the RTE resp. BSW Scheduler shall provide means to read the activation vector of this executable entity execution. If no activationReason element is provided the feature of being able to determine the activating RTEEvent is disabled for this ExecutableEntity.
canEnter	ExclusiveArea	*	ref	This means that the executable entity can enter/leave the referenced exclusive area through explicit API calls. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=canEnter.exclusiveArea, canEnter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveAreaNestingOrder	ExclusiveAreaNestingOrder	*	ref	This represents the set of ExclusiveAreaNestingOrders recognized by this ExecutableEntity.
minimumStartInterval	TimeValue	0..1	attr	Specifies the time in seconds by which two consecutive starts of an ExecutableEntity are guaranteed to be separated.
reentrancyLevel	ReentrancyLevelEnum	0..1	attr	The reentrancy level of this ExecutableEntity. See the documentation of the enumeration type ReentrancyLevelEnum for details. Please note that nonReentrant interfaces can have also reentrant or multicoreReentrant implementations, and reentrant interfaces can also have multicoreReentrant implementations.

▽

△

Class	*ExecutableEntity* (abstract)			
runsInside	ExclusiveArea	*	ref	The executable entity runs completely inside the referenced exclusive area. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runsInside.exclusiveArea, runsInside.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this code entity. Via an association to the same SwAddrMethod, it can be specified that several code entities (even of different modules or components) shall be located in the same memory without already specifying the memory section itself.

Table 7.8: ExecutableEntity

Class	ExecutableEntityActivationReason			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the ability to define the reason for the activation of the enclosing Executable Entity.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Aggregated by	*ExecutableEntity*.activationReason			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bitPosition	PositiveInteger	0..1	attr	This attribute allows for defining the position of the enclosing ExecutableEntityActivationReason in the activation vector.

Table 7.9: ExecutableEntityActivationReason

[constr_1939] Existence of attribute `ExecutableEntityActivationReason.bitPosition` ⌈For each `ExecutableEntityActivationReason`, attribute `bitPosition` shall exist **`at the time when the contract phase generation is executed`**.⌋*()*

Please note that the attribute `ExecutableEntityActivationReason.symbol` is needed for the generation of a unique identifier that represents the specific activation reason in the RTE code.

7.2.6 Runnable Entity for Initialization Purpose

One way to make sure that certain initializations are applied before a software-component enters its state of normal operation is to use the AUTOSAR mode-management, in particular by defining a `ModeDeclarationGroup` that contains a specific `ModeDeclaration` with the semantics of representing a mode that is exclusively used for setting up and initializing a software-component.

However, this approach comes with a certain amount of footprint that may be acceptable in some cases but there may also be cases where a simpler approach comes in handy. The simple approach to initialization consists of a RunnableEntity that is triggered by a special kind of RTEEvent, i.e. the so-called InitEvent.

[TPS_SWCT_01525] InitEvent references a RunnableEntity in the role startOnEvent ⌈In addition to using a mode-based approach for executing initialization RunnableEntitys it is also possible to let an InitEvent reference a RunnableEntity in the role startOnEvent.

This approach to the initialization of software-components is orthogonal to the mode-based approach. Especially, the RunnableEntitys triggered by an InitEvent are expected to be executed after the RTE has been fully initialized. This means restrictions regarding the availability of RTE APIs during the ECU initialization are not relevant for RunnableEntitys triggered by an InitEvent.⌋*(RS_SWCT_03290)*

[constr_1257] No WaitPoints allowed ⌈A RunnableEntity referenced by an InitEvent in the role startOnEvent shall not aggregate a WaitPoint **at the time when the RTE is generated**.⌋*()*

Rationale: a WaitPoint may indefinitely defer the completion of the RunnableEntitys triggered by an InitEvent and therefore contradict the semantics of the RunnableEntity.

[constr_1258] Value of minimumStartInterval for RunnableEntitys triggered by an InitEvent ⌈The value of the attribute ExecutableEntity.minimumStartInterval for a RunnableEntitys that is triggered by an InitEvent shall always be set to 0 **at the time when the RTE is generated**.⌋*()*

Rationale: it does not make sense to talk about intervals of activating RunnableEntitys triggered by an InitEvent as these are not supposed to be executed repeatedly.

[constr_1259] Aggregation of AsynchronousServerCallPoint and AsynchronousServerCallResultPoint ⌈A RunnableEntity referenced by an InitEvent in the role startOnEvent may aggregate an AsynchronousServerCallPoint but it shall not aggregate an AsynchronousServerCallResultPoint **at the time when the RTE is generated**.⌋*()*

Rationale: as mentioned before WaitPoints shall not be aggregated by a RunnableEntitys triggered by an InitEvent in the role startOnEvent. It is allowed (although considered unlikely to happen) to have an AsynchronousServerCallPoint but it is not allowed to fetch the result of the call within the same RunnableEntity.

A RunnableEntity triggered by an InitEvent in the role startOnEvent *may* aggregate a SynchronousServerCallPoint but the usage of this configuration is discouraged.

[constr_1260] No mode disabling for InitEvents ⌈An InitEvent shall not have a reference to a ModeDeclaration in the role disabledMode **at the time when the RTE is generated**.⌋*()*

Rationale: the concept of RunnableEntity triggered by an InitEvent is (as mentioned before) orthogonal to the mode concept and therefore shall be implemented independent of modes.

7.3 RTEEvent

During execution, several RTEEvents will occur, such as the reception of a remote invocation of a ClientServerOperation on a PPortPrototype or a timeout on an RPortPrototype that is not receiving the VariableDataPrototypes it expects to receive.

[TPS_SWCT_01314] RTEEvent ⌈An RTEEvent defines:

- what the trigger for the occurrence of that RTEEvent is
- whether specific ModeDeclarations disable the processing of this RTEEvent
- which RunnableEntity shall be started when this RTEEvent occurs.

⌋*(RS_SWCT_03040)*

Class	***AbstractEvent*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the abstract ability to model an event that can be taken to implement application software or basic software in AUTOSAR.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*BswEvent*, *RTEEvent*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
activation Reason Representation	ExecutableEntity ActivationReason	0..1	ref	If the activationReasonRepresentation is referenced from the enclosing AbstractEvent this shall be taken as an indication that the latter contributes to the activating vector of this ExecutableEntity that owns the referenced ExecutableEntityActivationReason.

Table 7.10: AbstractEvent

Class	***RTEEvent*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	Abstract base class for all RTE-related events			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Subclasses	AsynchronousServerCallReturnsEvent, BackgroundEvent, DataReceiveErrorEvent, DataReceivedEvent, DataSendCompletedEvent, DataWriteCompletedEvent, ExternalTriggerOccurredEvent, InitEvent, InternalTriggerOccurredEvent, ModeSwitchedAckEvent, OperationInvokedEvent, OsTaskExecutionEvent, SwcModeManagerErrorEvent, SwcModeSwitchEvent, TimingEvent, TransformerHardErrorEvent			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	*RTEEvent* (abstract)			
disabledMode	ModeDeclaration	*	iref	Reference to the Modes that disable the Event. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=disabledMode.contextPort, disabledMode.contextModeDeclarationGroupPrototype, disabledMode.targetModeDeclaration **InstanceRef implemented by:**RModeInAtomicSwcInstanceRef
startOnEvent	RunnableEntity	0..1	ref	The referenced RunnableEntity starts when the corresponding RTEEvent is raised.

Table 7.11: RTEEvent

Class	**AsynchronousServerCallReturnsEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when an asynchronous server call is finished.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
eventSource	AsynchronousServerCallResultPoint	0..1	ref	The referenced AsynchronousServerCallResultPoint raises this AsynchronousServerCallReturnsEvent when the asynchronous server call returns.

Table 7.12: AsynchronousServerCallReturnsEvent

[constr_1940] Existence of attribute AsynchronousServerCallReturnsEvent.eventSource ⌈For each AsynchronousServerCallReturnsEvent, attribute eventSource shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**DataSendCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the referenced explicit data element has been sent or an error occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
eventSource	VariableAccess	0..1	ref	The referenced VariableAccess raises this DataSendCompletedEvent when the explicit write access was successful or an error occurred.

Table 7.13: DataSendCompletedEvent

[constr_1941] Existence of attribute DataSendCompletedEvent.eventSource ⌈For each DataSendCompletedEvent, attribute eventSource shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**DataWriteCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when an implicit write access was successful or an error occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eventSource	VariableAccess	0..1	ref	The referenced VariableAccess raises this DataWrite CompletedEvent when the implicit write access was successful or an error occurred.

Table 7.14: DataWriteCompletedEvent

[constr_1942] Existence of attribute DataWriteCompletedEvent.eventSource ⌈For each DataWriteCompletedEvent, attribute eventSource shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**DataReceivedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the referenced data element is received.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
data	VariableDataPrototype	0..1	iref	The referenced VariableDataPrototype raises this Data ReceivedEvent when the data has been received. **InstanceRef implemented by:**RVariableInAtomicSwc InstanceRef

Table 7.15: DataReceivedEvent

[constr_1943] Existence of attribute DataReceivedEvent.data ⌈For each DataReceivedEvent, attribute data shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**DataReceiveErrorEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the Com layer detects and notifies an error concerning the reception of the referenced ValiableDataPrototype.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
data	VariableDataPrototype	0..1	iref	The referenced VariableDataPrototype raises this Data ReceiveErrorEvent when there was an error during the reception. **InstanceRef implemented by:**RVariableInAtomicSwc InstanceRef

Table 7.16: DataReceiveErrorEvent

[constr_1944] Existence of attribute DataReceiveErrorEvent.data ⌈For each DataReceiveErrorEvent, attribute data shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	**OperationInvokedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the ClientServerOperation referenced in OperationInvokedEvent.operation shall be invoked.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
operation	ClientServerOperation	0..1	iref	This represents the ClientServerOperation which shall be invoked. **InstanceRef implemented by:**POperationInAtomicSwc InstanceRef

Table 7.17: OperationInvokedEvent

[constr_1945] Existence of attribute OperationInvokedEvent.operation ⌈For each OperationInvokedEvent, attribute operation shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1523] No mode disabling for OperationInvokedEvents ⌈An OperationInvokedEvent shall not have a reference to a ModeDeclaration in the role disabledMode **at the time when the RTE is generated**.⌋*()*

Rationale for the existence of [constr_1523]:

The RTE does not support the disabling of server RunnableEntitys by modes. Instead, the server shall respond with an explicit error code if the execution of the server operation is not possible in specific side conditions.

For more explanation about the semantics of meta-class TimingEvent, please refer to section 7.2.3.

Class	**BackgroundEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is used to start RunnableEntities that are supposed to be executed in the background.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 7.18: BackgroundEvent

Class	**SwcModeSwitchEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the specified mode change occurs.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
activation	ModeActivationKind	0..1	attr	Specifies if the event is raised on entering or exiting a specific mode or is raised on the transition between two modes.
mode (ordered)	ModeDeclaration	0..2	iref	The referenced mode or the transition between two modes raises this SwcModeSwitchEvent. **InstanceRef implemented by:**RModeInAtomicSwc InstanceRef

Table 7.19: SwcModeSwitchEvent

[constr_1946] Existence of attribute `SwcModeSwitchEvent.activation` ⌈For each `SwcModeSwitchEvent`, attribute `activation` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1947] Existence of reference `SwcModeSwitchEvent.mode` ⌈For each `SwcModeSwitchEvent`, the reference to `ModeDeclaration` in the role `mode` shall exist **`at the time when the RTE is generated`**.⌋*()*

Enumeration	**ModeActivationKind**
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	Kind of mode switch condition used for activation of an event, as further described for each enumeration field.
Aggregated by	BswModeSwitchEvent.activation, SwcModeSwitchEvent.activation
Literal	***Description***
onEntry	On entering the referred mode. **Tags:**atp.EnumerationLiteralIndex=0
onExit	On exiting the referred mode. **Tags:**atp.EnumerationLiteralIndex=1
onTransition	On transition of the 1st referred mode to the 2nd referred mode. **Tags:**atp.EnumerationLiteralIndex=2

Table 7.20: ModeActivationKind

Class	**ModeSwitchedAckEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the referenced ModeSwitchPoint has been processed or an error occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	ModeSwitchedAckEvent			
eventSource	ModeSwitchPoint	0..1	ref	The referenced ModeSwitchPoint raises this ModeSwitchedAckEvent when the ModeSwitchPoint has been processed.

Table 7.21: ModeSwitchedAckEvent

[constr_1948] Existence of attribute ModeSwitchedAckEvent.eventSource ⌈For each ModeSwitchedAckEvent, attribute eventSource shall exist **at the time when the RTE is generated**.⌋*()*

Class	**ExternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the referenced Trigger has occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
trigger	Trigger	0..1	iref	The referenced Trigger raises this ExternalTriggerOccurredEvent. **InstanceRef implemented by:**RTriggerInAtomicSwcInstanceRef

Table 7.22: ExternalTriggerOccurredEvent

[constr_1949] Existence of attribute ExternalTriggerOccurredEvent.trigger ⌈For each ExternalTriggerOccurredEvent, attribute trigger shall exist **at the time when the RTE is generated**.⌋*()*

Class	**InternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when the referenced InternalTriggeringPoint has occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
eventSource	InternalTriggeringPoint	0..1	ref	The referenced InternalTriggeringPoint raises this InternalTriggerOccurredEvent.

Table 7.23: InternalTriggerOccurredEvent

[constr_1950] Existence of attribute InternalTriggerOccurredEvent.eventSource ⌈For each InternalTriggerOccurredEvent, the attribute eventSource shall exist **at the time when the RTE is generated**.⌋*()*

Class	InitEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This RTEEvent is supposed to be used for initialization purposes, i.e. for starting and restarting a partition. It is not guaranteed that all RunnableEntities referenced by this InitEvent are executed before the 'regular' RunnableEntities are executed for the first time. The execution order depends on the task mapping.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.24: InitEvent

Class	TransformerHardErrorEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when data are received which should trigger a Client/Server operation or an external Trigger but during transformation of the data a hard transformer error occurred.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operation	ClientServerOperation	0..1	iref	This represents the ClientServerOperation for which the transformer can raise this TransformerHardErrorEvent. **InstanceRef implemented by:**POperationInAtomicSwcInstanceRef
requiredTrigger	Trigger	0..1	iref	This represents the Trigger for which the transformer can raise this TransformerHardErrorEvent. **InstanceRef implemented by:**RTriggerInAtomicSwcInstanceRef

Table 7.25: TransformerHardErrorEvent

[constr_1397] Existence of attributes of TransformerHardErrorEvent ⌈For any given TransformerHardErrorEvent, **either** the attribute TransformerHardErrorEvent.operation **or** TransformerHardErrorEvent.requiredTrigger shall exist **at the time when the contract phase generation is executed**.⌋*()*

In other words, the attributes operation and requiredTrigger of meta-class TransformerHardErrorEvent shall be used mutually exclusive.

[TPS_SWCT_01315] Interaction of RunnableEntity with RTEEvent ⌈As described in the Virtual Functional Bus specification [3], the RunnableEntitys of an AtomicSwComponentType can interact with the occurrence of such RTEEvents in two ways:

- the RTE can be instructed to enable a specific RunnableEntity when the RTEEvent occurs
- the RTE can provide WaitPoints, that allow a RunnableEntity to block until an RTEEvent in a set of RTEEvents occurs.

⌋*(RS_SWCT_03040)*

Class	OsTaskExecutionEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This RTEEvent is supposed to execute RunnableEntities which have to react on the execution of specific OsTasks. Therefore, this event is unconditionally raised whenever the OsTask on which it is mapped is executed. The main use case for this event is scheduling of Runnables of Complex Drivers which have to react on task executions. **Tags:**atp.Status=draft			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.26: OsTaskExecutionEvent

[constr_10016] Applicability of OsTaskExecutionEvent ⌈An OsTaskExecutionEvent is only applicable for a SwcInternalBehavior in the context of a ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponentType, or ServiceSwComponentType **at any time in the workflow**.⌋*()*

7.3.1 Defining an Event

The description of the SwcInternalBehavior includes a description of all RTEEvents that the SwcInternalBehavior of the AtomicSwComponentType relies on.

[TPS_SWCT_01316] Abstract base class RTEEvent ⌈The meta-class RTEEvent shows up as an "abstract" base-class in the meta-model: the exact attributes of the RTEEvent depend on the specific sub-class of RTEEvent that is used for the purpose.⌋*()*

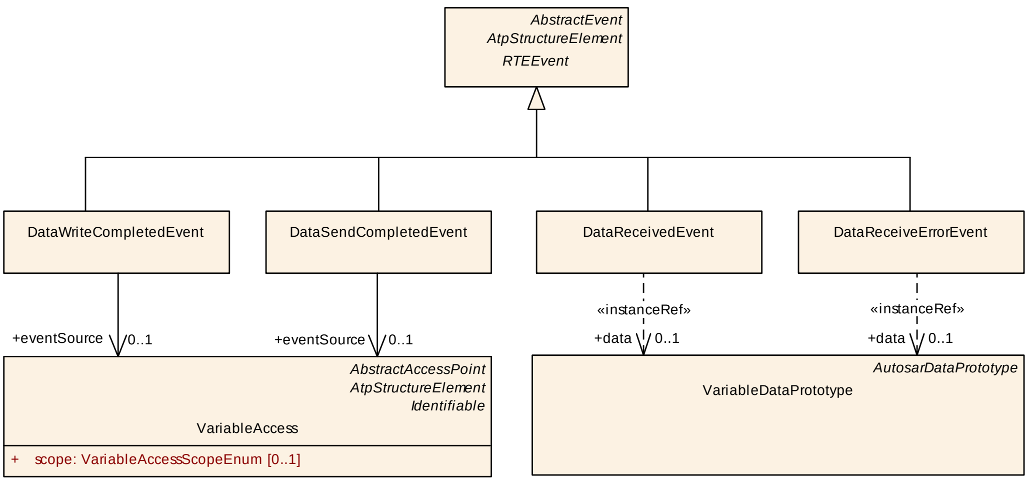


Figure 7.9: RTEEvents used in the context of sender/receiver communication

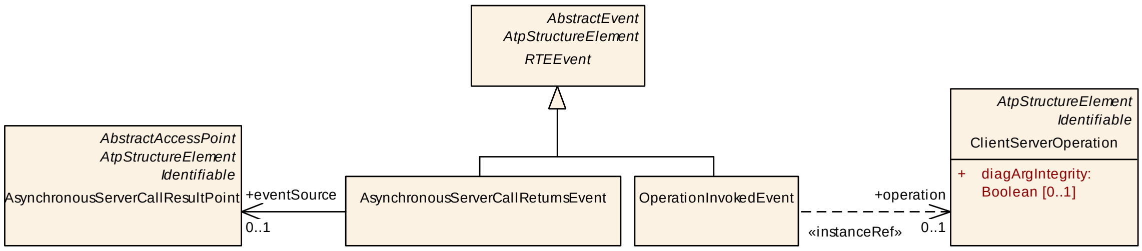


Figure 7.10: RTEEvents used in the context of client/server communication

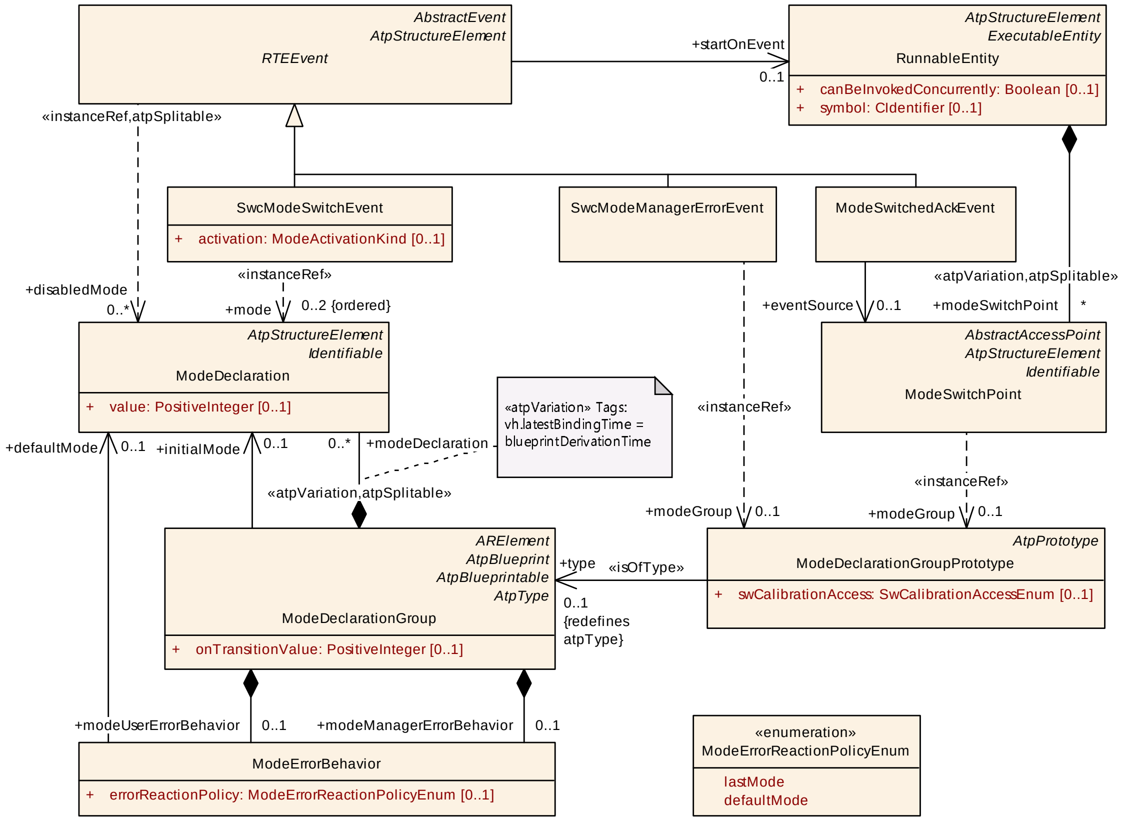


Figure 7.11: RTEEvents used in the context of mode communication

Please note that more explanation about the semantics of the meta-classes SwcMode-ManagerErrorEvent and ModeErrorBehavior can be found in section 9.4.

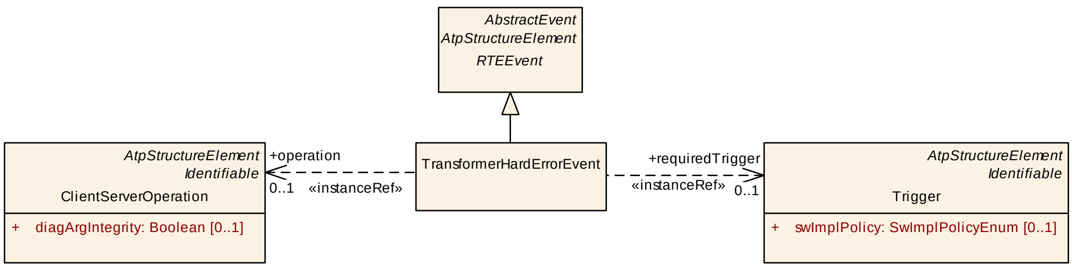


Figure 7.12: RTEEvent used in the context of data transformation

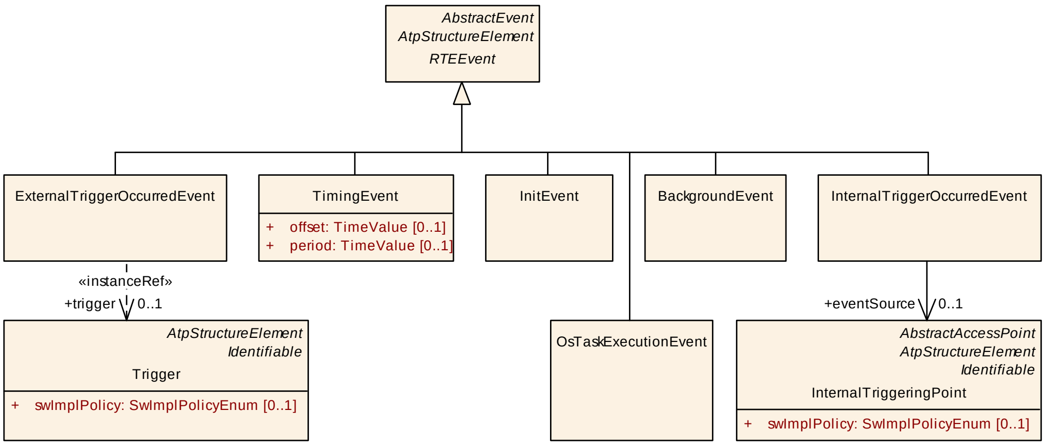


Figure 7.13: RTEEvents for purposes other than communication

The details of the various kinds of concrete RTEEvents (such as the TimingEvent, DataSendCompletedEvent, etc.), is described in chapters 7.5.1, 7.5.2 and 7.2.3.

7.3.2 Defining how to Respond to an Event

[TPS_SWCT_01317] RTE triggers RunnableEntity in response to occurring RTEEvent ⌈If the software-component description contains a reference from an RTEEvent to a RunnableEntity in the role startOnEvent, it is the responsibility of the RTE to trigger the execution of the corresponding RunnableEntity when the RTEEvent occurs.⌋*(RS_SWCT_03040)*

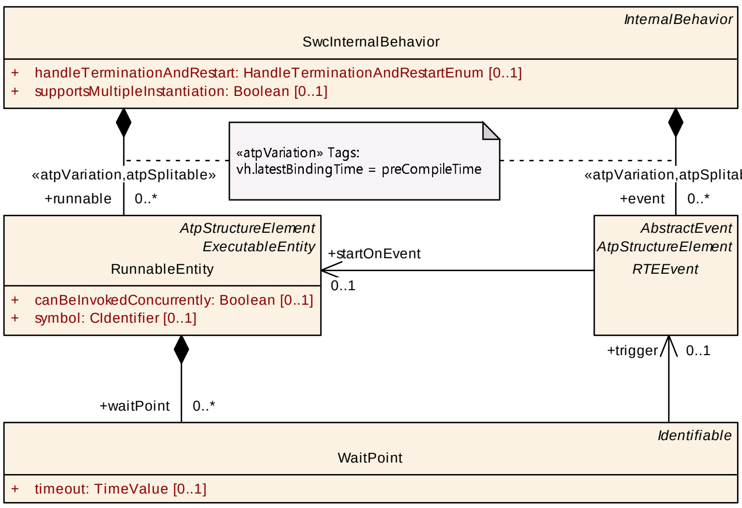


Figure 7.14: Description of the interaction between an RTEEvent and RunnableEntitys

Figure 7.14 gives an overview of the modeling of WaitPoint.

[TPS_SWCT_01318] RunnableEntity and WaitPoint ⌈In case the RunnableEntity wants to block and wait for RTEEvents (which makes the RunnableEntity into a cat. 2 RunnableEntity), the description of the RunnableEntity may include the definition of a WaitPoint.

Such a WaitPoint contains a reference to an RTEEvent that can unblock the specific WaitPoint. In other words: the WaitPoint will block until the referenced RTEEvents occurs or the period specified in the attribute timeout expires.⌋*(RS_SWCT_03040)*

[constr_1091] RTEEvents that may reference a WaitPoint ⌈A WaitPoint shall only be referenced from the listed RTEEvents:

- DataReceivedEvent
- DataSendCompletedEvent
- ModeSwitchedAckEvent
- AsynchronousServerCallReturnsEvent

This rule shall be imposed **at the time when the contract phase is executed**.⌋*()*

[TPS_SWCT_01319] RTEEvent can be used to trigger WaitPoints in different RunnableEntitys ⌈It is in general possible that a single RTEEvent can be used to trigger WaitPoints in different RunnableEntitys.⌋*()*

Concerning DataReceivedEvents consider as well [constr_2021].

Class	**WaitPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This defines a wait-point for which the RunnableEntity can wait.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Aggregated by	RunnableEntity.waitPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
timeout	TimeValue	0..1	attr	Time in seconds before the WaitPoint times out and the blocking wait call returns with an error indicating the timeout.
trigger	RTEEvent	0..1	ref	This is the RTEEvent this WaitPoint is waiting for.

Table 7.27: WaitPoint

[constr_1951] Existence of attribute WaitPoint.timeout ⌈For each WaitPoint, attribute timeout shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1952] Existence of reference WaitPoint.trigger ⌈For each WaitPoint, the reference to RTEEvent in the role trigger shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1096] SwcModeSwitchEvent and WaitPoint ⌈A RunnableEntity that has a WaitPoint shall not be referenced by a SwcModeSwitchEvent **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01320] RunnableEntitys of category 2 ⌈RunnableEntitys that aggregate a WaitPoint are by definition of category 2 and therefore are not required to terminate ever. It is therefore difficult to let a RunnableEntity of category 2 implement a mode switch.⌋*(RS_SWCT_03040)*

[constr_1097] RunnableEntity that has a WaitPoint ⌈A RunnableEntity that has a WaitPoint shall not be referenced by an RTEEvent that has a reference in the role disabledMode **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01324] Mode switches need to be completed in finite time ⌈Mode switches need to be completed in finite time and a RunnableEntity that has a WaitPoint can never guarantee that the WaitPoint is resolved within finite time.⌋*(RS_SWCT_03040)*

In addition to this, the RunnableEntity with a WaitPoint that would be affected by a mode disabling would typically already run when the mode disabling applies. It could not be terminated at this point in time.

7.4 Communication among Runnable Entities

It is taken for granted that particular RunnableEntitys within a specific AtomicSwComponentType will need to communicate among each other.

[TPS_SWCT_01321] Communication among RunnableEntitys ⌈The RTE needs to provide synchronization mechanisms to the RunnableEntitys such that safe (in the multi-threading sense) exchange of data is possible.

In this case, the use of PortPrototypes is (although technically feasible) not required for the purpose.⌋*(RS_SWCT_00120)*

[TPS_SWCT_01592] Communication among RunnableEntitys of different instances of the same AtomicSwComponentType ⌈The communication among RunnableEntitys of different instances of the same AtomicSwComponentType is only supported via PortPrototypes.⌋*(RS_SWCT_00120)*

Several concepts for implementing communication among RunnableEntitys can be identified.

As an introduction, the section 2.3.1 describes the various techniques that the RTE might use to provide efficient interaction between RunnableEntitys within one AtomicSwComponentType.

Two possible approaches for formal specification of this kind of communication are described:

- Specifying that several RunnableEntitys belong in a specific ExclusiveArea
- Specifying the data exchanged between the RunnableEntitys

7.4.1 Description Possibility 1: Exclusive Area

This section describes how the concept of ExclusiveAreas can be used in the description of the SwcInternalBehavior of an AtomicSwComponentType.

Please note that ExclusiveAreas are actually owned by the base class of SwcInternalBehavior, i.e. InternalBehavior. These ExclusiveAreas do not imply a specific implementation (e.g. with mutual-exclusion semaphores).

Class	**ExclusiveArea**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Prevents an executable entity running in the area from being preempted.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*InternalBehavior*.exclusiveArea			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.28: ExclusiveArea

[TPS_SWCT_01031] ExclusiveArea ⌈An ExclusiveArea merely specifies a constraint on the scheduling policy and configuration of the RTE:

If two or more RunnableEntitys refer to the same ExclusiveArea only one of these RunnableEntitys is allowed to be executed while being inside that ExclusiveArea.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

In other words: these RunnableEntitys shall not run concurrently (preempt each other) while executing inside the ExclusiveArea.

Please find more details about the formal definition of meta-class ExclusiveArea in Figure 7.15.

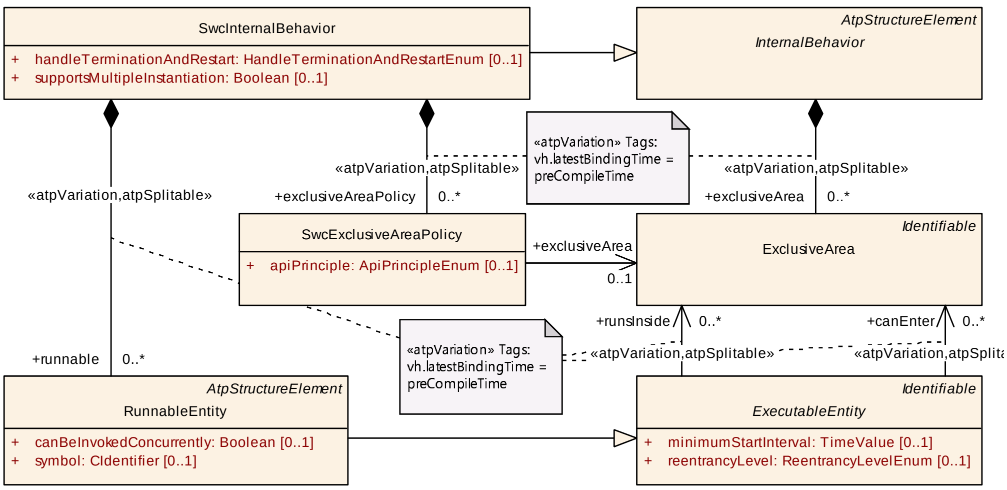


Figure 7.15: Description of logical exclusive areas

[TPS_SWCT_01049] Two ways to use the ExclusiveAreas ⌈There are in general two ways to use the ExclusiveAreas. During its execution, a RunnableEntity can enter and exit an ExclusiveArea (in which case ExecutableEntity.canEnter shall exist).

As an alternative, it can be specified that the entire execution of a given RunnableEntity shall be guarded by an ExclusiveArea (this requires the existence of ExecutableEntity.runsInside).⌋*(RS_SWCT_00120, RS_SWCT_02090)*

Please note that the options for entering an ExclusiveArea are documented in section 7.4.1.1 and section 7.4.1.2

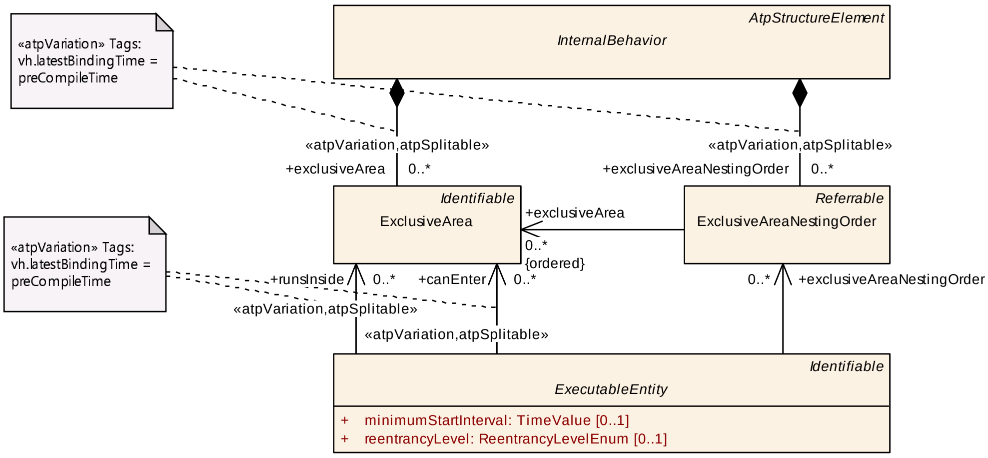


Figure 7.16: Description of nested usage of ExclusiveArea

[TPS_SWCT_01457] **ExclusiveAreaNestingOrder** ⌈The optional ExclusiveAreaNestingOrders shall (if used at all) describe possible nesting orders (including single ExclusiveAreas) which can occur in the RunnableEntity. Each possible locking situation requires its own ExclusiveAreaNestingOrder.⌋*(RS_SWCT_03055)*

[TPS_SWCT_01458] **Indicate that the locking behavior is fully described for RunnableEntity** ⌈All ExclusiveAreas which are configured in the InternalBehavior should be referenced by an ExclusiveAreaNestingOrder to indicate that the locking behavior is fully described for this RunnableEntity.⌋*(RS_SWCT_03055)*

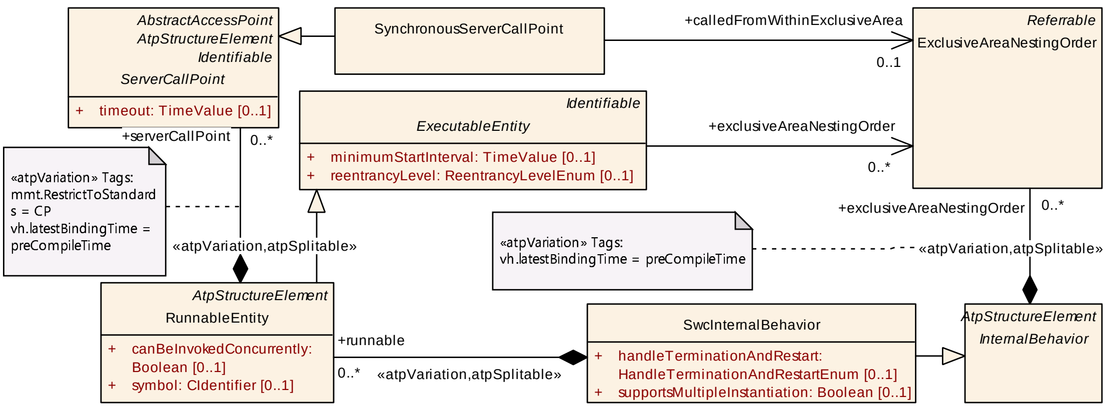


Figure 7.17: Nested usage of ExclusiveArea and the impact on SynchronousServerCallPoint

[TPS_SWCT_01459] Locking behavior is not described for this RunnableEntity
⌈If ExclusiveAreas are not referenced by any ExclusiveAreaNestingOrder (this is the default scenario), this means that the locking behavior is not described for this RunnableEntity and the provided information might be incomplete and cannot be used for a global offline analysis of locking behavior.⌋*(RS_SWCT_03055)*

An ExclusiveAreaNestingOrder is aggregated by the InternalBehavior that in turn also owns RunnableEntity.

Class	**ExclusiveAreaNestingOrder**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the ability to define a nesting order of ExclusiveAreas. A nesting order (that may occur in the executable code) is formally defined to be able to analyze the resource locking behavior.			
Base	*ARObject*, *Referrable*			
Aggregated by	*InternalBehavior*.exclusiveAreaNestingOrder			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
exclusiveArea (ordered)	ExclusiveArea	*	ref	This represents a specific scenario of how Exclusive Areas can be used in terms of the nesting order.

Table 7.29: ExclusiveAreaNestingOrder

[TPS_SWCT_01460] Relation of SynchronousServerCallPoint to ExclusiveAreaNestingOrder ⌈In case other RunnableEntitys are invoked synchronously from within the RunnableEntity the ExclusiveAreaNestingOrder can then be referenced by one or several SynchronousServerCallPoints to specify the calling environment of the invoked server with regard to ExclusiveAreas.⌋ *(RS_SWCT_03055)*

The purpose of this configuration is to analyze the resource locking behavior for complete call trees.

7.4.1.1 Entire Runnable Runs in the Exclusive Area

[TPS_SWCT_01050] RunnableEntity always runs inside an ExclusiveArea
⌈In the first approach, the formal description specifies that certain RunnableEntitys always run inside an ExclusiveArea.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

For example, if the formal description specifies that both RunnableEntity 'r1' and RunnableEntity 'r2' run within ExclusiveArea 's1', the RTE shall make sure that RunnableEntitys 'r1' and 'r2' never run concurrently; the scheduler should never preempt 'r1' to run 'r2'.

Note that this pattern does not force the RTE to implement this by using semaphores or mutexes that are taken before the RunnableEntity starts and given when the RunnableEntity returns. It only obliges the RTE to make sure that both RunnableEntitys are never running concurrently.

This requirement could be implemented by several of the implementation strategies described above. For example:

1. Scheduling strategy: if, for example, RunnableEntitys 'r1' and 'r2' are mapped to the same task, the criterion is automatically satisfied. For this purpose it is necessary to make sure that the OS can only execute a single instance of the task into which the RunnableEntitys are put.
2. Mutual exclusion semaphores: in case 'r1' and 'r2' are mapped to different tasks ('T1', respectively 'T2'), the OS shall make sure that while 'T1' is executing 'r1', 'T2' running 'r2' can never preempt it and vice-versa. This could be implemented by taking a mutual-exclusion semaphore before executing 'r1' (or 'r2') in the context of 't1' (or 't2') and returning the semaphore on exiting the RunnableEntity.

7.4.1.2 Runnable would Dynamically Enter and Leave the Exclusive Area

[TPS_SWCT_01051] RunnableEntity explicitly enters and leaves a specific ExclusiveArea ⌈In the second approach, the RunnableEntity would explicitly make API-calls to the RTE within the implementation of the RunnableEntity to enter and leave a specific ExclusiveArea.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

This could, for example, be implemented by means of the priority ceiling concept described in chapter 2.3.1.3.

Additionally, it is possible to define the execution time the RunnableEntity will spend in this ExclusiveArea segment. Please note that although this aspect is described in [6] the concept can be applied to software-components as well.

7.4.1.3 Configuration of API Generation

For certain usage scenarios of ExclusiveAreas it is considered advantageous if each RunnableEntity uses a distinct set of enter and exit APIs.

This distinct set of APIs support ExclusiveArea implementations where for the RunnableEntity(s) with the highest priority the lock is omitted.

This is possible when the RunnableEntity(s) with the highest priority can't be interrupted by RunnableEntitys scheduled with lower priority.

To support this kind of implementation, the software-component description has to state (by means of attribute SwcInternalBehavior.exclusiveAreaPolicy.apiPrinciple) that it requests APIs individually for each RunnableEntity referencing an ExclusiveArea in the role canEnter.

Class	**SwcExclusiveAreaPolicy**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior
Note	Options how to generate the ExclusiveArea related APIs. If no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply.

▽

△

Class	SwcExclusiveAreaPolicy			
Base	*ARObject*			
Aggregated by	SwcInternalBehavior.exclusiveAreaPolicy			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
apiPrinciple	ApiPrincipleEnum	0..1	attr	Specifies for this ExclusiveArea if either one common set of Enter and Exit APIs for the whole software component is requested from the Rte or if the set of Enter and Exit APIs is expected per RunnableEntity. The default value is "common".
exclusiveArea	ExclusiveArea	0..1	ref	This reference represents the ExclusiveArea for which the policy applies.

Table 7.30: SwcExclusiveAreaPolicy

[constr_1953] Existence of attribute SwcExclusiveAreaPolicy.apiPrinciple ⌈For each SwcExclusiveAreaPolicy that refers to an exclusiveArea, attribute apiPrinciple shall exist **at the time when the RTE is generated**.⌋*()*

Enumeration	ApiPrincipleEnum
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior
Note	This enumeration represents the ability to control the granularity of API generation.
Aggregated by	BswExclusiveAreaPolicy.apiPrinciple, SwcExclusiveAreaPolicy.apiPrinciple
Literal	**Description**
common	The Rte or SchM API is provided for the whole software component / BSW Module **Tags:**atp.EnumerationLiteralIndex=0
perExecutable	The Rte or SchM API is provided for a specific ExecutableEntity of a software component / BSW Module **Tags:**atp.EnumerationLiteralIndex=1

Table 7.31: ApiPrincipleEnum

[TPS_SWCT_01713] ExclusiveArea is entered and exited by a common set of APIs ⌈If the value of attribute SwcExclusiveAreaPolicy.apiPrinciple is set to ApiPrincipleEnum.common then the RTE provides **one set** of enter and exit APIs to be shared among all RunnableEntitys of the whole software-component.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

In this case, the same enter and exit code is executed by all affected RunnableEntitys and there is no way to have a special treatment for the RunnableEntity(s) executed in the context with the highest priority.

[TPS_SWCT_01714] ExclusiveArea is entered and exited by an individual set of APIs ⌈If the value of attribute SwcExclusiveAreaPolicy.apiPrinciple is set to ApiPrincipleEnum.perExecutable then the RTE provides **individual** sets of APIs for entering and exiting ExclusiveAreas for each affected RunnableEntity.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

In this case, the implementation of enter and exit code for the RunnableEntity executed in the execution context with the highest priority can be left empty.

In order to avoid the existence of contradicting settings of SwcExclusiveAreaPolicys for one ExclusiveArea [constr_1468] applies.

[constr_1468] Limitation on the number of SwcExclusiveAreaPolicys ⌈An ExclusiveArea shall only be referenced by **at most** one SwcExclusiveAreaPolicy **at the time when the contract phase generation is executed.**⌋*()*

7.4.2 Description Possibility 2: Inter-Runnable Variable

For certain cases the ExclusiveArea concept does not provide enough information to configure the RTE correctly. In these cases it may be advised to opt for a different approach that is based on the guarded access to variables protected by the RTE.

For the purpose of identifying pieces of data that shall be accessed concurrently from different RunnableEntitys formal support is required. In AUTOSAR, this aspect is summarized under the term "inter-runnable variable".

[TPS_SWCT_01052] Inter-runnable variable ⌈These so-called "inter-runnable variables" are described with the element VariableDataPrototype aggregated in the role explicitInterRunnableVariable or implicitInterRunnableVariable.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

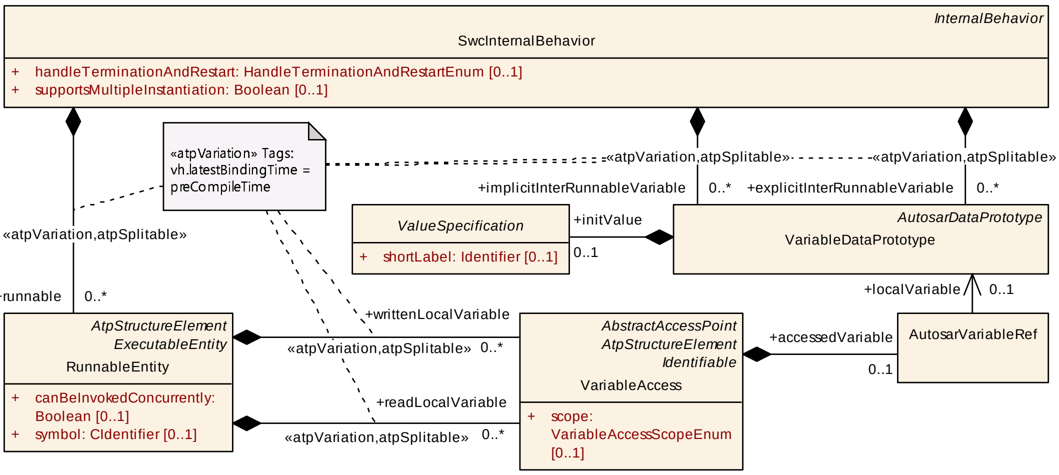


Figure 7.18: implicitInterRunnableVariable vs. explicitInterRunnableVariable

[TPS_SWCT_01053] Relationship of interchanged data with RunnableEntitys ⌈Furthermore, the relationship of these data with RunnableEntitys shall be specified.

For this specific purpose, RunnableEntity aggregates VariableAccess in the roles readLocalVariable and writtenLocalVariable.

Also, SwcInternalBehavior aggregates VariableDataPrototype in the roles explicitInterRunnableVariable and implicitInterRunnableVariable.

The connection between RunnableEntity and the explicitInterRunnableVariable and implicitInterRunnableVariable is created if the reference AutosarVariableRef.localVariable to the respective VariableDataPrototype exists.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

[TPS_SWCT_01521] Use AutosarVariableRef.localVariable for referencing inter-runnable variables ⌈A RunnableEntity that defines a VariableAccess in role writtenLocalVariable and readLocalVariable shall make use of AutosarVariableRef.localVariable.⌋*(RS_SWCT_03040)*

[constr_2026] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role writtenLocalVariable and readLocalVariable ⌈A VariableDataPrototype in the localVariable reference needs to be owned by the same SwcInternalBehavior as this RunnableEntity belongs to, and the referenced VariableDataPrototype has to be defined in the role implicitInterRunnableVariable or explicitInterRunnableVariable.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Obviously, the data-type of an implicitInterRunnableVariable or explicitInterRunnableVariable is described by the data type of the VariableDataPrototype (which is derived from DataPrototype).

[TPS_SWCT_01637] Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable ⌈It is possible (but not mandatory) to define an initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable.

For this purpose the VariableDataPrototype in the role of explicitInterRunnableVariable or implicitInterRunnableVariable is able to aggregate a ValueSpecification in the role initValue.⌋*(RS_SWCT_02090)*

The statement made by [TPS_SWCT_01637] is reflected by Figure 7.18

[TPS_SWCT_01522] No initial value is specified for implicitInterRunnableVariable or explicitInterRunnableVariable ⌈Please note that the behavior is undefined if no initial value is specified and a RunnableEntity reads an implicitInterRunnableVariable or explicitInterRunnableVariable before it is actually written to by another RunnableEntity.⌋*(RS_SWCT_03040)*

As already mentioned before, the concept of an "inter-runnable variable" can be used in *two different flavors* This is indicated by the two different roles explicitInterRunnableVariable or implicitInterRunnableVariable in which the VariableDataPrototype serving as the "inter-runnable variable" is aggregated.

These resemble the communication principles applied for the communication on the level of SwComponentTypes.

Please note that the two different kinds of inter-runnable variables are accessed via different RTE [2] API calls.

[TPS_SWCT_01054] Semantics of the explicitInterRunnableVariable ⌈The semantics of the explicitInterRunnableVariable is that *explicit* implies the direct access to the value of a VariableDataPrototype used in the role explicitInterRunnableVariable.

By this means it is possible to get different values for a specific VariableDataPrototype each time the corresponding API call is executed.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

[TPS_SWCT_01055] Semantics of implicitInterRunnableVariable ⌈The implicitInterRunnableVariable corresponds to an execution model where the value of an VariableDataPrototype does not change (for the reading RunnableEntity, obviously) during the runtime of a RunnableEntity.⌋*(RS_SWCT_00120, RS_SWCT_02090)*

This approach is in detail described in chapter 2.3.1.4.

[constr_1296] DataPrototypes used as explicitInterRunnableVariable or implicitInterRunnableVariable and category DATA_REFERENCE ⌈A VariableDataPrototype shall not be aggregated by SwcInternalBehavior in either the role:

- explicitInterRunnableVariable, or
- implicitInterRunnableVariable

if the VariableDataPrototype (after potential indirections via TYPE_REFERENCE are resolved) is either typed by, or mapped to, an:

- ImplementationDataType of category DATA_REFERENCE, or
- ImplementationDataType that contains subElements that (after potential indirections via TYPE_REFERENCE are resolved) are of category DATA_REFERENCE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

7.4.3 Inter Runnable Triggering

The concept of *inter-runnable triggering* allows one RunnableEntity to trigger another RunnableEntity within an AtomicSwComponentType. This approach conceptually supports the decoupling of calculation and processing sequences inside a software-component.

By mappings of the InternalTriggerOccurredEvents to OS Tasks running at different priorities the triggered RunnableEntitys are in turn executed with a different priority as the triggering RunnableEntity.

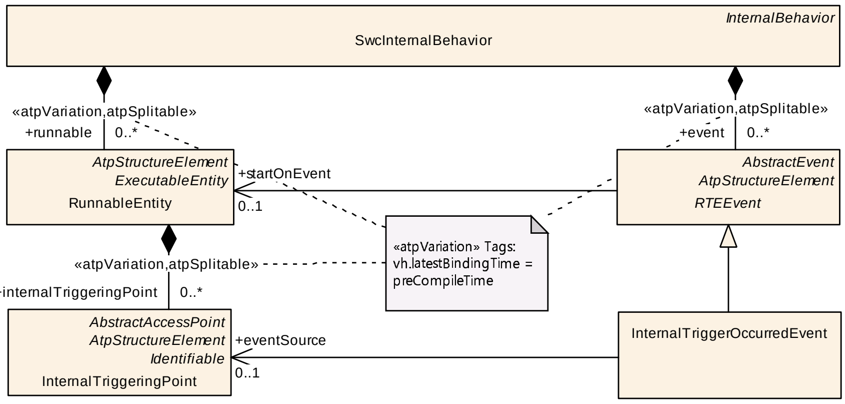


Figure 7.19: Model of software-component Inter Runnable Triggering

For example, a cyclically triggered RunnableEntity which shall not exceed a certain worst case execution time (WCET) activates a second RunnableEntity if an error occurred in order to be able to execute a (potentially) time-consuming exception-handling on a lower level of priority.

As illustrated in Figure 7.19, the triggering RunnableEntity needs an InternalTriggeringPoint.

The activation of RunnableEntitys in the same software-component instance is affected through the generic event-handling mechanism.

[TPS_SWCT_01523] Internal trigger event ⌈A RunnableEntity that shall be activated at the occurrence of an internal trigger event is defined by means of an InternalTriggerOccurredEvent which references the particular InternalTriggeringPoint and additionally the to-be-activated RunnableEntity.⌋*(RS_SWCT_03040)*

[TPS_SWCT_01022] Queued processing of internal trigger ⌈The attribute InternalTriggeringPoint.swImplPolicy can be used to specify a requirement whether the internal triggering of the enclosing RunnableEntity using the given InternalTriggeringPoint shall be queued.⌋*(RS_SWCT_03040)*

[constr_1182] Allowed values for InternalTriggeringPoint.swImplPolicy ⌈The **only** allowed values for the attribute swImplPolicy of meta-class InternalTriggeringPoint are either STANDARD (in which case the processing of the internal triggering does not use a queue) or QUEUED (in which case the processing of internal triggering positively uses a queue).

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

Class	**InternalTriggeringPoint**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger
Note	If a RunnableEntity owns an InternalTriggeringPoint it is entitled to trigger the execution of Runnable Entities of the corresponding software-component.

▽

△

Class	**InternalTriggeringPoint**			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.internalTriggeringPoint			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.

Table 7.32: InternalTriggeringPoint

The description of the corresponding *external* trigger communication is contained in chapter 7.5.3.

7.5 Data Access of RunnableEntities

This section describes the communication properties of an AtomicSwComponentType. This is done mainly from the point of view of a RunnableEntity (the concept of a RunnableEntity is introduced in chapter 7.2).

However, the usage of a PortPrototype in a specific role within an AtomicSwComponentType also has an impact on communication behavior.

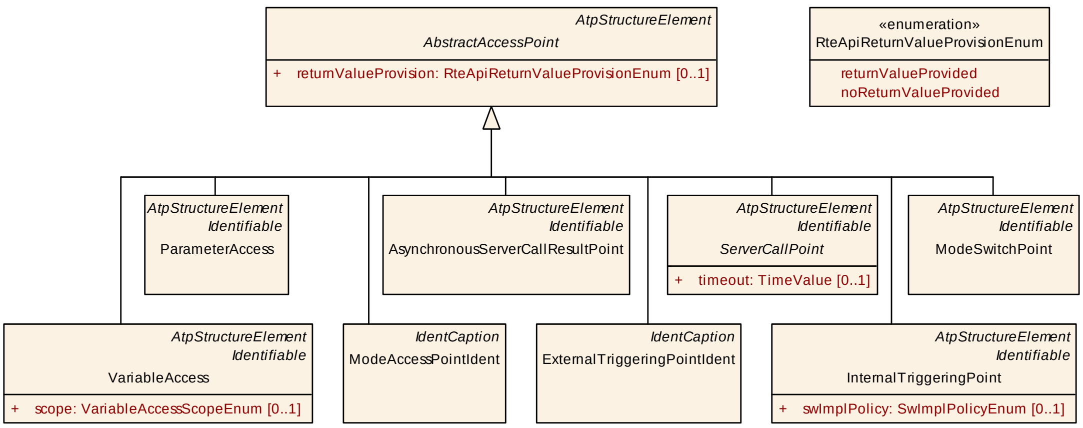


Figure 7.20: Modeling of the AbstractAccessPoint

Access of RunnableEntitys to the different elements in PortInterfaces or the InternalBehavior are modeled by a set of meta-classes specific to the communication pattern and the kind of access.

Nevertheless, all of those meta-classes inherit from AbstractAccessPoint in order to enable the ability to be referenced in a harmonized way to by additional descriptions.

Class	***AbstractAccessPoint*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::AccessCount			
Note	Abstract class indicating an access point from an ExecutableEntity.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	AsynchronousServerCallResultPoint, ExternalTriggeringPointIdent, InternalTriggeringPoint, ModeAccessPointIdent, ModeSwitchPoint, ParameterAccess, *ServerCallPoint*, VariableAccess			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
returnValueProvision	RteApiReturnValueProvisionEnum	0..1	attr	This attribute controls the provision of return values for RTE APIs that correspond to the enclosing access point.

Table 7.33: AbstractAccessPoint

Enumeration	**RteApiReturnValueProvisionEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::AccessCount
Note	This meta-class provides values to control how return values from RTE APIs are provided.
Aggregated by	*AbstractAccessPoint*.returnValueProvision
Literal	**Description**
noReturnValueProvided	The RTE API shall not provide a return value. **Tags:**atp.EnumerationLiteralIndex=1
returnValueProvided	The RTE API shall provide a return value. **Tags:**atp.EnumerationLiteralIndex=0

Table 7.34: RteApiReturnValueProvisionEnum

RunnableEntity that access DataPrototypes in the context of PortPrototypes are **not** allowed to extend the data access to sub-elements of the respective DataPrototypes.

For example, assume a DataPrototype that effectively implements a structure of two elements A and B. It is **not** supported to only send or receive only element A or B of the structure.

This assertion leads to the existence of [constr_1429] and, by extension, [constr_1430].

[constr_1429] Access to data within PortPrototypes from within RunnableEntitys ⌈For a VariableAccess that is aggregated in the roles

- RunnableEntity.dataWriteAccess
- RunnableEntity.dataReadAccess
- RunnableEntity.dataSendPoint
- RunnableEntity.dataReceivePointByArgument
- RunnableEntity.dataReceivePointByValue

the existence of the following attributes is not allowed:

- VariableAccess.accessedVariable.autosarVariable.contextDataPrototype

- VariableAccess.accessedVariable.autosarVariable.rootVariableDataPrototype
- VariableAccess.accessedVariable.autosarVariableInImplDatatype
- VariableAccess.accessedVariable.localVariable

In other words: in this case, only the references

- VariableAccess.accessedVariable.autosarVariable.portPrototype and
- VariableAccess.accessedVariable.autosarVariable.targetDataPrototype

shall exist and the latter shall **exclusively** refer to a VariableDataPrototype that is aggregated as either

- SenderReceiverInterface.dataElement or
- NvDataInterface.nvData.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1430] Access to local data from within RunnableEntitys ⌈For VariableAccess that is aggregated in the roles

- RunnableEntity.writtenLocalVariable
- RunnableEntity.readLocalVariable

the existence of the following attributes is not allowed:

- VariableAccess.accessedVariable.autosarVariableInImplDatatype
- VariableAccess.accessedVariable.autosarVariable

In other words, **only** the reference VariableAccess.accessedVariable.localVariable shall be used in this case.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1431] Access to parameters from within RunnableEntitys ⌈For a ParameterAccess that is aggregated in the role RunnableEntity.parameterAccess the existence of the following attributes is not allowed:

- ParameterAccess.accessedParameter.autosarParameter.contextDataPrototype
- ParameterAccess.accessedParameter.autosarParameter.rootParameterDataPrototype

In other words: in this case, **one** of the following alternatives is allowed to exist:

- a combination of
  - ParameterAccess.accessedParameter.autosarParameter.portPrototype and
  - ParameterAccess.accessedParameter.autosarParameter.targetDataPrototype that **exclusively** refers to a ParameterDataPrototype aggregated by a ParameterInterface in the role parameter.
- ParameterAccess.accessedParameter.localParameter that refers to a ParameterDataPrototype that is either aggregated as
  - InternalBehavior.constantMemory or
  - SwcInternalBehavior.perInstanceParameter or
  - SwcInternalBehavior.sharedParameter.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

7.5.1 RunnableEntities and Sender Receiver Communication

This section describes aspects relevant for the sender-receiver communication of a software-component. These mainly influence the behavior and API of the AUTOSAR RTE.

[TPS_SWCT_01322] Interaction patterns for the application of the sender-receiver paradigm ⌈The possible interaction patterns for the application of the sender-receiver paradigm are explained, namely:

1. Data-access in a cat. 1 RunnableEntity,
2. explicit sending,
3. the DataSendCompletedEvent: dealing with the success/failure of an explicit send, and
4. the DataReceivedEvent: responding to the reception of data
5. the DataReceiveErrorEvent: notifying an error concerning the reception of data.

⌋*(RS_SWCT_00200)*

7.5.1.1 Terminology

The AUTOSAR meta-model foresees two different approaches for sender-receiver communication. These are described in detail in chapters 7.5.1.2 and 7.5.1.3. However, it turned out that it is rather cumbersome to discuss issues of communication approaches directly on the basis of meta-classes and their attributes.

Therefore, it seems appropriate to introduce a dedicated terminology for this purpose. The approach eventually selected was originally introduced by the contributors to the RTE specification.

This terminology proposes to use the term “implicit” for communication based on *data-access* (for more information about details of this approach please consult chapter 7.5.1.2) and “explicit” for communication based on so-called *data-points* (please refer to chapter 7.5.1.3).

The motivation for the differentiation between “implicit” and “explicit” was originally the characteristics of the RTE specification that foresaw an API for handling a `dataSendPoint` or `dataReceivePointByValue` in contrast to the *data-access* that was supposed to be part of the function signature (therefore, no API was required) of a specific `RunnableEntity`.

Although the specification of the RTE changed in the meantime (and the original motivation no longer applies) it turned out that the terminology based on “implicit” and “explicit” communication was already widely used within AUTOSAR.

As no consensus could be reached over alternative proposals this terminology approach is taken over by this document as well.

7.5.1.2 Data Access

[TPS_SWCT_01323] Read and write access to a `dataElement` ⌈The `SwcInternalBehavior` may specify that a `RunnableEntity` needs read-access (respectively write-access) to the `VariableDataPrototype`s in the role `dataElement` of an `RPortPrototype` (respectively `PPortPrototype`, or `PRPortPrototype`).⌋*(RS_SWCT_00200)*

[TPS_SWCT_01325] Read and write access is only applicable for `RunnableEntity`s of category 1 ⌈The usage of the data-access mechanism to the `VariableDataPrototype`s is appropriate for cat. 1 `RunnableEntity`s only because it by concept guarantees finite response time (as opposed to e.g. unlimited blocking wait for some data).⌋*(RS_SWCT_00200)*

For more explanation, let’s suppose a cat. 2 `RunnableEntity` would have a `dataReadAccess` and a `dataWriteAccess`. The received `dataElement` would be updated **before** the `RunnableEntity` actually starts being executed and even if the `RunnableEntity` runs for a very long time the value of the `dataElement` would remain as is and never change.

On the other hand, the RunnableEntity might use its dataWriteAccess to perform a write access on the dataElement but the actual value might never make it beyond the RunnableEntity because

1. the latter is not required to terminate ever and
2. the actual write access is executed *after* the RunnableEntity terminates.

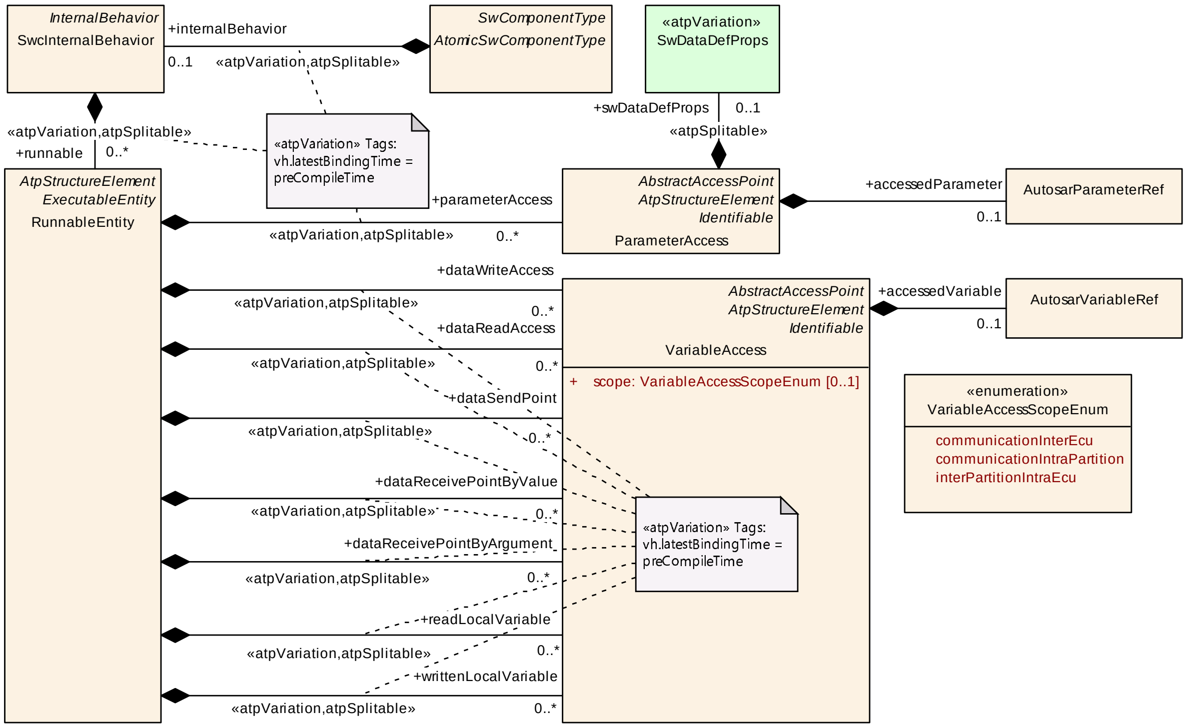


Figure 7.21: DataReadAccess and DataWriteAccess

Class	**VariableAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableData Prototype. The kind of access is specified by the role in which the class is used.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *ReceiverComSpec*.replaceWith, RunnableEntity.dataReadAccess, RunnableEntity.dataReceivePointByArgument, RunnableEntity.dataReceivePointByValue, RunnableEntity.dataSendPoint, RunnableEntity.dataWriteAccess, RunnableEntity.readLocalVariable, RunnableEntity.writtenLocalVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accessedVariable	AutosarVariableRef	0..1	aggr	This denotes the accessed variable.
scope	VariableAccessScopeEnum	0..1	attr	This attribute allows for constraining the scope of the corresponding communication. For example, it possible to express whether the communication is intended to cross the boundary of an ECU or whether it is intended not to cross the boundary of a single partition.

Table 7.35: VariableAccess

[constr_1954] Existence of attribute VariableAccess.accessedVariable ⌈For each VariableAccess, attribute accessedVariable shall exist **at the time when the contract phase generation is executed**.⌋()

Enumeration	VariableAccessScopeEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements
Note	This enumeration defines scopes for communication.
Aggregated by	VariableAccess.scope
Literal	**Description**
communicationInterEcu	This case is foreseen to express that the corresponding communication shall be considered inter-ECU, i.e. it will cross the ECU boundary. This is considered the default case. **Tags:**atp.EnumerationLiteralIndex=0
communicationIntraPartition	This case is foreseen to express that the corresponding communication shall **not** cross the boundary of a partition. **Tags:**atp.EnumerationLiteralIndex=1
interPartitionIntraEcu	In this case the communication shall cross the boundaries of partitions within one ECU but it shall not cross the boundaries of the ECU itself. **Tags:**atp.EnumerationLiteralIndex=2

Table 7.36: VariableAccessScopeEnum

[TPS_SWCT_01326] Constrain the scope of a specific communication ⌈The purpose of the attribute scope of meta-class VariableAccess is to constrain the scope of the corresponding communication.

The main use-case for this ability is the development of a software-component where certain end-points of communication from or to the software-component are known to fulfill a certain constraint, e.g. execute within the same partition.⌋*(RS_SWCT_00200)*

[TPS_SWCT_01328] Default value of attribute scope ⌈The default value of attribute scope is set to communicationInterEcu.⌋*(RS_SWCT_00200)*

[constr_1141] Applicability of the scope attribute ⌈

The attribute scope of meta-class VariableAccess shall **only** be applied with respect to the aggregation of VariableAccess in the following roles:

- dataReadAccess
- dataWriteAccess
- dataSendPoint
- dataReceivePointByValue
- dataReceivePointByArgument

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

This aspect is depicted in Figure 7.21.

7.5.1.3 Explicit Sending and Receiving

[TPS_SWCT_01330] RunnableEntity can also have dataSendPoints ⌈A RunnableEntity can also have dataSendPoints (i.e. aggregate VariableAccess in the role dataSendPoint).

Using an instanceRef association, these eventually reference a VariableDataPrototype in the context of an AbstractProvidedPortPrototype, owned by the AtomicSwComponentType that is associated with the RunnableEntity that in turn owns the dataSendPoint.⌋*(RS_SWCT_00200)*

[constr_2004] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataSendPoint ⌈A VariableAccess in the role dataSendPoint shall refer to a PPortPrototype or PRPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01331] dataWriteAccess vs. dataSendPoint ⌈As opposed to the dataWriteAccess:

- Using the dataSendPoint, the RunnableEntity needs to explicitly “send” through an API; when using a dataWriteAccess, the RunnableEntity only needs to modify the value of certain variables.
- Using dataSendPoint, the Runnable can decide to “send” an arbitrary number of times; when using dataWriteAccess the new value of the VariableDataPrototype is only made available after the RunnableEntity terminates.
- The presence of a dataSendPoint per definition lets the corresponding RunnableEntity attain cat. 1B.

⌋*(RS_SWCT_00200)*

[constr_1773] Value of attribute dataSendPoint.returnValueProvision ⌈All RunnableEntity.dataSendPoint that refer to the same accessedVariable shall define the identical value for attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1773]: different RunnableEntitys could aggregate VariableAccess in the role dataSendPoint with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected VariableAccess aggregated in the role dataSendPoint agree on the configuration of attribute returnValueProvision.

This relation is sketched in Figure 7.22.

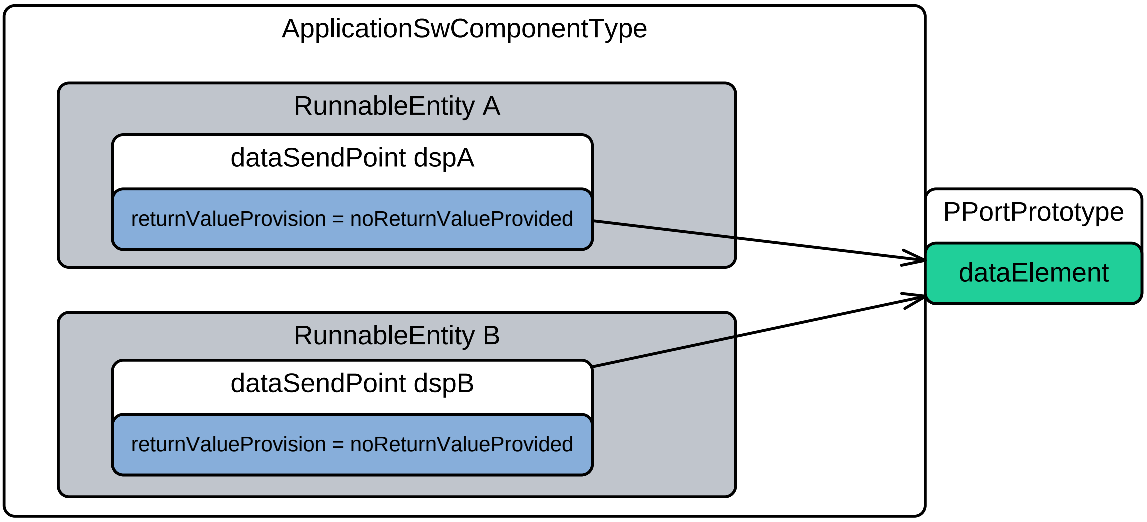


Figure 7.22: Modeling of attribute dataSendPoint.returnValueProvision

[TPS_SWCT_01663] dataReadAccess vs. dataReceivePointByValue or dataReceivePointByArgument ⌈As opposed to the dataReadAccess:

- Using the dataReceivePointByValue or dataReceivePointByArgument, the RunnableEntity always "receives" the latest value of the dataElement fore each call to the respective API during the execution of the RunnableEntity.
- When using a dataReadAccess, the value of the respective dataElement is received before the RunnableEntity starts and does not change during the execution of the RunnableEntity independently of the number of API calls for implicit reception.
- The presence of a dataReceivePointByValue or dataReceivePointByArgument per definition lets the corresponding RunnableEntity attain cat. 1B.

⌋*(RS_SWCT_00200)*

For more details, please refer to section 4.9.

[TPS_SWCT_01332] dataReceivePointByValue vs. dataReceivePointByArgument ⌈In analogy to explicitly sending data it is also possible to define explicit polling for new available data through a dataReceivePointByValue or dataReceivePointByArgument.⌋*()*

This aspect is visualized in Figure 7.24.

[constr_1277] SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue ⌈The SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue shall not be set to queued **at the time when the contract phase generation is executed**.⌋()

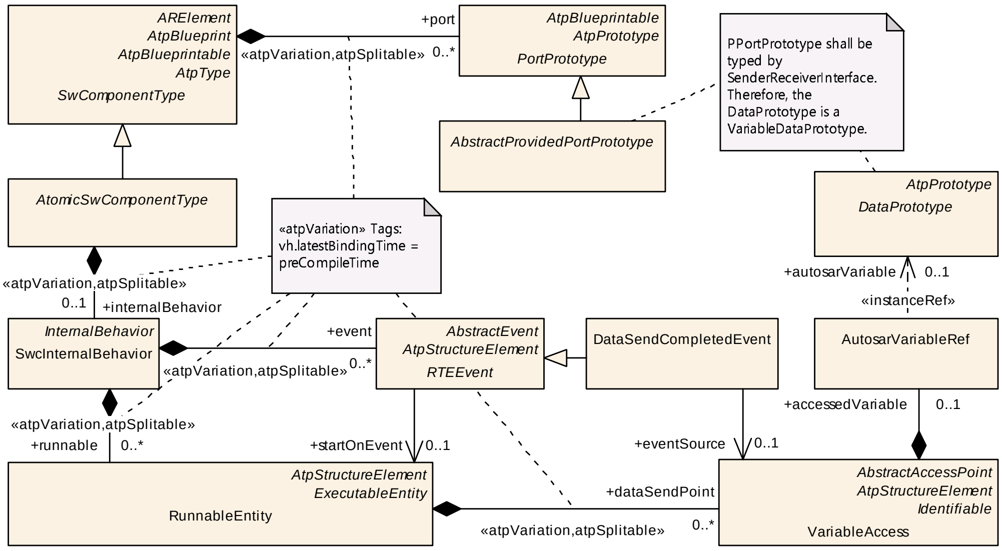


Figure 7.23: DataSendPoint

Rationale for [constr_1277]: when using the return value of the applicable RTE API function to return the value of a VariableDataPrototype there is no way[3] to provide an indication that the queue is empty. Therefore, the only safe approach is to not permit this scenario at all, hence the constraint.

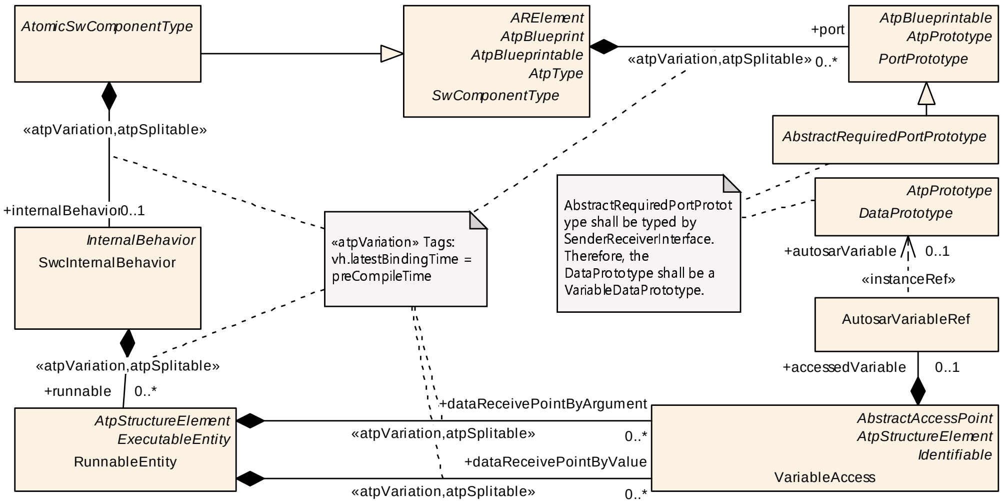


Figure 7.24: Definition of an explicit request to receive data

[TPS_SWCT_01333] **dataReceivePointByValue/dataReceivePointByArgument vs. dataReadAccess** ⌈By using a dataReceivePointByValue or dataReceivePointByArgument instead of dataReadAccess the constraining access to the referenced VariableDataPrototype (other RunnableEntitys shall not

[3] That is, other than to use a function argument to return the status of the queue but that would obviously beat the purpose of the API function.

change the VariableDataPrototype during the read execution) is limited to a short, well-defined amount of time.⌋*(RS_SWCT_00200)*

[TPS_SWCT_01334] RunnableEntitys of category 1 may have dataReceivePointByValues/dataReceivePointByArguments ⌈Therefore, category 1 RunnableEntitys may also have dataReceivePointByValues/dataReceivePointByArguments and consequently become RunnableEntitys of category 1B⌋*(RS_SWCT_00200)*

Please note that the categories of RunnableEntity are explained in section 7.2.4.4.

Similar to the dataReadAccess, constraints apply to the reference target of the AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument.

[constr_1774] Value of attribute dataReceivePointByArgument.returnValueProvision ⌈All RunnableEntity.dataReceivePointByArgument that refer to the same accessedVariable shall define the identical value for attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1774]: different RunnableEntitys could aggregate VariableAccess in the role dataReceivePointByArgument with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected VariableAccess aggregated in the role dataReceivePointByArgument agree on the configuration of attribute returnValueProvision.

This relation is exemplarily (for the case of dataSendPoint.returnValueProvision) sketched in Figure 7.22.

[constr_2005] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument ⌈A VariableAccess in the role dataReceivePointByValue or dataReceivePointByArgument shall refer to an RPortPrototype or PRPortPrototype that is typed by either a SenderReceiverInterface or an NvDataInterface **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01335] Combine dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint ⌈In general, it is possible to combine a dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint in the scope of a particular RunnableEntity.

This allows for a call to a blocking receive routine implemented by the RTE. The timeout attribute of meta-class WaitPoint can be used to specify the time until the blocking call expires.

But in case of non-queued communication it is **not supported** that a DataReceivedEvent is used in combination with a WaitPoint (see [constr_2021]). This contradicts the approach of the last-is-best semantics.⌋*(RS_SWCT_00200)*

[constr_2021] WaitPoint referencing a DataReceivedEvent can not be used for non-queued communication ⌈A WaitPoint referencing a DataReceivedEvent is permitted **if and only if** the swImplPolicy of the VariableDataPrototype referenced by this DataReceivedEvent is set to queued.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

Please note however, that in this case (in response to the presence of a WaitPoint) the RunnableEntity becomes category 2.

7.5.1.4 Implicit Sending and Receiving

Implicit sending and receiving aims at the optimization of computation effort for sender-receiver communication.

Instead of executing the full amount of functionality for each call to a send-API or receive-API the implicit communication only receives implicitly received values latest before the start of the execution of a RunnableEntity and sends implicitly sent values earliest after termination of the RunnableEntity.

[TPS_SWCT_01329] Access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles ⌈Please note that from the formal point of view access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles.

This means that dataReadAccess for a read-access while the write-access is defined by means of aggregating VariableAccess in the role dataWriteAccess.⌋*(RS_SWCT_00200)*

This aspect is depicted in Figure 7.19.

The following constraints apply to the reference target of the AutosarVariableRef of VariableAccess in role dataReadAccess or dataWriteAccess.

[constr_2002] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReadAccess ⌈A VariableAccess in the role dataReadAccess shall refer to an RPortPrototype or PRPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_2003] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataWriteAccess ⌈A VariableAccess in the role `dataWriteAccess` shall refer to a `PPortPrototype` or `PRPortPrototype` that is typed by either a `SenderReceiverInterface` or a `NvDataInterface`.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

By access with `VariableAccess` in the `dataReadAccess` role always the last value of the `VariableDataPrototype` buffered before the `RunnableEntity` starts will be read during the execution of the `RunnableEntity`.

It would therefore not make any sense to provide a queue of values for the purpose of accessing a `dataElement` in the role `dataReadAccess`.

[constr_2020] `dataReadAccess` can not be used for queued communication ⌈The `swImplPolicy` of the `VariableDataPrototype` referenced by a `VariableAccess` in role `dataReadAccess` shall **not** be set to `queued` **at the time when the contract phase generation is executed**.⌋*()*

[constr_1256] Acknowledgement feedback in n:1 writer case ⌈Within the scope of one `SwcInternalBehavior`, it is **not** allowed that two or more aggregated `RunnableEntity`s own either `dataSendPoint`s or `dataWriteAccess`s that in turn point to the identical `accessedVariable.autosarVariable.targetDataPrototype` **if** the attribute `transmissionAcknowledge` exists in the context of the `SenderComSpec` owned by the `dataSendPoint.accessedVariable.autosarVariable.portPrototype` (or the respective construct for `dataWriteAccess`) that also refers to said `dataElement`.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

The background of [constr_1256] is that if two or more `RunnableEntity`s exist that can write to the identical `dataElement` it may happen that more than one `RunnableEntity` actually write to the respective `dataElement` **before** the "first" acknowledgement is received. In this case it will never be possible to determine exactly which transmission has been acknowledged.

The difference between implicit and explicit sender/receiver communication is explained in [TPS_SWCT_01331] and [TPS_SWCT_01663].

7.5.1.5 DataSendCompletedEvent

[TPS_SWCT_01336] `dataSendPoint` also allows for the definition of a `DataSendCompletedEvent` ⌈The `dataSendPoint` also allows for the definition of a `DataSendCompletedEvent`. This `RTEEvent` occurs when the data has been successfully sent or when an error has occurred during sending.⌋*(RS_SWCT_00200)*

Please note that this feature can only be used if the `AtomicSwComponentType` describes the meaning of success or failure of the send-operation.

In particular, via a SenderComSpec class different acknowledgement requests (in this case: successful transmission) can be attached to a PPortPrototype or PRPort-Prototype, as is shown in Figure 4.38.

This will configure the RTE such that when data is sent the RTE will try to obtain the specified acknowledgement; possibly by waiting a certain timeout period.

[constr_2033] Timeout of DataSendCompletedEvent ⌈The timeout value of a WaitPoint associated with a DataSendCompletedEvent shall have the same value as the corresponding value of TransmissionAcknowledgementRequest.timeout **at the time when the RTE is generated**.⌋*()*

7.5.1.6 DataWriteCompletedEvent

[TPS_SWCT_01557] dataWriteAccess also allows for the definition of a DataWriteCompletedEvent ⌈The dataWriteAccess also allows for the definition of a DataWriteCompletedEvent. This RTEEvent occurs when the data has been successfully sent or when an error has occurred during sending.⌋*(RS_SWCT_00200)*

Please note that this feature can only be used if the AtomicSwComponentType describes the meaning of success or failure of the send-operation.

In particular, via a SenderComSpec class different acknowledgement requests (in this case: successful transmission) can be attached to a PPortPrototype or PRPort-Prototype, as is shown in Figure 4.38.

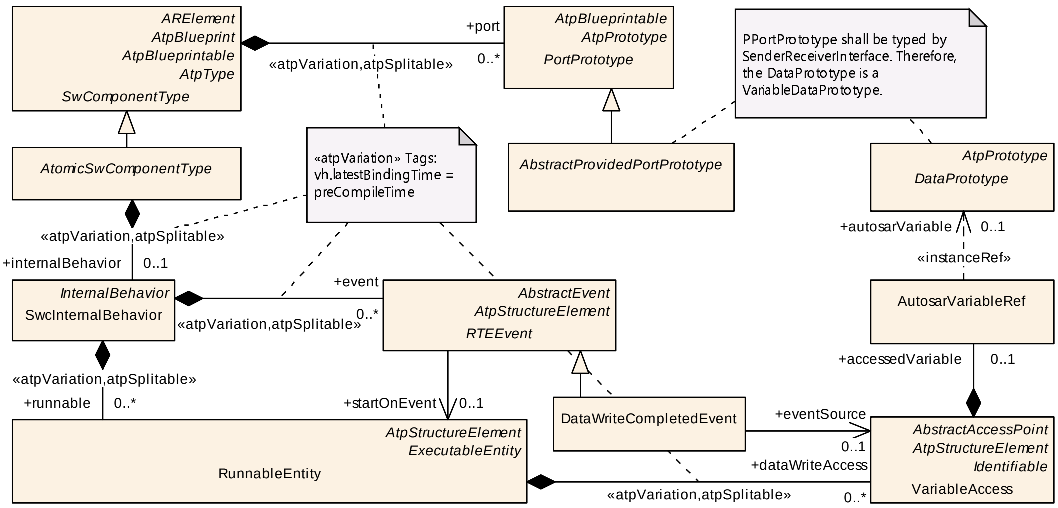


Figure 7.25: DataWriteAccess

[TPS_SWCT_01558] DataWriteCompletedEvent cannot be combined with a WaitPoint ⌈Please note that a DataWriteCompletedEvent cannot be associated with a WaitPoint, see [constr_1091].⌋*(RS_SWCT_00200)*

However, it is possible to configure the RTE such that when data is sent, the RTE will try to obtain the specified acknowledgement; possibly by waiting a certain `timeout` period.

7.5.1.7 DataReceivedEvent

[TPS_SWCT_01337] **`DataReceivedEvent`** ⌈A receiver is notified through the same event mechanism when a `VariableDataPrototype` is received. The `DataReceivedEvent` is directly associated with the corresponding `VariableDataPrototype`.⌋*(RS_SWCT_00200)*

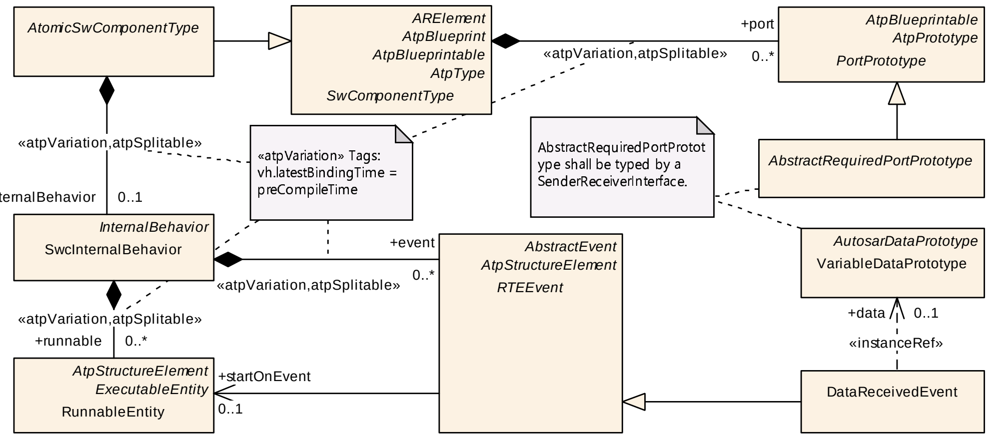


Figure 7.26: Receiver is notified by an event when new data has arrived

7.5.1.8 DataReceiveErrorEvent

[TPS_SWCT_01338] **`DataReceiveErrorEvent`** ⌈A receiver is notified of `DataReceiveErrorEvent` through the activation of its `RunnableEntity` which is referenced by this `RTEEvent`.

A `DataReceiveErrorEvent` includes a reference to a `VariableDataPrototype` and is raised by the RTE when an error concerning the reception of the referenced data is detected by the COM [4] layer.

The following cases present some situations which will cause the RTE to raise a `DataReceiveErrorEvent`:

- the RTE receives a signal-outdated notification from the COM layer when a monitored periodic signal is not received in time. The COM layer monitors the validity of the signal's value based on the value of the `aliveTimeout` attribute of `ReceiverComSpec` referencing the `VariableDataPrototype` associated with

[4]In case of internal communication the RTE is not enforced to use the COM layer. It is also possible to implement the required behavior directly in the RTE.

the signal. If the time elapsed since the last update of a signal's value exceeds its aliveTimeout then the COM layer notifies the RTE of a signal outdated error.

- The RTE receives a signal invalid notification from the COM layer when the COM layer detects that an incoming signal has the predefined “invalid” value.

⌋*(RS_SWCT_00200)*

[constr_10073] Existence of DataReceiveErrorEvent ⌈A DataReceiveErrorEvent shall only exist if it latest **at the time when the contract phase generation is executed** refers to a given VariableDataPrototype in the role data where either

- the VariableDataPrototype is referenced from a NonqueuedReceiverComSpec in the role dataElement and the attribute aliveTimeout of the NonqueuedReceiverComSpec exists and is set to a value > 0 **or**
- the VariableDataPrototype is aggregated by a SenderReceiverInterface where attribute invalidationPolicy.handleInvalid exists and is set to the value keep.

⌋*()*

[TPS_SWCT_01339] RTE activates RunnableEntity in response to DataReceiveErrorEvent ⌈A DataReceiveErrorEvent is used by the RTE to activate a RunnableEntity that is supposed to handle the above-mentioned errors.

The error code will be made available to the activated RunnableEntity through the appropriate RTE API function.⌋*(RS_SWCT_00200)*

[TPS_SWCT_01340] DataReceiveErrorEvent cannot be combined with a WaitPoint ⌈Please note that a DataReceiveErrorEvent cannot be associated with a WaitPoint, see [constr_1091].

It can only be used for the receiver software-component in a sender-receiver communication and its data reference is restricted to VariableDataPrototypes with their swImplPolicy attribute not set to queued.⌋*(RS_SWCT_00200)*

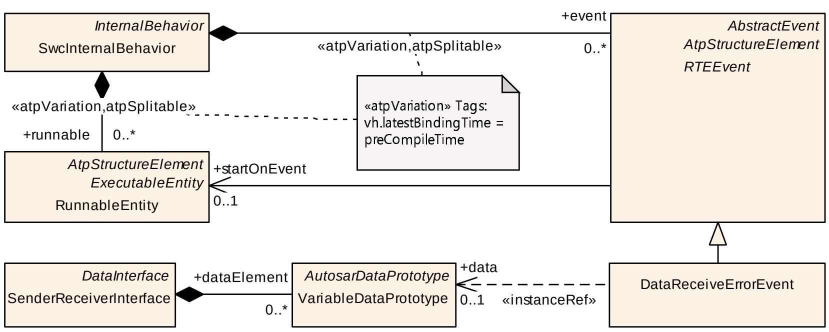


Figure 7.27: DataReceiveErrorEvent references a Runnable and a VariableDataPrototype

[TPS_SWCT_01341] DataReceiveErrorEvent is directly associated with the corresponding VariableDataPrototype ⌈The DataReceiveErrorEvent is directly associated with the corresponding VariableDataPrototype and references the RunnableEntity that is activated due to the occurrence of this RTEEvent.⌋*(RS_SWCT_00200)*

This aspect is depicted in Figure 7.27.

7.5.2 RunnableEntities and Client Server Communication

7.5.2.1 Invoking an Operation

[TPS_SWCT_01342] Invocation of a server operation ⌈A RunnableEntity invokes a server operation formally defined as a ClientServerOperation via an RPortPrototype of the enclosing SwComponentPrototype typed by a particular AtomicSwComponentType.⌋*(RS_SWCT_00200)*

[TPS_SWCT_01343] Synchronous vs. asynchronous invocation ⌈A ClientServerOperation itself can be invoked either "synchronously" or "asynchronously".⌋*(RS_SWCT_00200)*

In the majority of cases the ClientServerOperation will be invoked at a different SwComponentPrototype but in general it would be possible to invoke a ClientServerOperation on the same SwComponentPrototype as well.

The decision whether a specific ClientServerOperation is called synchronously or asynchronously needs to be specified in the formal description of the corresponding AtomicSwComponentType, namely in the context of an SwcInternalBehavior (see Figure 7.28 for more details).

But it is not supported to invoke the same instance of a ClientServerOperation synchronously and asynchronously together.

[constr_2022] Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints ⌈A ClientServerOperation of a particular RPortPrototype shall be mutually exclusive referenced by either a SynchronousServerCallPoints or an AsynchronousServerCallPoints **at the time when the contract phase generation is executed**.⌋*()*

[constr_1775] Value of attribute serverCallPoint.returnValueProvision ⌈All RunnableEntity.serverCallPoint that refer to the same operation shall define the identical value of attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1775]: different RunnableEntitys could aggregate ServerCallPoint with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected ServerCallPoints agree on the configuration of attribute returnValueProvision.

This relation is exemplarily (for the case of dataSendPoint.returnValueProvision) sketched in Figure 7.22.

[constr_2023] Consistency of timeout values ⌈The timeout values of all ServerCallPoints referencing the same instance of ClientServerOperation in a RPortPrototype shall be identical **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01345] Synchronous operation invocation ⌈In case of a synchronous operation invocation the particular RunnableEntity merely needs a SynchronousServerCallPoint.⌋*(RS_SWCT_00200)*

More information can be found in Figure 7.28.

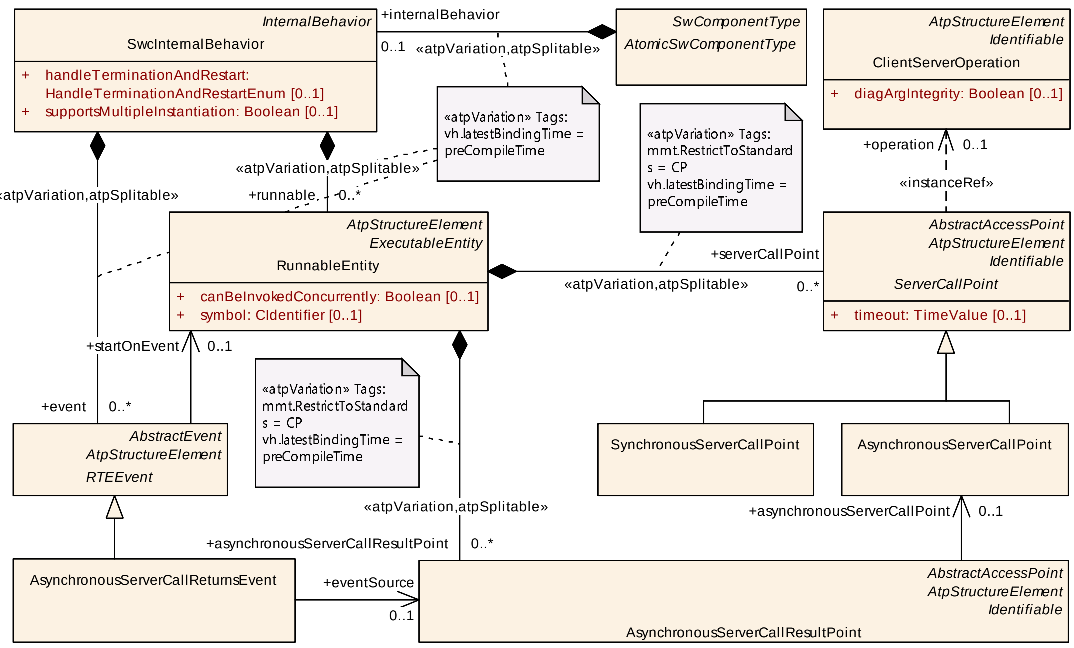


Figure 7.28: Model of a server call point.

[TPS_SWCT_01346] Asynchronous operation invocation ⌈Asynchronous invocation is a bit more complex because it is necessary to specify how to respond to a notification about the completion of the corresponding operation.

This is done using the generic RTEEvent mechanism: the notification about an asynchronously executed operation having completed is implemented as an AsynchronousServerCallReturnsEvent.

Therefore, if an AsynchronousServerCallReturnsEvent is raised the RTE can either trigger the execution of a specific RunnableEntity or the AtomicSwComponentType can implement a WaitPoint that blocks the execution of the calling

RunnableEntity until the AsynchronousServerCallReturnsEvent is recognized.⌋(RS_SWCT_00200)

For example, let's consider the case of an asynchronous call to a remote operation where the RTE is supposed to trigger a specific RunnableEntity when the operation completes. The description of the corresponding AtomicSwComponentType would typically contain the following elements:

1. The AtomicSwComponentType contains an RPortPrototype 'myPort' typed by a PortInterface that in turn contains the definition of an ClientServerOperation 'remoteOperation'.

2. The AtomicSwComponentType's SwcInternalBehavior contains at least two RunnableEntitys: the RunnableEntity 'main' is supposed to invoke the operation; the RunnableEntity 'callback' is the one that should be called when the operation completes.

3. The description of the RunnableEntity 'main' contains an AsynchronousServerCallPoint 'invokeMyOperation' referencing the respective ClientServerOperation in the PortInterface used to type the PortPrototype 'myPort'. This implies that the RunnableEntity is allowed to invoke this operation asynchronously.

4. The description of the RunnableEntity 'callback' contains an AsynchronousServerCallResultPoint 'fetchMyOperationResults' referencing the respective AsynchronousServerCallPoint 'invokeMyOperation'. This implies that the RunnableEntity is allowed to fetch the results of the asynchronously invoked operation.

5. The description of the SwcInternalBehavior includes an AsynchronousServerCallReturnsEvent 'myOperationReturns' which references the previously defined AsynchronousServerCallResultPoint 'fetchMyOperationResults'

6. The description of the AsynchronousServerCallReturnsEvent 'myOperationReturns' references the RunnableEntity 'callback', indicating that the RTE should trigger the execution of this Runnable when 'myOperationReturns' is raised.

[constr_1776] Value of attribute asynchronousServerCallResultPoint.returnValueProvision ⌈All RunnableEntity.asynchronousServerCallResultPoint that refer to the same AsynchronousServerCallPoint.operation shall define the identical value of attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋()

Rationale for the existence of [constr_1776]: different RunnableEntitys could aggregate AsynchronousServerCallResultPoint with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected AsynchronousServerCallResultPoints agree on the configuration of attribute returnValueProvision.

This relation is exemplarily (for the case of dataSendPoint.returnValueProvision) sketched in Figure 7.22.

Class	***ServerCallPoint*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a ServerCallPoint it is entitled to invoke a particular ClientServerOperation of a specific RPortPrototype of the corresponding AtomicSwComponentType			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	AsynchronousServerCallPoint, SynchronousServerCallPoint			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.serverCallPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operation	ClientServerOperation	0..1	iref	The operation that is called by this runnable. **InstanceRef implemented by:**ROperationInAtomicSwcInstanceRef
timeout	TimeValue	0..1	attr	Time in seconds before the server call times out and returns with an error message. It depends on the call type (synchronous or asynchronous) how this is reported.

Table 7.37: ServerCallPoint

[constr_1955] Existence of attribute ServerCallPoint.operation ⌈For each ServerCallPoint, attribute operation shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1956] Existence of attribute ServerCallPoint.timeout ⌈For each ServerCallPoint, attribute timeout shall exist **at the time when the RTE is generated**.⌋*()*

Class	**SynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	This means that the RunnableEntity is supposed to perform a blocking wait for a response from the server.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServerCallPoint*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.serverCallPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
calledFromWithinExclusiveArea	ExclusiveAreaNestingOrder	0..1	ref	This indicates that the call point is located at the deepest level inside one or more ExclusiveAreas that are nested in the given order.

Table 7.38: SynchronousServerCallPoint

[constr_1957] Existence of attribute AsynchronousServerCallResultPoint.asynchronousServerCallPoint ⌈For each AsynchronousServerCallResultPoint, the reference to AsynchronousServerCallPoint in the role asynchronousServerCallPoint shall exist **at the time when the contract phase generation is executed**.⌋*()*

Class	AsynchronousServerCallPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	An AsynchronousServerCallPoint is used for asynchronous invocation of a ClientServerOperation. IMPORTANT: a ServerCallPoint cannot be used concurrently. Once the client RunnableEntity has made the invocation, the ServerCallPoint cannot be used until the call returns (or an error occurs!) at which point the ServerCallPoint becomes available again.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServerCallPoint*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.serverCallPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.39: AsynchronousServerCallPoint

Class	AsynchronousServerCallResultPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a AsynchronousServerCallResultPoint it is entitled to get the result of the referenced AsynchronousServerCallPoint. If it is associated with AsynchronousServerCallReturnsEvent, this RTEEvent notifies the completion of the required ClientServerOperation or a timeout. The occurrence of this event can either unblock a WaitPoint or can lead to the invocation of a RunnableEntity.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.asynchronousServerCallResultPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
asynchronousServerCallPoint	AsynchronousServerCallPoint	0..1	ref	The referenced Asynchronous Server Call Point defines the asynchronous server call from which the results are returned.

Table 7.40: AsynchronousServerCallResultPoint

[constr_2006] Number of AsynchronousServerCallResultPoint referencing to one AsynchronousServerCallPoint ⌈The AsynchronousServerCallPoint may be referenced by at most one AsynchronousServerCallResultPoint **at the time when the contract phase generation is executed**.

If the reference exists, this means that only the RunnableEntity with this AsynchronousServerCallResultPoint can fetch the result of the asynchronous server invocation of this particular AsynchronousServerCallPoint.⌋*()*

Please note that if an AsynchronousServerCallPoint is **not** referenced by an AsynchronousServerCallResultPoint this means that there is no operation result to fetch or the caller **is not interested** in the result.

This information might be used by the RTE generator to optimize the data consistency mechanisms.

[TPS_SWCT_01347] Blocking access to operation result in an asynchronous operation invocation ⌈If the call of the RTE fetching the operations results shall block until the server returns, the RunnableEntity with the AsynchronousServerCallResultPoint needs additional a WaitPoint referencing the AsynchronousServerCallReturnsEvent which is associated with the AsynchronousServerCallResultPoint representing the operations results access.

In this case the AsynchronousServerCallReturnsEvent shall not define a startOnEvent reference to a RunnableEntity.⌋*(RS_SWCT_00200)*

[constr_2030] AsynchronousServerCallResultPoint combined with WaitPoint shall belong to the same RunnableEntity ⌈A WaitPoint referencing a AsynchronousServerCallReturnsEvent as well as a AsynchronousServerCallResultPoint referenced by said AsynchronousServerCallReturnsEvent shall be aggregated by the same RunnableEntity **at the time when the contract phase generation is executed**.⌋*()*

[constr_1521] Reference from AsynchronousServerCallReturnsEvent to AsynchronousServerCallResultPoint ⌈In the context of a RunnableEntity, a given AsynchronousServerCallResultPoint shall only be referenced by one AsynchronousServerCallReturnsEvent in the role eventSource **at the time when the contract phase generation is executed**.⌋*()*

7.5.2.2 Providing an Implementation of an Operation

A software-component can define an OperationInvokedEvent for each operation inside one of the server AbstractProvidedPortPrototypes. This way a RunnableEntity may respond to such an invocation through the generic event handling mechanisms described above (as formally expressed in Figure 7.29).

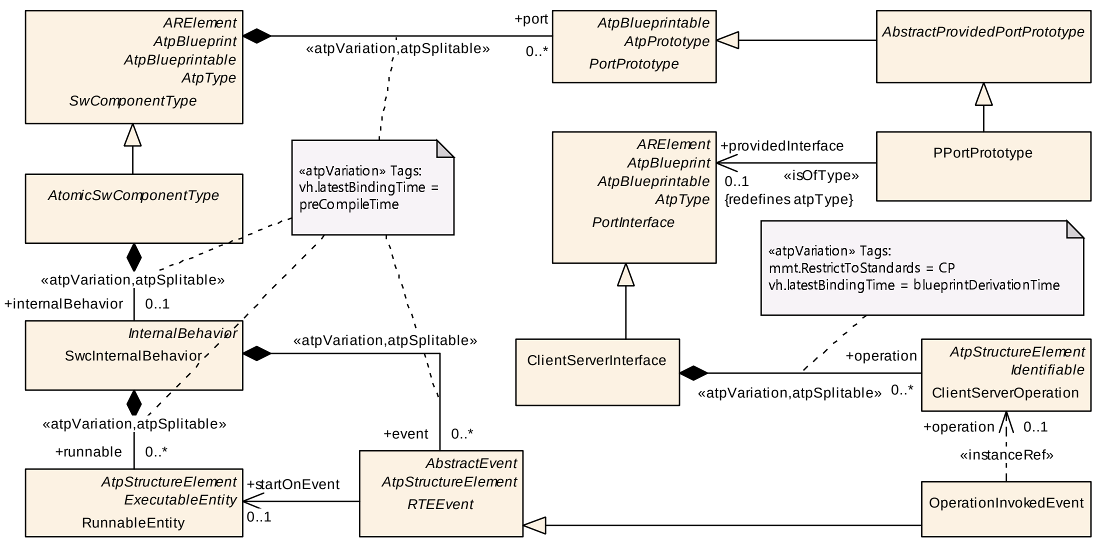


Figure 7.29: The OperationInvokedEvent references the operation that was called by a client.

7.5.2.3 Reacting on Data Transformation Errors

[TPS_SWCT_01624] Hard error occurs during the execution of a transformer chain ⌈If a hard error occurs during the execution of a transformer chain which is executed

- on the server side of a client/server communication and re-transforms the data which trigger a server RunnableEntity **or**
- on the trigger sink side of an inter-ECU external trigger communication,

this server RunnableEntity or trigger sink RunnableEntity cannot be started because the re-transformed data are not available.⌋*(RS_SWCT_03222)*

This might be a problem for the software-component if the software-component wants to react on transformer errors.

[TPS_SWCT_01616] Semantics of TransformerHardErrorEvent ⌈A software-component can define a TransformerHardErrorEvent

- for each ClientServerOperation inside one of the server PPortPrototypes (i.e. typed by a ClientServerInterface) or
- for each Trigger in trigger sink RPortPrototypes (i.e. typed by a TriggerInterface).

This way, a given RunnableEntity may define its response to a transformer error.⌋*(RS_SWCT_03222)*

7.5.3 RunnableEntities and External Trigger Event Communication

7.5.3.1 Trigger Source

[TPS_SWCT_01348] Trigger source ⌈A RunnableEntity of the triggering software-component raises an external trigger event via an AbstractProvidedPortPrototype of the enclosing SwComponentPrototype typed by a particular AtomicSwComponentType.

For this purpose the particular RunnableEntity needs an ExternalTriggeringPoint that references the particular instance of the trigger in a PPortPrototype.⌋*(RS_SWCT_00200)*

[constr_1777] Value of attribute externalTriggeringPoint.returnValueProvision ⌈All RunnableEntity.externalTriggeringPoint that refer to the same trigger shall define the identical value of attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1777]: different RunnableEntitys could aggregate ExternalTriggeringPoints with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected ExternalTriggeringPoints agree on the configuration of attribute returnValueProvision.

This relation is exemplarily (for the case of dataSendPoint.returnValueProvision) sketched in Figure 7.22.

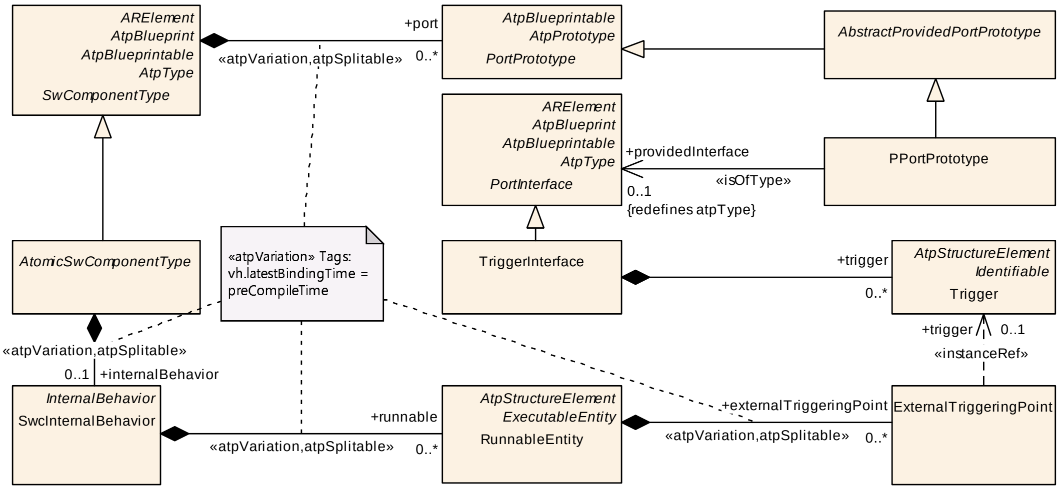


Figure 7.30: Model structure of a trigger source.

Class	**ExternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns an ExternalTriggeringPoint it is entitled to raise an ExternalTriggerOccurred Event.			
Base	*ARObject*			
Aggregated by	RunnableEntity.externalTriggeringPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
ident	ExternalTriggeringPointIdent	0..1	aggr	The aggregation in the role ident provides the ability to make the ExternalTriggeringPoint identifiable. From the semantical point of view, the ExternalTriggering Point is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-100
trigger	Trigger	0..1	iref	The trigger taken for the ExternalTriggeringPoint. **Tags:** xml.namePlural=TRIGGER-IREF xml.roleElement=false xml.roleWrapperElement=true xml.typeElement=true xml.typeWrapperElement=false **InstanceRef implemented by:**PTriggerInAtomicSwcTypeInstanceRef

Table 7.41: ExternalTriggeringPoint

7.5.3.2 Trigger Sink

The activation of RunnableEntitys in the trigger sink is effected through the generic event handling mechanism.

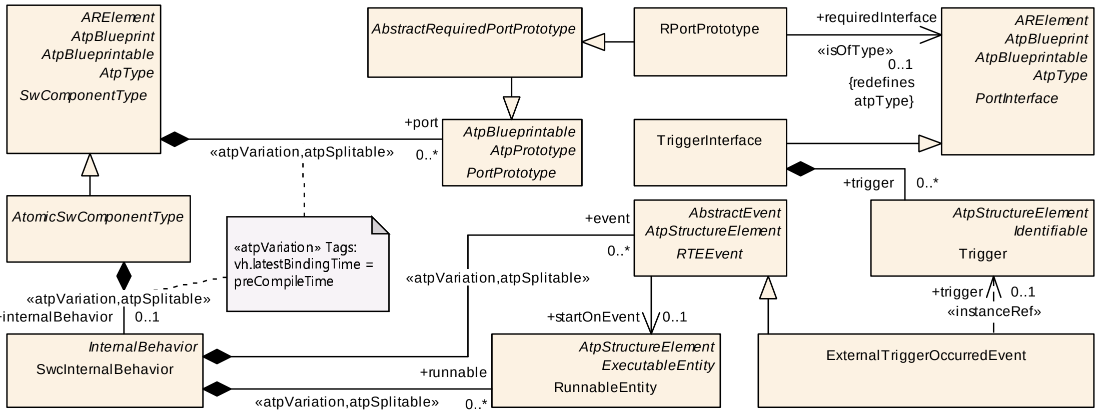


Figure 7.31: Model structure of a trigger sink

[TPS_SWCT_01349] Trigger sink ⌈The fact that a RunnableEntity shall be activated on occurrence of an external trigger event is formally defined by means of ExternalTriggerOccurredEvent that references a particular instance of the trigger in a RPortPrototype and additionally the RunnableEntity to be executed in response to the event.⌋*(RS_SWCT_00200)*

7.5.4 RunnableEntities and Parameter Access

There are several ways a Calibration Parameter is provided within a software component.

[TPS_SWCT_01350] Calibration Parameters shared among several SwComponentTypes ⌈As mentioned above, if Calibration Parameters are shared among several SwComponentTypes a dedicated PortInterface in a PortPrototype will be used.⌋*(RS_SWCT_00200)*

The designer of a software-component can use this access mechanism when designing a RunnableEntity using, as input value, a DataPrototype

- from an arbitrary RPortPrototype associated with a ClientServerInterface, SenderReceiverInterface or a NvDataInterface,
- VariableDataPrototype in the context of an SwcInternalBehavior

This input value will be fed to an interpolation routine whose result can be used internally or transferred to an adjacent SwComponentPrototype via dedicated PortPrototypes.

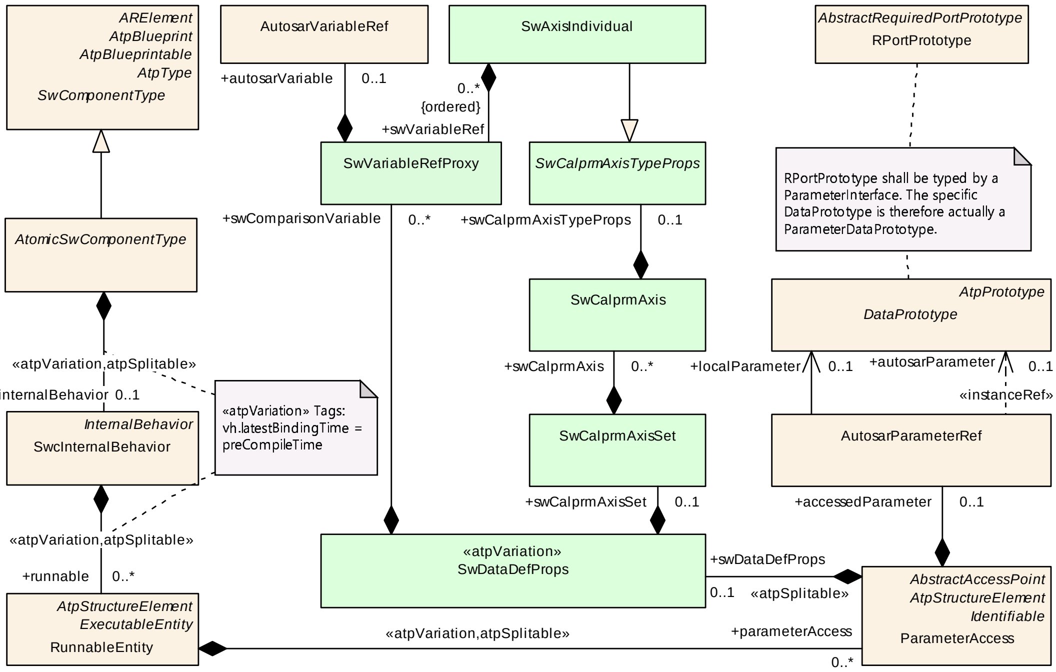


Figure 7.32: Runnable Access to a Calibration Port

Typically, there will be a dedicated RunnableEntity (with “ReceiveMode” set to “activation_of_runnable_entity”) that itself calls the interpolation routine with the appropriate input value and the appropriate ParameterDataPrototype.

Note that the ParameterAccess also allows to set input values or shared axis through SwDataDefProps which are specific to the access point.

The result of this interpolation routine call is provided as an ArgumentDataPrototype with direction being either set to out or inout in a ClientServerInterface.

Class	**ParameterAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a ParameterAccess implies that a RunnableEntity needs access to a ParameterData Prototype.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.parameterAccess			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
accessed Parameter	AutosarParameterRef	0..1	aggr	Reference to the accessed calibration parameter.
swDataDef Props	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps

Table 7.42: ParameterAccess

[constr_1958] Existence of attribute ParameterAccess.accessedParameter
⌈For each ParameterAccess, attribute accessedParameter shall exist **at the time when the contract phase generation is executed.**⌋*()*

[TPS_SWCT_01351] Access to a ParameterDataPrototype ⌈The access to a ParameterDataPrototype will be indicated

- by the ParameterAccess entity if the RunnableEntity wants to access it from a RPortPrototype.
- by defining the ParameterAccess association from a RunnableEntity to the ParameterDataPrototype in the roles sharedParameter or perInstanceParameter.

⌋*(RS_SWCT_00200)*

Please find more information about the topic of [TPS_SWCT_01351] in Figure 7.32 as well as in Figure 2.3 in the lower association from RunnableEntity to ParameterDataPrototype

Note: A ParameterDataPrototype in the roles constantMemory is not provided by the RTE and therefore the ParameterAccess association is not required to control the RTE API generation.

7.5.4.1 InstantiationDataDefProps

Typically, the accessibility and further information like alias names for a particular piece of data is modeled on the level of DataPrototypes (especially VariableDataPrototypes, ParameterDataPrototypes).

But due to the recursive structure of the meta-model concerning data types (an ApplicationCompositeDataType consists of DataPrototypes), a part of the relevant MCD information is described directly in the data type (in case of a ApplicationCompositeDataType).

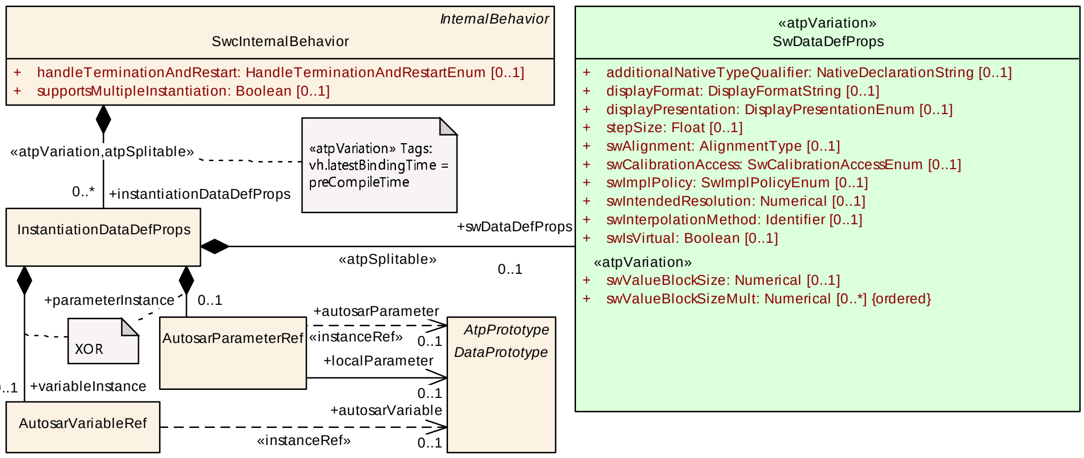


Figure 7.33: applying instantiation specific data definition properties

This is a strong restriction in the re-use of data types because the ApplicationCompositeDataType should be re-used for different VariableDataPrototypes and ParameterDataPrototypes to guarantee type compatibility on C-implementation level (e.g. data of a PortPrototype is stored in a PIM or a ParameterDataPrototype used as ROM Block and shall be typed by the same data type as NVRAM Block).

This restriction is overcome by InstantiationDataDefProps as shown in figure 7.33

Class	**InstantiationDataDefProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::InstantiationDataDefProps			
Note	This is a general class allowing to apply additional SwDataDefProps to particular instantiations of a Data Prototype. Typically the accessibility and further information like alias names for a particular data is modeled on the level of DataPrototypes (especially VariableDataPrototypes, ParameterDataPrototypes). But due to the recursive structure of the meta-model concerning data types (a composite (data) type consists out of data prototypes) a part of the MCD information is described in the data type (in case of Application CompositeDataType). This is a strong restriction in the reuse of data typed because the data type should be re-used for different VariableDataPrototypes and ParameterDataPrototypes to guarantee type compatibility on C-implementation level (e.g. data of a Port is stored in PIM or a ParameterDataPrototype used as ROM Block and shall be typed by the same data type as NVRAM Block). This class overcomes such a restriction if applied properly.			
Base	*ARObject*			
Aggregated by	NvBlockDescriptor.instantiationDataDefProps, ParameterSwComponentType.instantiationDataDefProps, SwcInternalBehavior.instantiationDataDefProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
parameter Instance	AutosarParameterRef	0..1	aggr	This reference identifies the particular DataPrototype (defined in the context of a composite ParameterData Prototype) on which the swDataDefProps shall be applied.
swDataDef Props	SwDataDefProps	0..1	aggr	These are the particular data definition properties which shall be applied **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps
variableInstance	AutosarVariableRef	0..1	aggr	This reference identifies the particular DataPrototype (defined in the context of a composite VariableData Prototype) on which the swDataDefProps shall be applied.

Table 7.43: InstantiationDataDefProps

[constr_1959] Existence of attribute InstantiationDataDefProps.swDataDefProps ⌈For each InstantiationDataDefProps, attribute swDataDefProps shall exist **at the time when the contract phase generation is executed.**⌋*()*

Documented use cases for the application of InstantiationDataDefProps can be found in [TPS_SWCT_01846], [TPS_SWCT_01847], and [TPS_SWCT_01848].

7.5.5 RunnableEntities and Mode Communication

For the communication of modes between RunnableEntitys we have to distinguish between two use cases.

[TPS_SWCT_01352] Requested mode is just sent and received as an ordinary data value ⌈In the first case, a requested mode is just sent and received as an ordinary data value without specifying the details of mode switching in the corresponding port interface.

This mechanism is used if the receiving RunnableEntity is not directly implementing a mode switch but does further processing of the mode request. This is especially needed to transfer mode requests between ECUs.

In this case, the mode is transferred via sender-receiver communication so that the involved RunnableEntitys just need the same type of APIs against the RTE as for sender-receiver communication.

This is possible, because ModeDeclarationGroupPrototypes can be mapped to an ImplementationDataTypes.⌋*(RS_SWCT_00200)*

This concept and the meta-classes needed for the mapping are further explained in chapter 4.2.5.

[TPS_SWCT_01353] RunnableEntitys react on a mode request via a corresponding RTEEvent ⌈In the second case, one RunnableEntity "sends" a mode request and one or more other RunnableEntitys react on the request via a corresponding RTEEvent or by being suppressed from being triggered any longer by other RTEEvents.

In this case, special APIs against the RTE are required and the RTE has to implement the actual mode switch. This kind of communication is only possible between software-components on the same ECU.⌋*(RS_SWCT_00200, RS_SWCT_03202)*

For further explanation of the general concept refer to chapter 4.2.5 and for the details of the meta-model for mode switches refer to chapter 9.

7.6 Port API Options

[TPS_SWCT_01354] PortAPIOption ⌈The RTE Generator needs additional options per PortPrototype to choose the proper generation schema. These are subsumed in the PortAPIOption element.⌋*(RS_SWCT_03040)*

Please note that meta-class PortAPIOption is depicted in Figure 7.34.

Class	PortAPIOption			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	Options how to generate the signatures of calls for an AtomicSwComponentType in order to communicate over a PortPrototype (for calls into a RunnableEntity as well as for calls from a Runnable Entity to the PortPrototype).			
Base	*ARObject*			
Aggregated by	SwcInternalBehavior.portAPIOption			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
enableTake Address	Boolean	0..1	attr	If set to true, the software-component is able to use the API reference for deriving a pointer to an object.
errorHandling	DataTransformation ErrorHandlingEnum	0..1	attr	This specifies whether a RunnableEntity accessing a Port Prototype that is referenced by this PortAPIOption shall specifically handle transformer errors or not.
indirectAPI	Boolean	0..1	attr	If set to true this attribute specifies an "indirect API" to be generated for the associated port which means that the software-component is able to access the actions on a port via a pointer to an object representing a port. This allows e.g. iterating over ports in a loop. This option has no effect for PPortPrototypes of client/server interfaces.
port	PortPrototype	0..1	ref	The option is valid for generated functions related to communication over this port
portArgValue (ordered)	PortDefinedArgument Value	*	aggr	An argument value defined by this port.
supported Feature	SwcSupportedFeature	*	aggr	This collection specifies which features are supported by the RunnableEntitys which access a PortPrototype that it referenced by this PortAPIOption.
transformer Status Forwarding	DataTransformation StatusForwardingEnum	0..1	attr	This attribute specifies whether a RunnableEntity accessing a PortPrototype that is referenced by this Port APIOption shall be able to forward a status code to the transformer chain.

Table 7.44: PortAPIOption

Enumeration	DataTransformationErrorHandlingEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions
Note	This enumeration defines different ways how a RunnableEntity shall handle transformer errors.
Aggregated by	PortAPIOption.errorHandling
Literal	***Description***
noTransformerError Handling	A runnable does not handle transformer errors. **Tags:**atp.EnumerationLiteralIndex=0
transformerError Handling	The runnable implements the handling of transformer errors. **Tags:**atp.EnumerationLiteralIndex=1

Table 7.45: DataTransformationErrorHandlingEnum

[constr_1960] Existence of attribute PortAPIOption.port ⌈For each PortAPIOption, attribute port shall exist **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01626] Error notification of data transformer errors ⌈If the attribute PortAPIOption.errorHandling is set to transformerErrorHandling then all RunnableEntitys accessing the PortPrototype referenced by port shall handle the extended transformer error notification.⌋*(RS_SWCT_03222, RS_SWCT_03221)*

[TPS_SWCT_03500] Status forwarding to data transformer ⌈If the attribute PortAPIOption.transformerStatusForwarding is set to transformerStatusForwarding, then all RunnableEntitys accessing the PortPrototype referenced in the role PortAPIOption.port shall provide the argument named forwardedCode to the transformer chain.⌋*(RS_SWCT_03040, RS_SWCT_03221)*

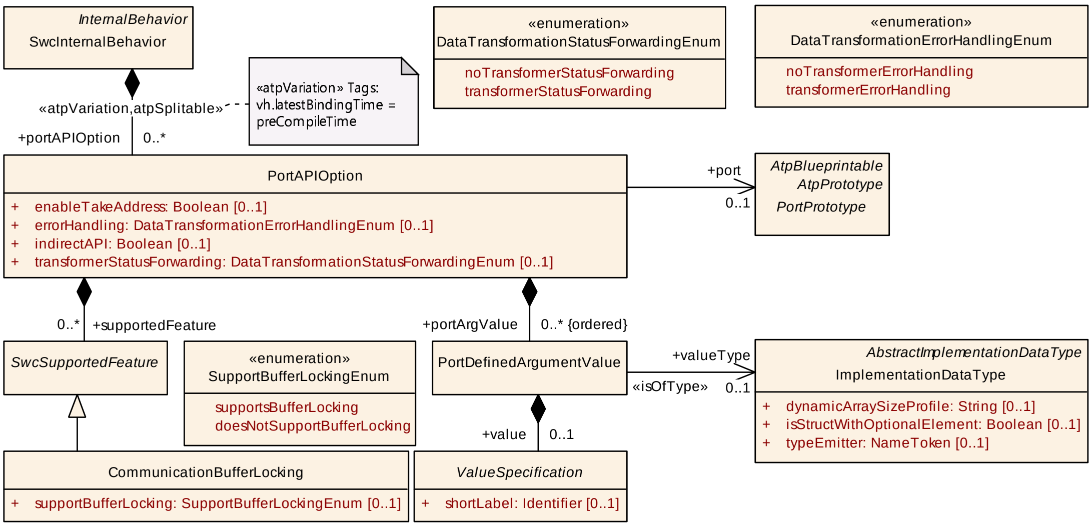


Figure 7.34: Port API Options.

[TPS_SWCT_03501] Applicability of status forwarding to data transformer ⌈The attribute PortAPIOption.transformerStatusForwarding shall only be set to transformerStatusForwarding if the transported data is originating from a transformed source and is not altered.⌋*(RS_SWCT_03040, RS_SWCT_03221)*

Enumeration	**DataTransformationStatusForwardingEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions
Note	This enumeration defines different ways how a RunnableEntity shall be able to forward status code into the transformer chain.
Aggregated by	PortAPIOption.transformerStatusForwarding
Literal	**_Description_**
noTransformer StatusForwarding	The RunnableEntity is not able to forward a transformer status code. **Tags:**atp.EnumerationLiteralIndex=0
transformerStatus Forwarding	The RunnableEntity is able to forward a transformer status code. **Tags:**atp.EnumerationLiteralIndex=1

Table 7.46: DataTransformationStatusForwardingEnum

The restriction of [TPS_SWCT_03501] is especially necessary in case of E2E protected data routing where the source of the data is received with a *transformer error* using PortAPIOption.errorHandling = transformerErrorHandling.

7.6.1 Enable to Take Address

[TPS_SWCT_01355] enableTakeAddress = true ⌈If the attribute enableTakeAddress = true the generated API functions related to this PortPrototype shall be implemented by means of true/native C functions (as opposed to function-like preprocessor macros) so that it is possible to access the API functions via their address (by means of function-pointers).⌋*(RS_SWCT_03040)*

The main focus of the feature is support for configuration of AUTOSAR Services which are limited to single instances.

[constr_2024] enableTakeAddress is restricted to single instantiation ⌈The definition of a PortAPIOption with enableTakeAddress set to true is only permitted for software-components where the attribute SwcInternalBehavior.supportsMultipleInstantiation is set to false **at the time when the contract phase generation is executed**.⌋*()*

7.6.2 Indirect API Generation

[TPS_SWCT_01356] indirectAPI option switches the generation of the RTE's indirect API functionality ⌈The indirectAPI option switches the generation of the RTE's indirect API functionality for a certain PortPrototype. The generated indirect API does allow iterating over ports within the SW-Component.⌋*(RS_SWCT_03040)*

7.6.3 Port Defined Argument Value

[TPS_SWCT_01357] Definition of implicit values that are passed by the RTE to the server's entry point ⌈In addition to the formal parameters of a client/server invocation that are defined as part of the server's PortInterface, it is possible to specify a number of implicit values that are passed by the RTE to the server's entry point.⌋*(RS_SWCT_03040)*

The initial need for this feature arises in the context of basic software services - although it is not limited to those.

For a service like the NVRAM manager, every accessing port is in addition to its logical identity - as a sequence of shortNames - uniquely identified through a NVRAM specific memory block id. This block id shall be defined in the context of ECU integration and not by the client components.

Instead of exposing this mechanism on the logical ClientServerInterface level in form of a formal argument, one or more PortDefinedArgumentValues can be specified.

[TPS_SWCT_01358] Values are hidden from the client components ⌈Because these values are specified in the context of the provide-port only they are hidden from the client components keeping their design and code independent of the server component details.⌋*(RS_SWCT_03040)*

In the example of the NVRAM manager, this allows to define the block id in the context of ECU integration and not by the client components.

Figure 7.34 shows the meta-model of Port API Options and the portArgValue.

[constr_1150] Usage of valueType for PortDefinedArgumentValue ⌈The valueType (typically this boils down to integer values used to specify an “id”) associated with PortDefinedArgumentValue shall be of category VALUE or TYPE_REFERENCE. The latter case is only supported if the value of category of the target data type is set to VALUE.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

In case of a PPortPrototype of the NVRAM example this list would have just one value of type int8 or int16 holding the memory block id.

[constr_1386] PortDefinedArgumentValue shall only be defined for AbstractProvidedPortPrototype ⌈A PortAPIOption which aggregates at least one PortDefinedArgumentValue in the role portArgValue shall reference an AbstractProvidedPortPrototype typed by a ClientServerInterface in the role port **at the time when the RTE is generated**.⌋*()*

To be clear, this means that PortDefinedArgumentValues may not be used together with RPortPrototypes.

Class	**PortDefinedArgumentValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PPortPrototypes of a ClientServer Interface.			
Base	*ARObject*			
Aggregated by	PortAPIOption.portArgValue			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	ValueSpecification	0..1	aggr	Specifies the actual value.
valueType	ImplementationData Type	0..1	tref	The implementation type of this argument value. It should not be composite type or a pointer. **Stereotypes:** isOfType

Table 7.47: PortDefinedArgumentValue

[constr_1961] Existence of attribute PortDefinedArgumentValue.value ⌈For each PortDefinedArgumentValue, attribute value shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1962] Existence of attribute PortDefinedArgumentValue.valueType ⌈For each PortDefinedArgumentValue, attribute valueType shall exist **at the time when the RTE is generated**.⌋*()*

7.6.4 Supported Features

Historically, the PortAPIOption has undergone a number of extensions that usually ended up in additional primitive or composite attributes.

As further requests for extensions keep coming in, focus was put on limiting the complexity of the overall modeling of PortAPIOption. In response to this, a new extension approach has been defined to keep the surroundings of PortAPIOption manageable.

In particular, PortAPIOption aggregates the abstract meta-class SwcSupportedFeature in the role supportedFeature (see Figure 7.34).

The actual aggregation of supportedFeature will consist of concrete sub-classes of SwcSupportedFeature.

It will be possible to add further sub-classes of SwcSupportedFeature to add further functionality without increasing the modeling complexity of PortAPIOption, at the expense of having to formulate additional constraints.

Class	***SwcSupportedFeature*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	This meta-class represents a abstract base class for features that can be supported by a RunnableEntity.			
Base	*ARObject*			
Subclasses	CommunicationBufferLocking			
Aggregated by	PortAPIOption.supportedFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.48: SwcSupportedFeature

7.6.4.1 Buffer Locking

[TPS_SWCT_01687] Support of locked communication buffers ⌈If a CommunicationBufferLocking where attribute supportBufferLocking is set to value supportsBufferLocking is aggregated in the role PortAPIOption.supportedFeature then all RunnableEntitys accessing the enclosing PortPrototype shall be able to support the return value RTE_E_COM_BUSY.⌋*(RS_SWCT_03040)*

[constr_1432] Multiplicity of CommunicationBufferLocking ⌈In a concrete aggregated set of PortAPIOption.supportedFeature, CommunicationBufferLocking shall exist **at most once at the time when the RTE is generated**.⌋*()*

Class	CommunicationBufferLocking			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	The aggregation of this meta-class specifies that a RunnableEntity supports locked communication buffers supplied by the RTE. It is able to cope with the error RTE_E_COM_BUSY.			
Base	*ARObject*, *SwcSupportedFeature*			
Aggregated by	PortAPIOption.supportedFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
supportBuffer Locking	SupportBufferLocking Enum	0..1	attr	This attribute is used to indicate the intended buffer locking behavior.

Table 7.49: CommunicationBufferLocking

[constr_1963] Existence of attribute CommunicationBufferLocking.supportBufferLocking ⌈For each CommunicationBufferLocking, attribute supportBufferLocking shall exist **at the time when the RTE is generated**.⌋*()*

Enumeration	SupportBufferLockingEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions
Note	This enumeration represents the ability to define the buffer locking behavior.
Aggregated by	CommunicationBufferLocking.supportBufferLocking
Literal	**Description**
doesNotSupport BufferLocking	Buffer locking is not supported. **Tags:**atp.EnumerationLiteralIndex=0
supportsBuffer Locking	Buffer locking is supported. **Tags:**atp.EnumerationLiteralIndex=1

Table 7.50: SupportBufferLockingEnum

[constr_10097] Buffer locking is only supported if returnValueProvision is set to returnValueProvided ⌈Setting the value of attribute PortAPIOption.supportedFeature.supportBufferLocking to value supportsBufferLocking is only supported if the AbstractAccessPoints that refer to values in the respective PortAPIOption.port do **not** define AbstractAccessPoint.returnValueProvision **or** set the value of AbstractAccessPoint.returnValueProvision to returnValueProvided.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

7.7 PerInstanceMemory

[TPS_SWCT_01359] Private memory per instance ⌈AtomicSwComponentTypes that support multiple instantiation (attribute supportsMultipleInstantiation == true) will typically need a given amount of private memory per instance. It is the responsibility of the RTE to provide a mechanism with which each instance of an AtomicSwComponentType can access its own instance-specific memory.⌋*(RS_SWCT_03040)*

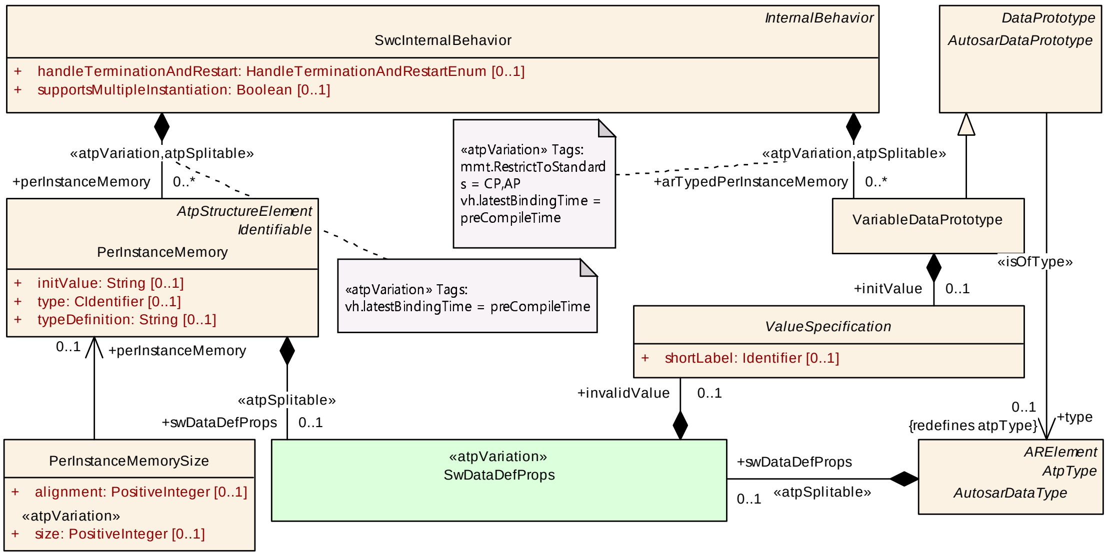


Figure 7.35: PerInstanceMemory

[TPS_SWCT_01360] Arbitrary number of per-instance memory blocks ⌈An AtomicSwComponentType can define an arbitrary number of per-instance memory blocks.⌋*(RS_SWCT_03040)*

[TPS_SWCT_01361] attribute supportsMultipleInstantiation == false ⌈AtomicSwComponentTypes that do *not* support multiple instantiation (attribute supportsMultipleInstantiation == false) do not necessarily need to use the PerInstanceMemory: because there will only be a single instance of the AtomicSwComponentType on an ECU, the AtomicSwComponentType can use static variables to store the AtomicSwComponentType's internal state.

However, the usage of PerInstanceMemory is also allowed in this case.⌋*(RS_SWCT_03040)*

[TPS_SWCT_01362] Initialization of PerInstanceMemory ⌈Note that the PerInstanceMemory is not initialized by the RTE if no initValue is defined. In this case, it is the responsibility of the AtomicSwComponentType to initialize the PerInstanceMemory.⌋*()*

7.7.1 PerInstanceMemory typed by "C" Data Types

[TPS_SWCT_01363] PerInstanceMemory typed by "C" Data Types ⌈For each such memory block, the software-component description shall provide the name of the data type (the "C"-type) it needs to store in the memory block in the attribute type.

This attribute allows for the RTE to generate an API function that provides a convenient and type-safe access to the data item.

In addition, the software-component description shall define the data type in the attribute typeDefinition. This attribute is supposed to contain a *C* typedef of the data type in valid C-syntax.⌋*(RS_SWCT_03040)*

In other words, this typeDefinition shall be formulated such that it can be included verbatim in a C header file.

[constr_2007] Consistency of typeDefinition attribute ⌈All PerInstanceMemorys of the same SwcInternalBehavior with identical type attribute shall define an identical typeDefinition attribute as well.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

[TPS_SWCT_01364] Initial value of a PerInstanceMemory typed by "C" Data Types ⌈The initValue is a comma separated list which can be used verbatim by the RTE generator as constant initializer.⌋()

[TPS_SWCT_01574] PerInstanceMemory.typeDefinition shall not contain a function pointer ⌈The attribute PerInstanceMemory.typeDefinition is not allowed to contain a function pointer.⌋()

Please note that, although [TPS_SWCT_01574] is formulated like a constraint and the statement that it makes certainly has a constraint-ish nature, there is hardly a way to actually **enforce** the regulation because the content of PerInstanceMemory.typeDefinition is non-formal (modeled by the non-specific i.e. String).

Therefore, a specification item has been used for the description of the respective semantics rather than a constraint.

More details on the use of these attributes in the generation of software-component header-files can be found in the RTE specification [2].

Class	**PerInstanceMemory**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PerInstanceMemory			
Note	Defines a 'C' typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is set to "true" or if the software-component defines NVRAM access via permanent blocks.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.perInstanceMemory			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	String	0..1	attr	Specifies initial value(s) of the PerInstanceMemory
swDataDefProps	SwDataDefProps	0..1	aggr	This represents the ability to to allocate RAM at specific memory sections, for example, to support the RAM Block recovery strategy by mapping to uninitialized RAM. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps
type	CIdentifier	0..1	attr	The name of the "C"-type
typeDefinition	String	0..1	attr	A definition of the type with the syntax of a 'C' typedef.

Table 7.51: PerInstanceMemory

[constr_1964] Existence of attribute PerInstanceMemory.type ⌈For each PerInstanceMemory, attribute type shall exist **at the time when the contract phase generation is executed**.⌋()

[constr_1965] Existence of attribute PerInstanceMemory.typeDefinition ⌈For each PerInstanceMemory, attribute typeDefinition shall exist **at the time when the contract phase generation is executed**.⌋*()*

7.7.2 PerInstanceMemory typed by AUTOSAR Data Types

[TPS_SWCT_01365] PerInstanceMemory typed by AUTOSAR Data Types ⌈A PerInstanceMemory typed with AUTOSAR data types is defined by a VariableDataPrototype in the role arTypedPerInstanceMemory.

VariableDataPrototype is derived from DataPrototype which has an association to an AutosarDataType.⌋*()*

This defines the data type of the AUTOSAR-typed PerInstanceMemory.

[TPS_SWCT_01366] Initial value of a PerInstanceMemory typed by AUTOSAR Data Types ⌈The initValue is described with a ValueSpecification⌋*(RS_SWCT_03040)*

[TPS_SWCT_01367] Typed by AUTOSAR data type vs. typed by C data type ⌈In difference to the “C” typed PerInstanceMemory the AUTOSAR-typed PerInstanceMemory is able to define information controlling the visibility in an MCD system via a SwDataDefProps for the purpose of measurement or defining an input value of an axis.⌋*(RS_SWCT_03040)*

For more information about the relevance for measurement please refer to chapter 5.4.3. The aspect of defining an input value of an axis is explained in chapter 5.4.5.

Note: Due to the use of AutosarDataType the AUTOSAR-typed PerInstanceMemory can not support C++ specific types or pointer types directly.

7.8 Static Memory and Constant Memory

[TPS_SWCT_01368] Describe static and constant memory ⌈Static memory (formalized by means of InternalBehavior.staticMemory) and constant memory (formalized by means of InternalBehavior.constantMemory) can be used whenever AutosarDataTypes should be used in the implementation of an AtomicSwComponentType but no involvement of the RTE (for memory allocation and management) is required.⌋*(RS_SWCT_03040)*

This includes special cases of measurement and calibration but also debugging.

[TPS_SWCT_01483] Use static and constant memory to support Measurement and Calibration ⌈The information about these characteristic values and variables is given with the purpose to support Measurement and Calibration and has to be taken into account for the generation of A2L files.

A proprietary generator shall take care of these data for the purpose of generating A2L.⌋*(RS_SWCT_03040)*

Please note that the topic "measurement and calibration" is discussed in chapter 2.2.

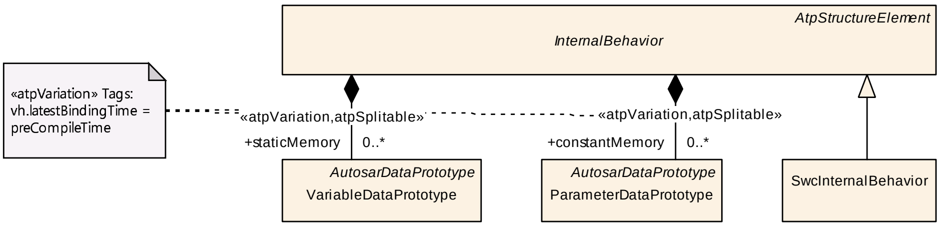


Figure 7.36: Static Memory and Constant Memory

[TPS_SWCT_01369] Static and constant memory is not instantiated by the RTE ⌈In contrast to the other kinds of memory like `implicitInterRunnableVariable`, `implicitInterRunnableVariable`, `PerInstanceMemory`, `sharedParameter` or `perInstanceParameter` the `staticMemory` and `constantMemory` are **not** instantiated by the RTE.⌋*(RS_SWCT_03040)*

This allows for more efficient implementations (especially for software-components provided as object code) by avoidance of the additional indirection caused by the RTE's component data structure.

Further on, this kind of memory reduces the dependencies of the software-component implementation to generated RTE code which is appreciated for safety related functionalities.

Due to the instantiation of the memory by the software-component's implementation the `constantMemory` behaves like a `sharedParameter` (see chapter 2.2.3.2)

[constr_2028] `staticMemory` is restricted to single instantiation ⌈The `staticMemory` is only supported if the attribute `supportsMultipleInstantiation` of the owning `SwcInternalBehavior` is set to `false` **`at the time when the RTE is generated`**⌋*()*

This constraint prevents hidden communication between `SwComponentPrototype`s of the same `SwComponentType`.

[TPS_SWCT_01849] `shortName` of `constantMemory` and `staticMemory` ⌈The `shortName` of a `VariableDataPrototype` in role `staticMemory` or a `ParameterDataPrototype` in role `constantMemory` has to be equal with the 'C' identifier of the described variable or constant.⌋*(RS_SWCT_03040)*

7.9 Included AUTOSAR Data Types

[TPS_SWCT_01155] `IncludedDataTypeSet` ⌈An `IncludedDataTypeSet` declares that a set of `AutosarDataType`s are used for the C / C++ implementation of the software component. The AutosarDataTypes become part of the contract.⌋ *(RS_SWCT_03040)*

[TPS_SWCT_01156] Required if the AutosarDataType is not used for any DataPrototype ⌈This information is required if the AutosarDataType is not used for any DataPrototype owned by this software component or if a prefix for C language identifiers belonging to AutosarDataTypes shall be defined.⌋ *(RS_SWCT_03040)*

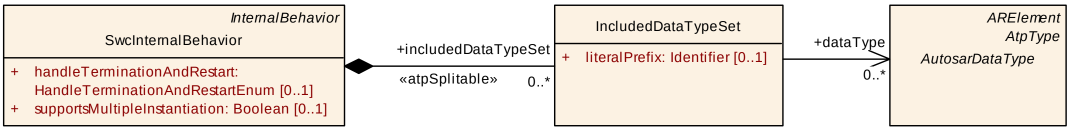


Figure 7.37: Included AUTOSAR Data Types

This supports the common usage of the AUTOSAR data type system for RTE provided memory objects and memory objects declared by the software component implementation.

Further on, this enables the generation of the RTE Application Types Header File for AUTOSAR services containing the required data types for the C-API before the data type usage in dedicated ports for an ECU is known.

Class	**IncludedDataTypeSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::IncludedDataTypes			
Note	An includedDataTypeSet declares that a set of AutosarDataType is used by a basic software module or a software component for its implementation and the AutosarDataType becomes part of the contract. This information is required if the AutosarDataType is not used for any DataPrototype owned by this software component or if the enumeration literals, lowerLimit and upperLimit constants shall be generated with a literalPrefix. The optional literalPrefix is used to add a common prefix on enumeration literals, lowerLimit and upper Limit constants created by the RTE.			
Base	*ARObject*			
Aggregated by	BswInternalBehavior.includedDataTypeSet, SwcInternalBehavior.includedDataTypeSet			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataType	AutosarDataType	*	ref	AutosarDataType belonging to the includedDataTypeSet
literalPrefix	Identifier	0..1	attr	LiteralPrefix defines a common prefix for all AutosarData Types of the includedDataTypeSet to be added on enumeration literals, lowerLimit and upperLimit constants created by the RTE.

Table 7.52: IncludedDataTypeSet

[TPS_SWCT_01157] Attribute literalPrefix of IncludedDataTypeSet ⌈In addition, the literalPrefix might be used to separate the namespace of C language identifiers belonging to equally named AutosarDataTypes used for the same software component C implementation.⌋ *(RS_SWCT_03040)*

7.10 Included Mode Declaration Groups

[TPS_SWCT_01153] **IncludedModeDeclarationGroupSet** ⌈Similar to the consideration of data types using IncludedDataTypeSet, SwcInternalBehavior aggregates IncludedModeDeclarationGroupSet that in turn allows for referencing ModeDeclarationGroups with the intent to express that the referenced ModeDeclarationGroups are used in the context of the enclosing AtomicSwComponentType.⌋*(RS_SWCT_03040, RS_SWCT_03110)*

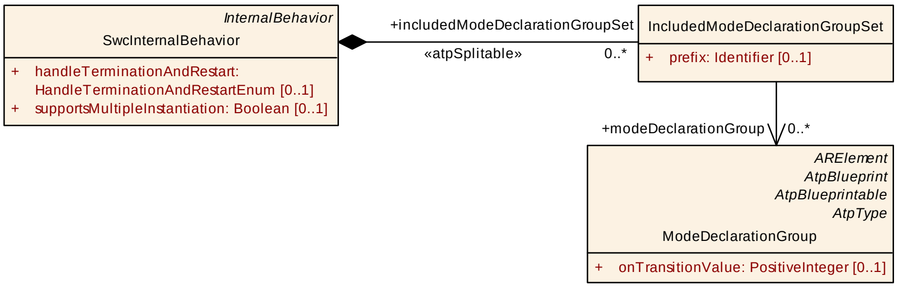


Figure 7.38: Included ModeDeclarationGroups

Class	**IncludedModeDeclarationGroupSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	An IncludedModeDeclarationGroupSet declares that a set of ModeDeclarationGroups used by the software component for its implementation and consequently these ModeDeclarationGroups become part of the contract.			
Base	*ARObject*			
Aggregated by	BswInternalBehavior.includedModeDeclarationGroupSet, SwcInternalBehavior.includedModeDeclarationGroupSet			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
mode Declaration Group	ModeDeclarationGroup	*	ref	This represents the referenced ModeDeclarationGroup.
prefix	Identifier	0..1	attr	The prefix shall be used by the RTE generator as a prefix for the creation of symbols related to the referenced ModeDeclarationGroups, e.g RTE_TRANSITION_<ModeDeclarationGroup>.

Table 7.53: IncludedModeDeclarationGroupSet

[TPS_SWCT_01154] Attribute prefix of IncludedModeDeclarationGroupSet ⌈The optional attribute prefix of IncludedModeDeclarationGroupSet can be used to define a prefix that the RTE generator shall use to define symbols related to the included ModeDeclarationGroups with the intent to avoid potential name clashes.⌋ *(RS_SWCT_03040, RS_SWCT_03110)*

Rationale: If the attribute prefix is required, changes to software-component source code may be necessary.

7.11 Service Needs

7.11.1 Overview

[TPS_SWCT_01043] ApplicationSwComponentTypes are independent from actual ECU Hardware ⌈ApplicationSwComponentTypes are designed to be independent of their mapping to actual ECU Hardware.⌋*(RS_SWCT_02060)*

However, each software-component might need services which are provided by the ECU Basic Software through AUTOSAR Services.

[TPS_SWCT_01044] ServiceNeeds ⌈The ServiceNeeds are used to provide detailed information what the software-component expects from the AUTOSAR Services when integrated on an actual ECU.

Note that only AtomicSwComponentTypes and NvBlockSwComponentTypes can be connected to AUTOSAR Services.⌋*(RS_SWCT_02060)*

Please note that some ServiceNeeds are on display in Figures 7.39, 13.16, 13.13, and 13.18.

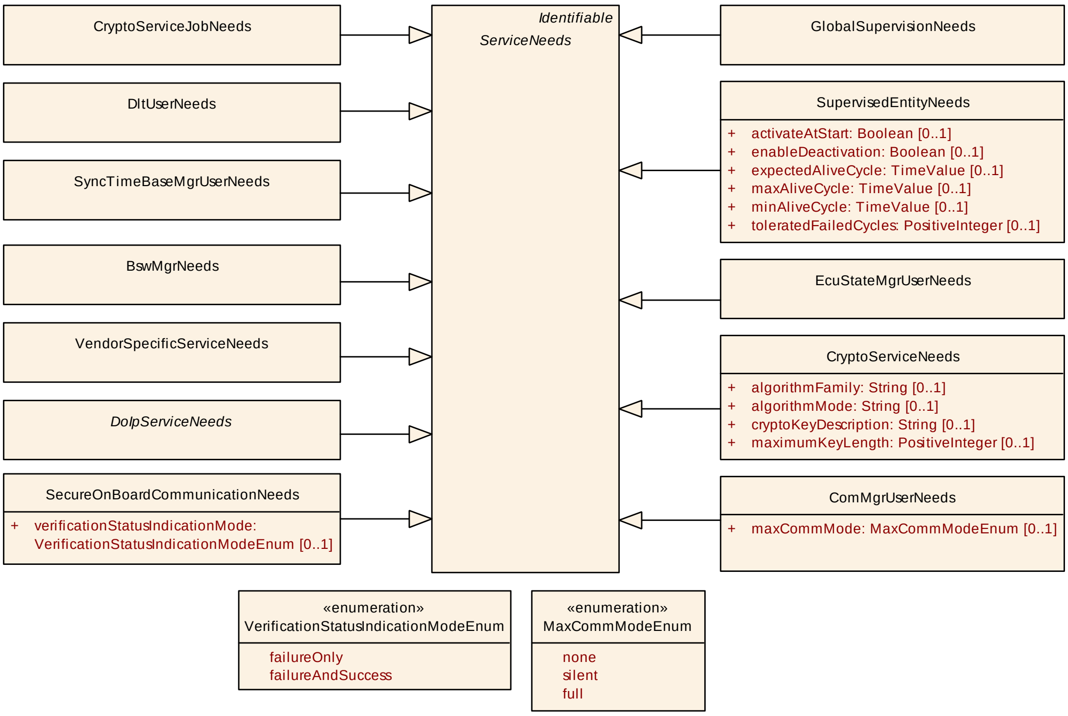


Figure 7.39: Meta-class ServiceNeeds and some of its specializations

[TPS_SWCT_01045] Actual values of ECU configuration parameters fulfill the requirements given by the ServiceNeeds ⌈When integrating application software-components on an ECU, the actual values of ECU configuration parameters shall be chosen so that they fulfill the requirements given by the ServiceNeeds of all the integrated AtomicSwComponentTypes.⌋*(RS_SWCT_02060)*

Note that the actual values of configuration parameters will in addition depend on the properties of the basic software and the hardware of that specific ECU, see also chapter 11.

For further information about the relation between the ServiceNeeds and the ECU configuration parameters see [34].

The meta-class ServiceNeeds and the sub-classes for several Services are located in the CommonStructure package of the meta-model because they are also used in the Basic Software Module Description Template [6].

The semantics of sub-classes of ServiceNeeds is explained in the respective sub-chapters of chapter 13.

Class	***ServiceNeeds*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This expresses the abstract needs that a Software Component or Basic Software Module has on the configuration of an AUTOSAR Service to which it will be connected. "Abstract needs" means that the model abstracts from the Configuration Parameters of the underlying Basic Software.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswMgrNeeds, ComMgrUserNeeds, CryptoKeyManagementNeeds, CryptoServiceJobNeeds, CryptoServiceNeeds, *DiagnosticCapabilityElement*, DltUserNeeds, *DoIpServiceNeeds*, EcuStateMgrUserNeeds, ErrorTracerNeeds, FunctionInhibitionAvailabilityNeeds, FunctionInhibitionNeeds, GlobalSupervisionNeeds, HardwareTestNeeds, IdsMgrCustomTimestampNeeds, IdsMgrNeeds, IndicatorStatusNeeds, J1939DcmDm19Support, J1939RmIncomingRequestServiceNeeds, J1939RmOutgoingRequestServiceNeeds, NvBlockNeeds, SecureOnBoardCommunicationNeeds, SupervisedEntityCheckpointNeeds, SupervisedEntityNeeds, SyncTimeBaseMgrUserNeeds, V2xDataManagerNeeds, V2xFacUserNeeds, V2xMUserNeeds, VendorSpecificServiceNeeds			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.54: ServiceNeeds

Please note that the vast majority of the subclasses of meta-class ServiceNeeds are associated with standardized behavior of AUTOSAR services. However, there are cases where a user-specific behavior is required and for this purpose a specific flavor of ServiceNeeds is available.

[TPS_SWCT_01693] Usage of VendorSpecificServiceNeeds ⌈It is possible to define VendorSpecificServiceNeeds for the purpose of implementing a vendor-specific, i.e. non-standardized, service. VendorSpecificServiceNeeds does not provide any attributes and its meaning shall be described by means of the category attribute.⌋*(RS_SWCT_02060)*

Class	**VendorSpecificServiceNeeds**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This represents the ability to define vendor-specific service needs.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds

▽

△

Class	VendorSpecificServiceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 7.55: VendorSpecificServiceNeeds

7.11.2 Assignment of Service Needs to Ports and Data

[TPS_SWCT_01046] ServiceNeeds are defined in the scope of the SwcInternalBehavior ⌈ServiceNeeds specified by AtomicSwComponentTypes are defined in the scope of the SwcInternalBehavior because in several cases they need associations to other parts of the SwcInternalBehavior.

In most cases they are related to certain PortPrototypes belonging to the AtomicSwComponentTypes because AtomicSwComponentTypes communicate with AUTOSAR Services via these PortPrototypes.⌋*(RS_SWCT_02060)*

In addition, a ServiceNeeds element can also have relations to some data declared within the same SwcInternalBehavior, namely some use cases of the NVRAM Service, require a Permanent RAM Block and/or ROM Block declared in the context of the single software component.

A further use case requires that a ServiceNeeds element is linked to a PortGroup. Especially, a ServiceNeeds can represent a group of PortPrototypes as input to configure the communication manager in order to handle the communication state of those PortPrototypes.

These relationships to PortPrototypes, data and PortGroups are required as input for tools in order to generate the XML descriptions and configurations of the basic software which implements the Service according to the needs of several AtomicSwComponentTypes are integrated on an ECU, see chapter 11.

The relationship to PortPrototypes is defined via the meta-class RoleBasedPortAssignment and the relationship to data is defined via the meta-class RoleBasedDataAssignment.

Class	RoleBasedPortAssignment			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular service port (RPortPrototype or PPort Prototype) of an AtomicSwComponentType. With this assignment, the role of the service port can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct connector.			
Base	*ARObject*			
Aggregated by	NvBlockDescriptor.clientServerPort, SwcServiceDependency.assignedPort			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**RoleBasedPortAssignment**			
portPrototype	PortPrototype	0..1	ref	Service PortPrototype used in the assigned role. This PortPrototype shall either belong to the same AtomicSwComponentType as the SwcInternalBehavior which owns the ServiceDependency or to the same NvBlockSwComponentType as the NvBlockDescriptor.
role	Identifier	0..1	attr	This is the role of the assigned Port in the given context. The value shall be a shortName of the Blueprint of a Port Interface as standardized in the Software Specification of the related AUTOSAR Service.

Table 7.56: RoleBasedPortAssignment

Both are aggregating an attribute `role` which allows to define the role of the `PortPrototype`s or data in the specific context.

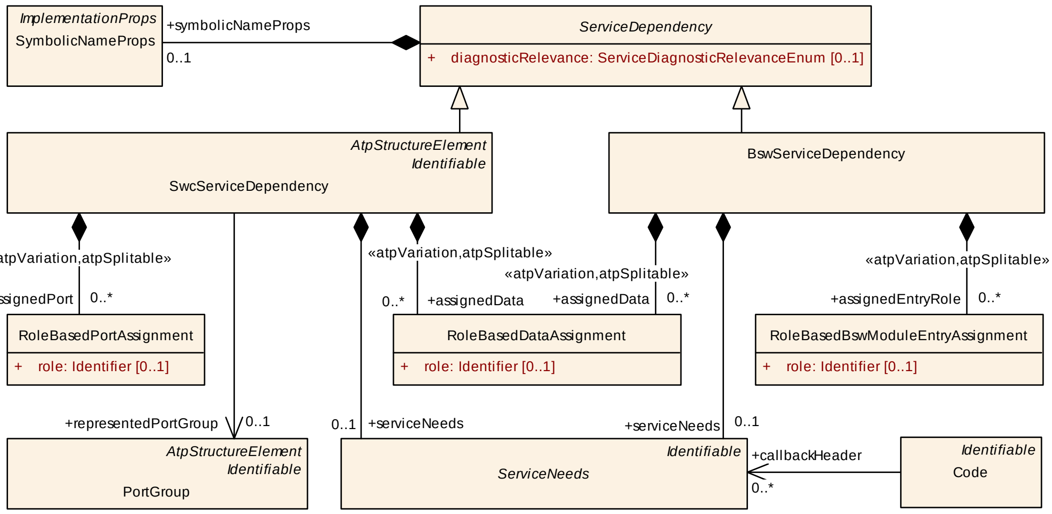


Figure 7.40: `ServiceDependency` is the abstract base class of `SwcServiceDependency`

[constr_2027] **`SwcServiceDependency` shall be defined for service ports only** ⌈A `PortPrototype` that is referenced by a `SwcServiceDependency` via `assignedPort` or via `assignedData` shall be typed by a `PortInterface` that has `isService` set to `true` **`at the time when the RTE is generated`**.

This rule does **not** apply to `PortPrototype`s referenced by a `RoleBasedPortAssignment` where the attribute `role` is set to any of the following values:

- `NvMService`
- `NvMNotifyJobFinished`
- `NvMNotifyInitBlock`
- `NvMAdmin`
- `NvMMirror`
- `NvDataPort`

Furthermore, the rule does **not** apply to the case described in [TPS_SWCT_01579], [TPS_SWCT_01831], [TPS_SWCT_01580], and [TPS_SWCT_01572].⌋()

The actual mapping between the `ServiceNeeds` element and its various relationships is provided by the meta-class `SwcServiceDependency` as shown in figure 7.41.

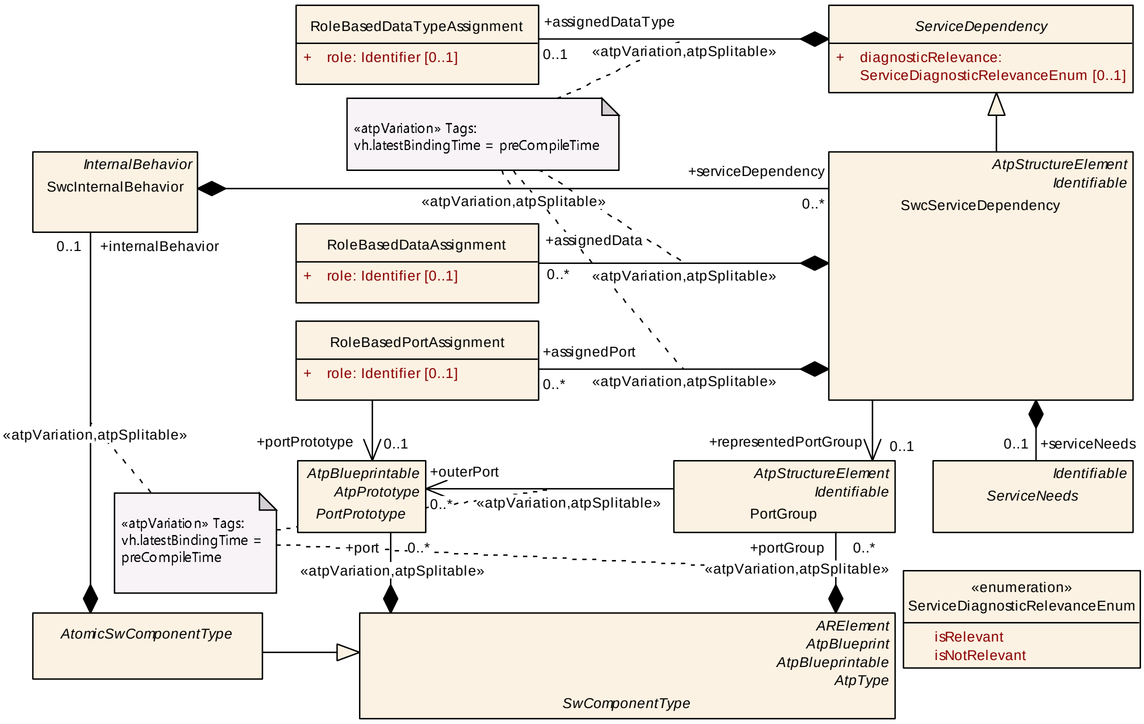


Figure 7.41: `SwcServiceDependency` in the `SwcInternalBehavior`

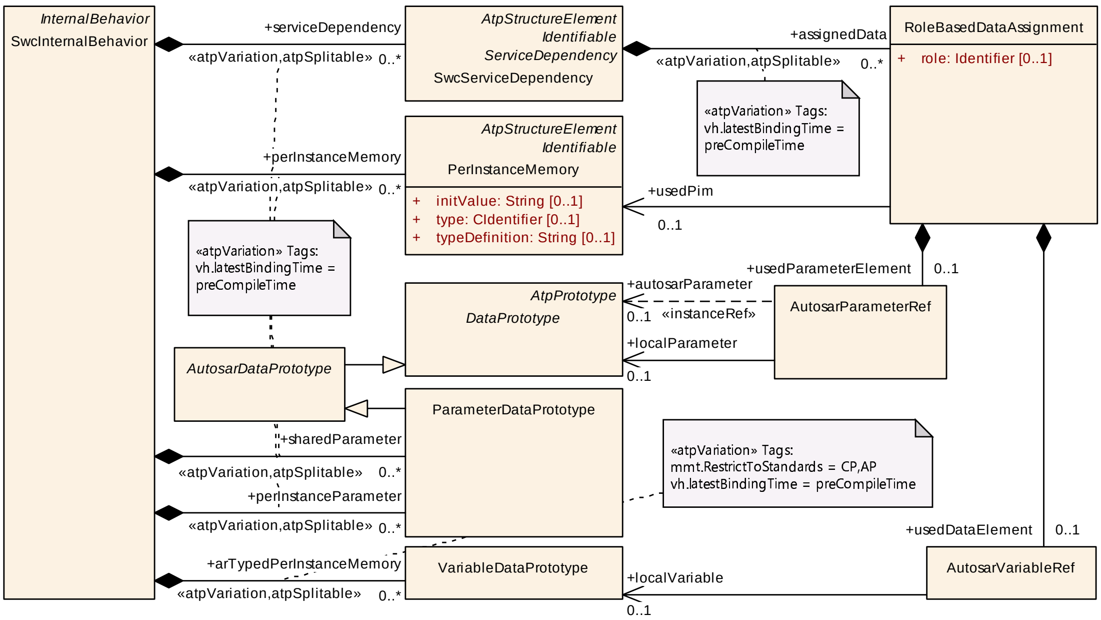


Figure 7.42: Details of `RoleBasedDataAssignment` for local data

Note the difference between the associations to PortPrototypes and to PortGroups: While the RoleBasedPortAssignment is part of the SwcInternalBehavior a PortGroup is defined for the SwComponentType (thus belongs to the VFB level) and it is linked to the PortGroups of other SwComponentTypes.

This means a PortGroup represents a system feature, whereas the RoleBasedPortAssignment is a local feature for the purpose of communication with the AUTOSAR Service.

Class	**RoleBasedDataAssignment**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class specifies an assignment of a role to a particular data object in either • the SwcInternalBehavior of a software component (or in the BswInternalBehavior of a BSW module or BSW cluster) in the context of an AUTOSAR Service or • an NvBlockDescriptor to sort out the assignment of event-based writing strategies to data elements in a PortPrototype. With this assignment, the role of the data can be mapped to a DataPrototype that is used in the context of the definition of a specific ServiceNeeds or NvBlockDescriptor, so that a tool is able to create the correct access or writing strategy.			
Base	*ARObject*			
Aggregated by	BswServiceDependency.assignedData, NvBlockDescriptor.writingStrategy, SwcServiceDependency.assignedData			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
role	Identifier	0..1	attr	This is the role of the assigned data in the given context. Possible values need to be specified on M1 level. Additionally the TPS Software Component Template provides a list of applicable roles for various service dependencies and service use cases in chapter 13 "Service Dependencies and Service Use Cases" (e.g., ramBlock in case of the needs for a permanent RAM block).
usedData Element	AutosarVariableRef	0..1	aggr	The VariableDataPrototype used in this role, e.g. • Permanent RAM Block of an NVRAM Block which shall belong to the same SwcInternal Behavior or BswInternalBehavior. • In the role signalBasedDiagnostics it has to refer to a VariableDataPrototype in a SenderReceiver Interface or a NvDataInterface.
usedParameter Element	AutosarParameterRef	0..1	aggr	The ParameterDataPrototype used in this role, e.g. • ROM Block of an NVRAM Block. It shall belong to the same SwcInternalBehavior or Bsw Internalbehavior. • In the role signalBasedDiagnostics it has to refer to a ParameterDataPrototype in a Parameter Interface.
usedPim	PerInstanceMemory	0..1	ref	The (untyped) PerInstanceMemory used in this role (e.g. as a Permanent RAM Block for an NVRAM Block).

Table 7.57: RoleBasedDataAssignment

[TPS_SWCT_01556] Rule for setting RoleBasedPortAssignment.role ⌈The value of RoleBasedPortAssignment.role cannot arbitrarily set but shall to equal to the shortName of the applicable PortInterface taken from the standardized AUTOSAR Service Interface model (this implies that the category of the

ARPackage that owns the PortInterface is set to BLUEPRINT[5] and the top-most ARPackage.shortName is set to AUTOSAR, see also [30]).⌋(RS_SWCT_02060)

[TPS_SWCT_01660] Values of SwcServiceDependency.category reserved by the standard ⌈The following values of SwcServiceDependency.category are reserved by the AUTOSAR standard:

SERVICE : this applies for all the cases where SwcServiceDependency is intended to be used for the design of ServiceSwComponentTypes.

NV_BLOCK_COMPONENT : this applies if the SwcServiceDependency is intended to be used for the design of an NvBlockSwComponentType.

⌋(RS_SWCT_02060)

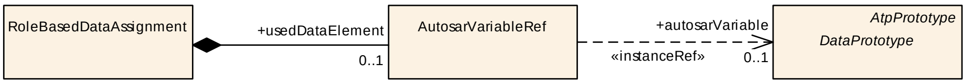


Figure 7.43: Details of RoleBasedDataAssignment for accessing DataPrototypes in PortPrototypes

Class	**SwcServiceDependency**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceDependency*			
Aggregated by	AdaptiveSwcInternalBehavior.serviceDependency, *AtpClassifier*.atpFeature, SwcInternalBehavior.serviceDependency			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
assignedData	RoleBasedDataAssignment	*	aggr	Defines the role of an associated data object of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedData, assignedData.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
assignedPort	RoleBasedPortAssignment	*	aggr	Defines the role of an associated port of the same component. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedPort, assignedPort.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

[5]see [TPS_STDT_00033]

△

Class	SwcServiceDependency			
representedPort Group	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup shall be local to this atomic SWC, but via the links between the Port Groups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	0..1	aggr	The associated ServiceNeeds.

Table 7.58: SwcServiceDependency

Class	***ServiceDependency*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Collects all dependencies of a software module or component on an AUTOSAR Service related to a specific item (e.g. an NVRAM Block, a diagnostic event etc.). It defines the quality of service (Service Needs) of this item as well as (optionally) references to additional elements. This information is required for tools in order to generate the related basic software configuration and ServiceSwComponentTypes.			
Base	*ARObject*			
Subclasses	BswServiceDependency, SwcServiceDependency			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
assignedData Type	RoleBasedDataType Assignment	0..1	aggr	This is the role of the assignment data type in the given context. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedDataType, assignedData Type.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
diagnostic Relevance	ServiceDiagnostic RelevanceEnum	0..1	attr	If this attribute indicates a relevance for diagnostics then the integrator has a much easier time identifying the candidates for the configuration of the diagnostic stack. Example: identification of mode conditions (e.g. communication between application and BswM) relevant for the Dcm.
symbolicName Props	SymbolicNameProps	0..1	aggr	This attribute can be taken to contribute to the creation of symbolic name values.

Table 7.59: ServiceDependency

Enumeration	**ServiceDiagnosticRelevanceEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration provides values to describe the diagnostic relevance of a SwcServiceDependency (specifically if the aggregated ServiceNeeds itself does not indicate a relevance for diagnostics).
Aggregated by	*ServiceDependency*.diagnosticRelevance
Literal	***Description***
isNotRelevant	This value indicates that a relevance for diagnostics does not exist. **Tags:**atp.EnumerationLiteralIndex=0
isRelevant	This value indicates a relevance for diagnostics. **Tags:**atp.EnumerationLiteralIndex=1

Table 7.60: ServiceDiagnosticRelevanceEnum

Class	**SymbolicNameProps**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class can be taken to contribute to the creation of symbolic name values.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Aggregated by	*ServiceDependency*.symbolicNameProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 7.61: SymbolicNameProps

[TPS_SWCT_01661] Default value of SwcServiceDependency.category ⌈If the attribute SwcServiceDependency.category does not exist then the value **SERVICE** shall be assumed for SwcServiceDependency.category.⌋*(RS_SWCT_02060)*

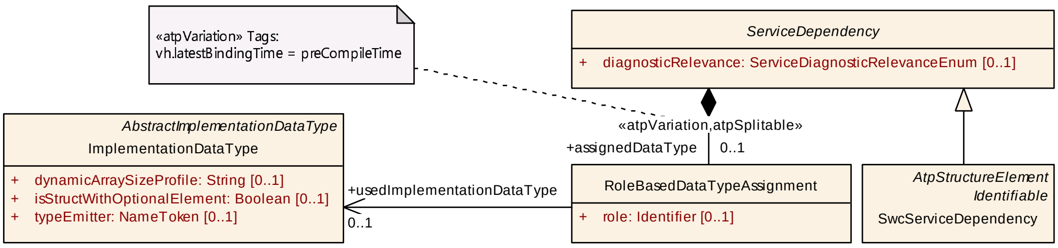


Figure 7.44: Details of RoleBasedDataTypeAssignment for local data

Class	**RoleBasedDataTypeAssignment**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular data type of a software component (or in the BswModuleBehavior of a module or cluster) in the context of an AUTOSAR Service. With this assignment, the role of the data type can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct access.			
Base	*ARObject*			
Aggregated by	*ServiceDependency*.assignedDataType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
role	Identifier	0..1	attr	This is the role of the associated data type in the given context.
used Implementation DataType	ImplementationData Type	0..1	ref	This represents the associated ImplementationDataType.

Table 7.62: RoleBasedDataTypeAssignment

[constr_10020] Existence of attribute RoleBasedDataTypeAssignment.usedImplementationDataType ⌈For each RoleBasedDataTypeAssignment, attribute usedImplementationDataType shall exist **at the time when the RTE is generated**.⌋*()*

Please note that there are cases where the granularity of existing ServiceInterfaces does not match the granularity of existing SwcServiceDependency.serviceNeeds.

In other words, there are Service Interfaces that cover the semantics of different kinds of ServiceNeeds. One example is the ClientServerInterface DataServices_{Data} which basically supports the access to diagnostic values **as well as** I/O control of the same value.

Figure 7.45 provides a graphical sketch of how the modeling for this case is foreseen.

[TPS_SWCT_01689] Relation between SwcServiceDependencys and PortPrototypes ⌈It is positively possible to create a model where two or more SwcServiceDependencys, by way of the RoleBasedPortAssignment or RoleBasedDataAssignment, refer to a single PortPrototype.⌋*(RS_SWCT_02060)*

As indicated by Figure 7.45, there are two potentially competing SwcServiceDependencys that could be taken to contribute their shortName for filling in the suffix of the DataServices_{Data}.

In this case, it is actually necessary to settle the "over-supply of shortNames" by regulation of the AUTOSAR standard. [TPS_SWCT_01691] has been created for this purpose.

Another realistic example where [TPS_SWCT_01689] applies is an AtomicSwComponentType that exposes a PPortPrototype typed by a SenderReceiverInterface and the dataElement(s) within the PPortPrototype are both accessed as diagnostic values (see chapter 13.8.4.3) **and** are used to send mode requests to the BswM (see chapter 13.6.4).

Note that in this case a regulation regarding the shortNames of the affected SwcServiceDependencys is not required because the applicable SenderReceiverInterface is not standardized and does not require the assignment of a name suffix from the existing model.

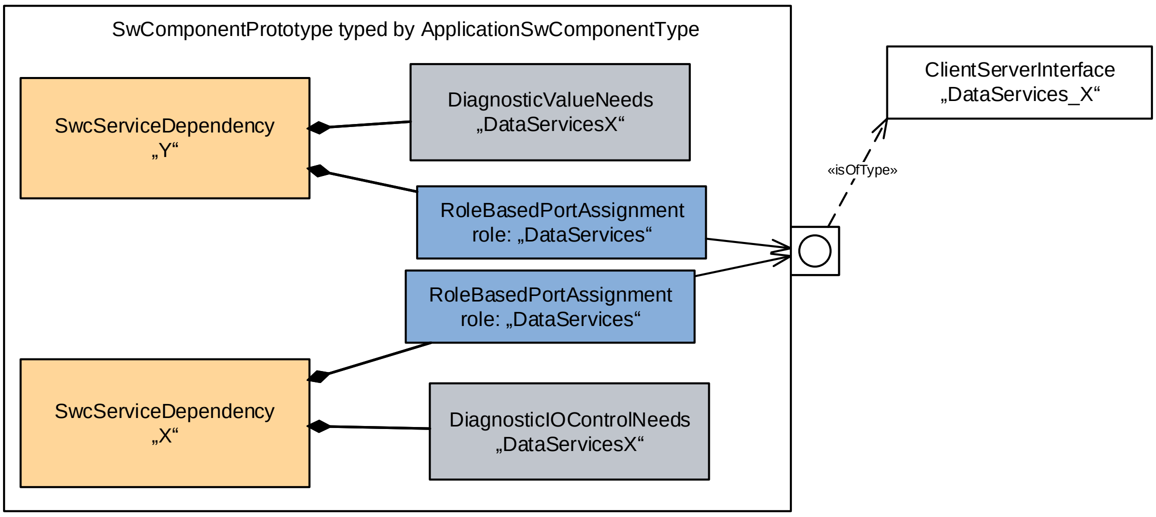


Figure 7.45: Two SwcServiceDependencys referencing one PortPrototype

[TPS_SWCT_01005] Usage of SwcServiceDependencys for vendor-specific services ⌈SwcServiceDependencys can also be used for vendor-specific services. In this case the SwcServiceDependency shall not contain any of the standardized ServiceNeeds. For this purpose the VendorSpecificServiceNeeds is available.⌋*(RS_SWCT_02060)*

[TPS_SWCT_01833] Semantics of ServiceDependency.diagnosticRelevance ⌈The attribute ServiceDependency.diagnosticRelevance can be used to indicate a diagnostic relevance of the ServiceDependency, especially for ServiceDependencys that are not related to diagnostics, but which are still required for the integration of the diagnostic stack.⌋*(RS_SWCT_02060)*

One example for the usage of ServiceDependency.diagnosticRelevance would be the communication between the application software and the BswM, i.e. in the context of a SwcServiceDependency that aggregates a BswMgrNeeds. This communication may be relevant for the configuration of so-called "mode conditions" in the Dcm.

Please note that the modeling of attribute is very general and can be applied to every foreseeable service configuration. However, the only know use case that is actually relevant is the mentioned communication between application software and the BswM.

In order to not open the door for the uncontrolled usage of uncharted territory (that may nevertheless later be explored, use case by use case), restrictions apply for the usage of ServiceDependency.diagnosticRelevance, as documented in [constr_10032].

[constr_10032] Restrictions for the usage of ServiceDependency.diagnosticRelevance ⌈The attribute ServiceDependency.diagnosticRelevance shall only be used for a SwcServiceDependency that aggregates a BswMgrNeeds **at the time when the RTE is generated**.⌋*()*

7.12 Variation Point Proxy

[TPS_SWCT_01370] VariationPointProxy ⌈Description of variability inside a software-component may exist in two different levels of abstraction:

- A **structural** variation point affects the existence or non-existence of structural model elements. A structural variation point is modeled by means of the meta-class VariationPoint.
- A **functional** variation point affects solely the functionality in the implementation (read: source code) of the software-component. A functional variation point is modeled by means of the meta-class VariationPointProxy.

In other words, this enables the developer of a software-component to implement variability that is limited to the software-component's functionality. This kind of variability is resolved

- by a code generator (bindingTime = codeGenerationTime)

- by the preprocessor (bindingTime = preCompileTime).
- as a post-build value evaluation (in this case postBuildValueAccess and postBuildVariantCondition shall exist).

⌋(*RS_SWCT_03100*)

Class	**VariationPointProxy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::VariantHandling			
Note	The VariationPointProxy represents variation points of the C/C++ implementation. In case of bindingTime = compileTime the RTE provides defines which can be used for Pre Processor directives to implement compileTime variability.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	BswInternalBehavior.variationPointProxy, SwcInternalBehavior.variationPointProxy			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
conditionAccess	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the Variation Point.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association to ImplementationDataType shall be taken as an implementation hint by the RTE generator.
postBuildValue Access	PostBuildVariant Criterion	0..1	ref	This represents the applicable PostBuildVariantCriterion in the context of a VariationPointProxy. Note that the technical details how to access the particular postBuildValueAccess are still considered internal to the RTE and are consequently not standardized.
postBuildVariant Condition	PostBuildVariant Condition	*	aggr	This represents that applicable PostBuoldVariant Condition in the context of aVariationPointProxy.
valueAccess	AttributeValueVariation Point	0..1	aggr	This value acts as Binding Function for the VariationPoint.

Table 7.63: VariationPointProxy

Class	<<atpMixedString>> **ConditionByFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents a condition which is computed based on system constants according to the specified expression. The expected result is considered as boolean value. The result of the expression is interpreted as a condition. • "0" represents "false"; • a value other than zero is considered "true"			
Base	*ARObject*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Aggregated by	VariationPoint.swSyscond, VariationPointProxy.conditionAccess			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bindingTime	BindingTimeEnum	1	attr	This attribute specifies the point in time when condition may be evaluated at earliest. At this point in time all referenced system constants shall have a value. **Tags:**xml.attribute=true

Table 7.64: ConditionByFormula

Class	PostBuildVariantCriterion			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. Tags:atp.recommendedPackage=PostBuildVariantCriterions			
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table 7.65: PostBuildVariantCriterion

Class	PostBuildVariantCondition			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies the value which shall be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they shall all match to bind the variation point. In other words binding can be represented by (criterion1 == value1) && (condition2 == value2) ...			
Base	*ARObject*			
Aggregated by	VariationPoint.postBuildVariantCondition, VariationPointProxy.postBuildVariantCondition			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
matching Criterion	PostBuildVariant Criterion	1	ref	This is the criterion which needs to match the value in order to make the PostbuildVariantCondition to be true.
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 7.66: PostBuildVariantCondition

Please note that in the first two cases of the second bullet list in [TPS_SWCT_01370] the evaluation of `conditionAccess` shall replace the formula by the result.

The name `VariationPointProxy` was motivated by the fact that it represents a model element that is not directly related to the **structure** but to the code and from this point of view acts as a proxy to the **functional** variation existing in the code.

The consequence of the two levels of abstraction is that (from a model processing point of view) it would be possible to bind all structural variation points entirely while keeping some or all of the functional variation points unbound. This is an explanation for the existence of [TPS_SWCT_01371].

[TPS_SWCT_01371] **`VariationPointProxy` vs. `VariationPoint`** ⌈The difference between a `VariationPoint` and a `VariationPointProxy` is that if during the process of binding the formula evaluates to 0 the `VariationPointProxy` remains in the model while the `VariationPoint` as well as its owner is removed from the model.⌋*(RS_SWCT_03100)*

Nevertheless, the binding of the variability is described by the means of `SwSystemconstantValueSet`s and `PostBuildVariantCriterionValueSet`s.

[TPS_SWCT_01448] Pre-defined values for the category of VariationPointProxy ⌈AUTOSAR pre-defines two possible values for the category of VariationPointProxy. The meaning of the values, however, depends on the particular modeling of individual VariationPointProxys, see [TPS_SWCT_01370].

VALUE In the "pre-build" case this means that valueAccess shall yield an integer literal. In the "post-build" case, on the other hand, this means that postBuildValueAccess shall yield an integer value conform with the implementationDataType.

In this context, [constr_1388] applies.

CONDITION In this case it is **possible** (though not mandatory) to define a VariationPointProxy that actually works in a combination of the "pre-build" and "post-build" scenario.

In other words, in the "pre-build" case conditionAccess shall yield a *boolean* value and in the "post-build" case postBuildVariantCondition shall also yield a *boolean* value.

An *and* operator shall be applied to all boolean values returned by conditionAccess and the collection of postBuildVariantCondition in order to yield the actual result of the condition. [TPS_GST_00259] and [SWS_Rte_08069] apply.

For the postBuildVariantCondition an implicit reference to the Platform Data Type boolean shall be assumed.

In contrast to the value VALUE it is possible to define a VariationPointProxy that uses **both** conditionAccess **and** postBuildVariantCondition.

⌋*(RS_SWCT_03100)*

[constr_1388] VariationPointProxy of category VALUE shall not mix "pre-build" and "post-build" use-cases ⌈If the value of category of the VariationPointProxy is set to VALUE then there can only be one value yield from the evaluation of a VariationPointProxy. In other words, a VariationPointProxy of category VALUE shall not mix the "pre-build" and "post-build" use-cases.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1389] Restriction regarding the value of category of VariationPointProxy.implementationDataType ⌈VariationPointProxy.implementationDataType shall **not** be of category STRUCTURE, ARRAY, UNION, FUNCTION_REFERENCE, and DATA_REFERENCE.

The VariationPointProxy.implementationDataType shall be of category VALUE or TYPE_REFERENCE that, after all references are resolved, yields an ImplementationDataType of category VALUE.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋()

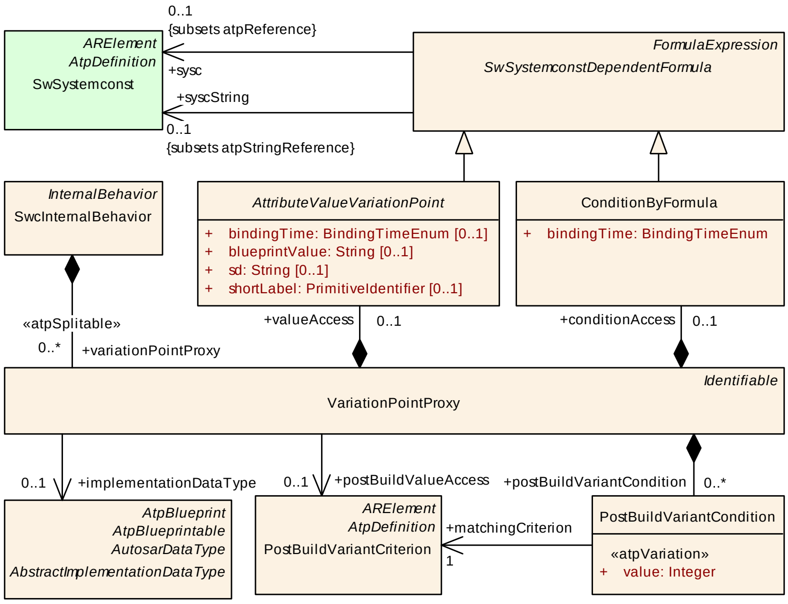


Figure 7.46: VariationPointProxy

[TPS_SWCT_01372] bindingTime = preCompileTime ⌈In case of bindingTime = preCompileTime the RTE provides macro definitions that can be used for pre-processor directives to implement preCompileTime variability in C/C++ code.⌋*(RS_-SWCT_03100)*

The generation of macro definitions implies the creation of suffixes that indicate the type of the constant value (e.g. "u" for "unsigned") defined in the context of the macro. This is especially important where the MISRA [35] rules need to be observed.

According to the definition of the AUTOSAR formula language (see TPS Generic Structure Template [11]), numerical values yielded by a formula expression will not have a suffix that clarifies the data type associated with the constant value.

It is therefore in general necessary to provide a formal association between the value yielded by a formula expression and an ImplementationDataType that can provide the information about the suffix to be applied in the specific case.

[TPS_SWCT_01858] Optional existence of reference VariationPointProxy.implementationDataType if the bindingTime is set to preCompileTime ⌈If the value of attribute bindingTime of VariationPointProxy.valueAccess is set to the value preCompileTime, then the creation of a reference to an ImplementationDataType in the role VariationPointProxy.implementationDataType shall be taken to determine the suffix to qualify the numerical value yielded from the formula expression for the creation of a preprocessor macro.⌋*(RS_SWCT_03100)*

For more clarification, the actual information about the suffix shall be derived from the value of ImplementationDataType.swDataDefProps.baseType.baseTypeDefinition.baseTypeEncoding and baseTypeSize.

Class	<<atpMixedString>> ***AttributeValueVariationPoint*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This class represents the ability to derive the value of the Attribute from a system constant (by Sw SystemconstDependentFormula). It also provides a bindingTime.			
Base	*ARObject*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses	*AbstractEnumerationValueVariationPoint*, *AbstractNumericalVariationPoint*, BooleanValueVariationPoint, FloatValueVariationPoint, IntegerValueVariationPoint, PositiveIntegerValueVariationPoint, TimeValue ValueVariationPoint, UnlimitedIntegerValueVariationPoint			
Aggregated by	VariationPointProxy.valueAccess			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bindingTime	BindingTimeEnum	0..1	attr	This is the binding time in which the attribute value needs to be bound. If this attribute is missing, the attribute is not a variation point. In particular this means that It needs to be a single value according to the type specified in the pure model. It is an error if it is still a formula. **Tags:**xml.attribute=true
blueprintValue	String	0..1	attr	This represents a description that documents how the value shall be defined when deriving objects from the blueprint. **Tags:**xml.attribute=true
sd	String	0..1	attr	This special data is provided to allow synchronization of Attribute value variation points with variant management systems. The usage is subject of agreement between the involved parties. **Tags:**xml.attribute=true
shortLabel	PrimitiveIdentifier	0..1	attr	This allows to identify the variation point. It is also intended to allow RTE support for CompileTime Variation points. **Tags:**xml.attribute=true

Table 7.67: AttributeValueVariationPoint

[TPS_SWCT_01373] RTE generator shall evaluate the `SwSystemconstDependentFormula` ⌈It is in the scope of the RTE generator to evaluate the `SwSystemconstDependentFormula` which has a higher precedence than the standard C Preprocessor and to provide the resulting values to the software-component's implementation.⌋*(RS_SWCT_03100)*

For further details (beyond the statements made in [TPS_SWCT_01372] and [TPS_SWCT_01373]) about the impact of the existence of a `VariationPointProxy` on the RTE please refer to [2].

Please note that the usage of attributes of meta-class `VariationPointProxy` is not arbitrarily possible but subject to conditions. In particular, there are certain use-cases that dictate how and with which multiplicity attributes of `VariationPointProxy` shall be used.

In particular, the applicable use-cases are defined by a combination of the binding time, i.e. *PreBuild* (all pre-build binding times are summarized as *PreBuild*) vs. *PostBuild*, and the value of `VariationPointProxy.category` (the details are explained in [constr_1253], respectively).

[constr_1253] Allowed multiplicities for attributes of VariationPointProxy depending on the applicable binding time and the value of VariationPointProxy.category ⌈

BindingTime	category	Allowed Attribute Multiplicity
PreBuild	VALUE	valueAccess [1], implementationDataType [0..1]
	CONDITION	conditionAccess [1]
PostBuild	VALUE	postBuildValueAccess [1], implementationDataType [1]
	CONDITION	postBuildVariantCondition [1..*], conditionAccess [0..1]

This rule shall be imposed **at the time when the contract phase generation is executed**.

⌋*()*

For clarification, the multiplicities of attributes of meta-class VariationPointProxy that are **not** explicitly mentioned in a given row of the table contained in [constr_1253] shall be interpreted as [0].

8 Implementation

Previous versions of this document contained a comprehensive description of the meta-class `Implementation`. This meta-class still exists but the description of most of its content has been moved to another document, in particular the specification of the `Basic Software Module Description Template` [6].

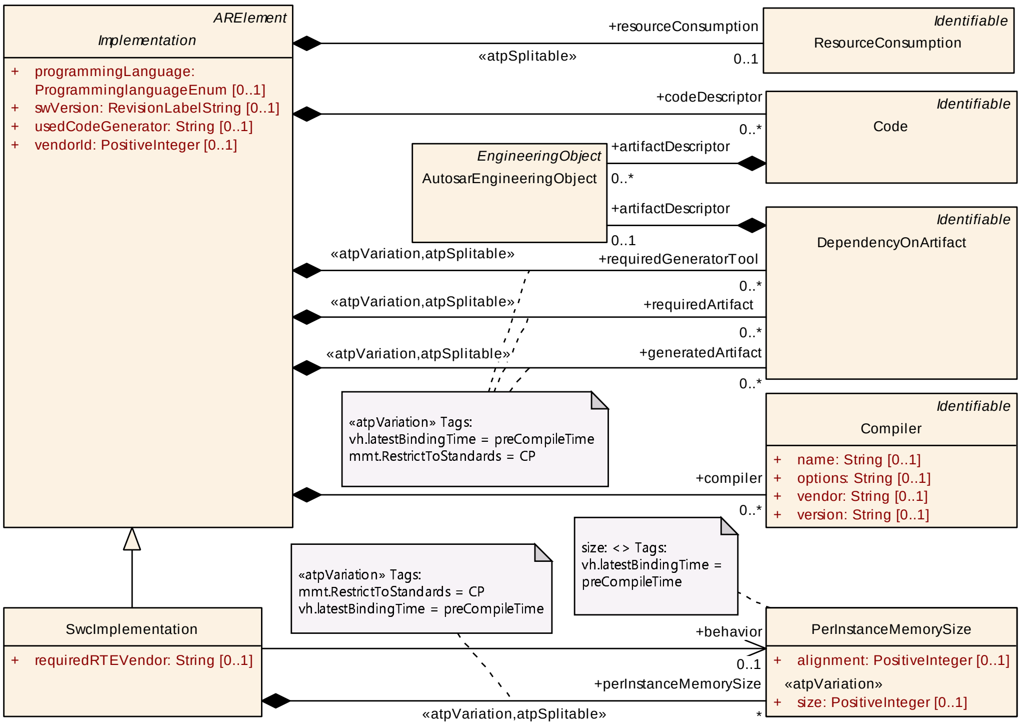


Figure 8.1: Implementation part specific to the Software Component Template

Please note that the `Software Component Template` and the `Basic Software Module Description Template` share the content of `Implementation`. However, the semantics of `Implementation` is closer to the `Basic Software Module Description Template`.

Nevertheless, there is still content strictly related to the `Software Component Template`. This part of `Implementation` consisting of `SwcImplementation` (see Figure 8.1) remains in this document.

Class	***Implementation*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	BswImplementation, SwcImplementation			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Implementation*** (abstract)			
buildAction Manifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=buildActionManifest.buildActionManifest, buildActionManifest.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released
generated Artifact	DependencyOnArtifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=generatedArtifact.shortName, generated Artifact.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is <<atpSplitable>> because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=mcSupport
programming Language	Programminglanguage Enum	0..1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredArtifact.shortName, required Artifact.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
required GeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredGeneratorTool.shortName, required GeneratorTool.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
resource Consumption	ResourceConsumption	0..1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=resourceConsumption.shortName

▽

△

Class	***Implementation*** (abstract)			
swcBsw Mapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or Bsw Implementtion or for both.
swVersion	RevisionLabelString	0..1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.
usedCode Generator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table 8.1: Implementation

[constr_1966] Existence of attribute `Implementation.swVersion` ⌈For each `Implementation`, attribute `swVersion` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1967] Existence of attribute `Implementation.vendorId` ⌈For each `Implementation`, attribute `vendorId` shall exist **`at the time when the RTE is generated`**.⌋*()*

[constr_1968] Existence of attribute `Implementation.codeDescriptor` ⌈For each `Implementation`, at least one aggregation of `Code` in the role `codeDescriptor` shall exist **`at the time when the RTE is generated`**.⌋*()*

Class	**Code**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	A generic code descriptor. The type of the code (source or object) is defined via the category attribute of the associated engineering object.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Aggregated by	*Implementation*.codeDescriptor			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
artifact Descriptor	AutosarEngineering Object	*	aggr	Refers to the artifact belonging to this code descriptor.
callbackHeader	ServiceNeeds	*	ref	The association callbackHeader describes in which header files the function declarations of callback functions are provided to a service module. With this information the service module can include the appropriate header files in its configuration files.

Table 8.2: Code

Class	SwcImplementation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software. **Tags:**atp.recommendedPackage=SwcImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
behavior	SwcInternalBehavior	0..1	ref	The internal behavior implemented by this Implementation.
perInstance MemorySize	PerInstanceMemory Size	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstance MemorySize is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects, in this case PerInstanceMemory. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceMemorySize, perInstance MemorySize.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
required RTEVendor	String	0..1	attr	Identify a specific RTE vendor. This information is potentially important at the time of integrating (in particular: linking) the application code with the RTE. The semantics is that (if the association exists) the corresponding code has been created to fit to the vendor-mode RTE provided by this specific vendor. Attempting to integrate the code with another RTE generated in vendor mode is in general not possible.

Table 8.3: SwcImplementation

[constr_1969] Existence of attribute `SwcImplementation.behavior` ⌈For each `SwcImplementation`, attribute `behavior` shall exist **at the time when the RTE is generated**.⌋*()*

Class	PerInstanceMemorySize			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	Resources needed by the allocation of PerInstanceMemory for each SWC instance. Note that these resources are not covered by an ObjectFileSection, because they are supposed to be allocated by the RTE.			
Base	*ARObject*			
Aggregated by	SwcImplementation.perInstanceMemorySize			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
alignment	PositiveInteger	0..1	attr	Required alignment (1,2,4,...) of the referenced Per InstanceMemory. Unit: byte.
perInstance Memory	PerInstanceMemory	0..1	ref	This represents the referenced PerInstanceMemory.

▽

△

Class	PerInstanceMemorySize			
size	PositiveInteger	0..1	attr	Size (in bytes) of the reference perInstanceMemory. The aggregation of PerInstanceMemorySize is subject to variability with the purpose to support variability in the software components implementations. Different algorithms in the implementation might require a different PerInstanceMemorySize. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 8.4: PerInstanceMemorySize

[constr_1970] Existence of attribute PerInstanceMemorySize.alignment ⌈For each PerInstanceMemorySize, attribute alignment shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1971] Existence of attribute PerInstanceMemorySize.perInstanceMemory ⌈For each PerInstanceMemorySize, the reference to PerInstanceMemory in the role perInstanceMemory shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1972] Existence of attribute PerInstanceMemorySize.size ⌈For each PerInstanceMemorySize, attribute size shall exist **at the time when the RTE is generated**.⌋*()*

9 Mode Management

In general, the Software Component Template doesn't define the kind of modes that shall be supported by State Managers or software-components explicitly. However the Software Component Template provides generic mechanisms for describing modes.

In this section the general relationship between modes, interfaces, and software-components is discussed.

The assumption from the software-component point of view is that State Managers are using a Standardized AUTOSAR PortInterface[1] to influence the SwComponentType and also provide a PortInterface to get requests and confirmations from the SwComponentType.

They will be implemented as AUTOSAR services and be part of the Basic Software on each ECU. The actual modes a State Manager provides will have to be standardized as well to allow compatibility between software-components.

It is also possible to define a mode manager in the Application Software and the same functionality is supported as for mode managers implemented in the Basic Software.

[TPS_SWCT_01581] Communication patterns for mode-related communication ⌈Mode-related communication shall implement a 1:1 or 1:n scenario but the creation of an n:1 configuration shall be considered invalid.⌋*(RS_SWCT_03200, RS_SWCT_03110)*

As a consequence of [TPS_SWCT_01581], [constr_1101] is formulated.

[constr_1101] Mode-related communication ⌈An RPortPrototype typed by ModeSwitchInterface shall not be referenced by more than one SwConnector **at the time when the RTE is generated**.⌋*()*

9.1 Declaration of Modes

The SW-Component Template provides some simple means to define collections of modes.

[TPS_SWCT_01071] ModeDeclaration ⌈The name of the mode is the most important attribute that has to be provided for each ModeDeclaration. The ModeDeclarations are grouped together within the ModeDeclarationGroup.⌋*(RS_SWCT_03200, RS_SWCT_03110)*

[TPS_SWCT_01067] Initial mode ⌈The initialMode is active before any mode switches occurred.⌋*(RS_SWCT_03200)*

This is shown in Figure 9.1

[1] See also AUTOSAR Glossary for "Standardized AUTOSAR Interface".

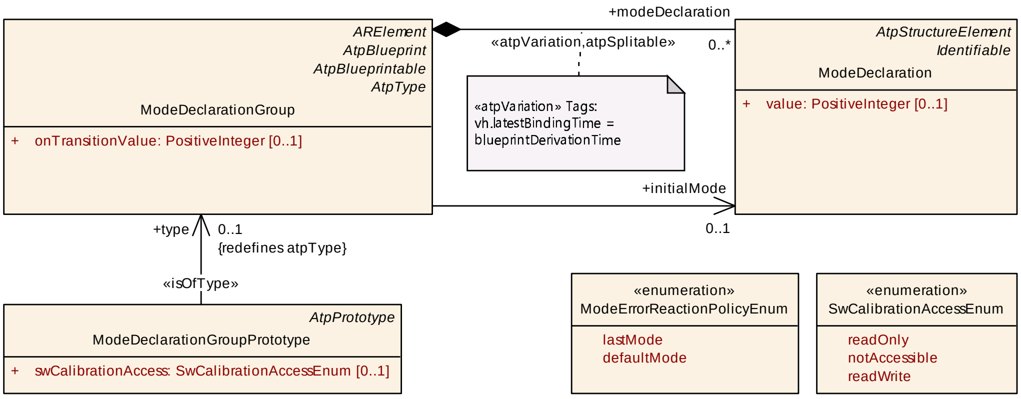


Figure 9.1: ModeDeclaration

The class ModeDeclarationGroup has been introduced to support the grouping of modes and (on M1 level) to provide predefined sets of modes that could be standardized and re-used. The set of modes eventually defines a flat (i.e. no hierarchical states) state-machine where only one mode can be active at a given point in time.

Again, please note that the actual definition of modes and their relationship is not in the responsibility of this document. In other words: the definition of modes represents M1 artifacts whereas this document is limited to describing M2 model elements.

Both ModeDeclaration and ModeDeclarationGroup own attributes that facilitate the generation of C source code from the formal definition.

[TPS_SWCT_01008] Definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration ⌈The attributes ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue allow for the definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration and ModeDeclarationGroup in the source code.⌋*(RS_SWCT_03200)*

[constr_1399] Standardized values of ModeDeclarationGroup.category ⌈The AUTOSAR standard defines the following values of the attribute ModeDeclarationGroup.category with a standardized meaning:

- EXPLICIT_ORDER
- ALPHABETIC_ORDER

[TPS_SWCT_01010] defines the meaning of these values.

It is **not allowed** to define any custom or project-specific value of the attribute ModeDeclarationGroup.category.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

As the attributes ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue are optional the following rule applies:

[constr_1298] Existence of attributes if category of a ModeDeclarationGroup is set to EXPLICIT_ORDER ⌈The attributes ModeDeclarationGroup.onTransitionValue and ModeDeclaration.value (for each ModeDeclaration) shall be set if the category of a ModeDeclarationGroup is set to EXPLICIT_ORDER.

This rule shall be imposed **at the time when the contract phase generation is executed**.⌋*()*

[constr_1299] Existence of attributes if category of a ModeDeclarationGroup is set to other than EXPLICIT_ORDER ⌈The attributes ModeDeclarationGroup.onTransitionValue or ModeDeclaration.value (for any ModeDeclaration) shall **not** be set if the category of a ModeDeclarationGroup is set to any value **other than** EXPLICIT_ORDER **at the time when the contract phase generation is executed**.⌋*()*

[constr_1181] Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue ⌈The numerical values used to define the value attributes and the onTransitionValue attribute of a ModeDeclarationGroup shall not overlap **at the time when the contract phase generation is executed**.⌋*()*

In other words, it is not allowed that the values of two value attributes within one ModeDeclarationGroup have the same numerical value. Neither is it allowed that the numerical value of the ModeDeclarationGroup.onTransitionValue attribute and the numerical value of one of the corresponding value attributes are identical.

[TPS_SWCT_01009] The numerical values used to define the values of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue can be arbitrarily defined ⌈As long as the constraints [constr_1181], [constr_1298], and [constr_1299] are fulfilled, the numerical values used to define the values of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue can be arbitrarily defined. The numerical values are not required to be consecutive. Gaps are positively allowed.⌋*(RS_SWCT_03200)*

Example: the following example of a set of numerical values fulfills all requirements on the definition of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue: {1,2, 5, 100}.

Please note that the ability to define ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue introduces second heuristics for "ordering" ModeDeclarations. If ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue are not defined the assignment of numerical values to the representations of individual ModeDeclarations it is up to the RTE generator to come up with the applicable numerical values.

[TPS_SWCT_01010] categorys for the definition of a ModeDeclarationGroup ⌈In order to support a clear separation between the two possible ways to influence the definition of the programmatic representation of ModeDeclarations two categorys shall be defined for the definition of a ModeDeclarationGroup.

- The value of category of a ModeDeclarationGroup shall be set to EXPLICIT_ORDER if it is intended to control the source code generation by means of the values of the attributes ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue.
- The value of category of a ModeDeclarationGroup shall be set to ALPHABETIC_ORDER if it is intended to let the RTE generator control the source code generation according to the alphabetical sorting.

⌋*(RS_SWCT_03200)*

More information regarding this aspect can be found in [SWS_Rte_02568].

[TPS_SWCT_01011] Default category of a ModeDeclarationGroup ⌈For reasons of backwards-compatibility with previous releases of AUTOSAR the default value of the category of a ModeDeclarationGroup shall be ALPHABETIC_ORDER.⌋*(RS_SWCT_03200)*

Class	**ModeDeclaration**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeDeclarationGroup.modeDeclaration			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
value	PositiveInteger	0..1	attr	The RTE shall take the value of this attribute for generating the source code representation of this Mode Declaration.

Table 9.1: ModeDeclaration

Class	**ModeDeclarationGroup**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:**atp.recommendedPackage=ModeDeclarationGroups			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initialMode	ModeDeclaration	0..1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.
mode Declaration	ModeDeclaration	*	aggr	The ModeDeclarations collected in this ModeDeclaration Group. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeDeclaration.shortName, mode Declaration.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
modeManager ErrorBehavior	ModeErrorBehavior	0..1	aggr	This represents the ability to define the error behavior expected by the mode manager in case of errors on the mode user side (e.g. terminated mode user).

▽

△

Class	ModeDeclarationGroup			
modeTransition	ModeTransition	*	aggr	This represents the avaliable ModeTransitions of the ModeDeclarationGroup
modeUserError Behavior	ModeErrorBehavior	0..1	aggr	This represents the definition of the error behavior expected by the mode user in case of errors on the mode manager side (e.g. terminated mode manager).
onTransition Value	PositiveInteger	0..1	attr	The value of this attribute shall be taken into account by the RTE generator for programmatically representing a value used for the transition between two statuses.

Table 9.2: ModeDeclarationGroup

[constr_1973] Existence of attribute ModeDeclarationGroup.initialMode ⌈For each ModeDeclarationGroup, the reference to ModeDeclaration in the role initialMode shall exist **at the time when the contract phase generation is executed**.⌋*()*

[constr_1974] Existence of attribute ModeDeclarationGroup.modeDeclaration ⌈For each ModeDeclarationGroup, at least one ModeDeclaration shall be aggregated in the role modeDeclaration **at the time when the contract phase generation is executed**.⌋*()*

[TPS_SWCT_01450] Semantics of a ModeTransition ⌈In addition to the ability to specify ModeDeclarations within a ModeDeclarationGroup it is also feasible to define possible transitions between ModeDeclarations within the given ModeDeclarationGroup. This can be done by means of aggregation ModeTransition at ModeDeclarationGroup in the role modeTransition.⌋*(RS_SWCT_03200)*

More details are explained in Figure 9.2.

[TPS_SWCT_01451] Relations between ModeTransition and ModeDeclaration ⌈ModeTransition has two associations with the multiplicity 1 to ModeDeclaration:

- The reference enteredMode denotes a ModeDeclaration that can be entered as part of the enclosing ModeTransition.
- The reference exitedMode denotes a ModeDeclaration that can be exited as part of the enclosing ModeTransition.

⌋*(RS_SWCT_03200)*

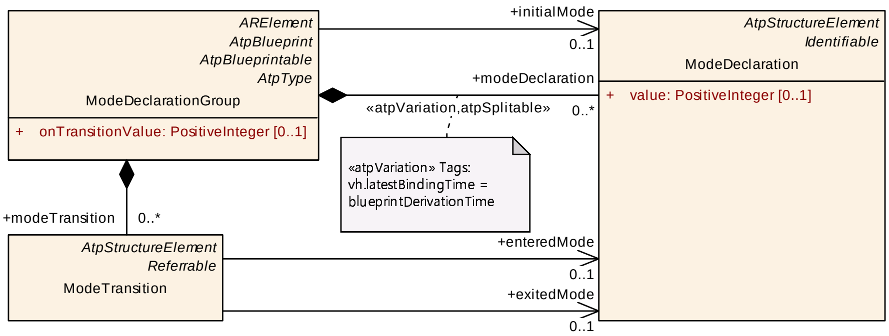


Figure 9.2: ModeTransition

[constr_1193] ModeDeclaration shall be referenced by at least one ModeTransition in the role enteredMode ⌈For each ModeDeclaration at least one ModeTransition shall reference the ModeDeclaration in the role enteredMode.

This constraint shall apply **at the time when the RTE is generated only** if there is at least one ModeTransition defined in the context of the enclosing ModeDeclarationGroup and it shall **not** apply to the initialMode.⌋*()*

For clarification, the ModeDeclarationGroup.initialMode does not need to be referenced by an enteredMode because by identifying this ModeDeclaration in the role initialMode it is clear that the ModeDeclaration will be entered at least once.

Class	**ModeTransition**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	This meta-class represents the ability to describe possible ModeTransitions in the context of a Mode DeclarationGroup.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeDeclarationGroup.modeTransition			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
enteredMode	ModeDeclaration	0..1	ref	This represents the entered model of the ModeTransition.
exitedMode	ModeDeclaration	0..1	ref	This represents the exited mode of the ModeTransition

Table 9.3: ModeTransition

[constr_1975] Existence of attribute ModeTransition.enteredMode ⌈For each ModeTransition, the reference to ModeDeclaration in the role enteredMode shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1976] Existence of attribute ModeTransition.exitedMode ⌈For each ModeTransition, the reference to ModeDeclaration in the role exitedMode shall exist **at the time when the RTE is generated**.⌋*()*

9.2 Modes and Events

[TPS_SWCT_01376] Software-components need to be capable of reacting to state changes ⌈Software-components need to be capable of reacting to state changes issued by some Mode Manager and adopt their behavior to the new situation.⌋*(RS_SWCT_03110)*

Such a mode dependent software-component is shown in Figure 9.3.

[TPS_SWCT_01077] Configure the response to mode changes ⌈Since the behavior of AtomicSwComponentTypes is mainly determined by the RunnableEntitys contained in the SwcInternalBehavior, it is necessary to configure the response to mode changes on the level of RunnableEntitys.⌋*(RS_SWCT_03120)*

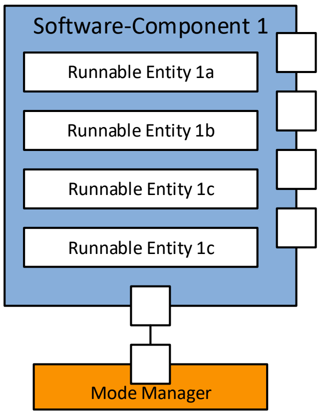


Figure 9.3: State Managers and software-components

Figure 9.4 shows an excerpt of the meta-model illustrating how the relationship between the current mode and the SwcInternalBehavior of the AtomicSwComponentType can be described.

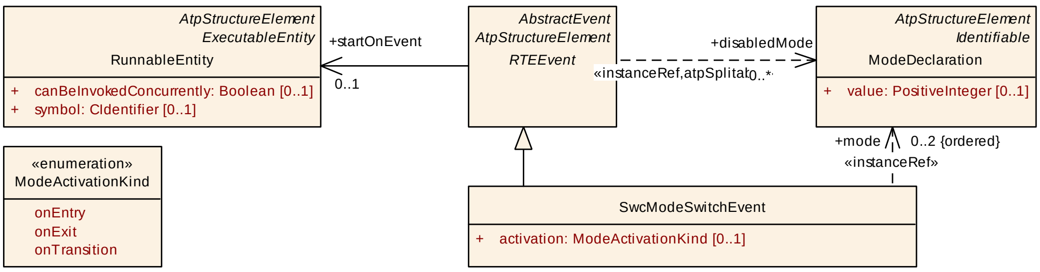


Figure 9.4: Modes and events

[TPS_SWCT_01377] Two mechanisms to define how SwcInternalBehavior should interact with the mode management ⌈A AtomicSwComponentType can use two mechanisms to define how its SwcInternalBehavior should interact with the mode management.⌋*(RS_SWCT_03110)*

Both mechanisms are visible in Figure 9.4.

[TPS_SWCT_01378] AtomicSwComponentType can define an SwcModeSwitchEvent to execute RunnableEntity ⌈Using the first mechanism, an AtomicSwComponentType can define an SwcModeSwitchEvent to specify that a particular RunnableEntity shall be started whenever a mode is entered, exited, or a transition between two specified modes occurs.⌋*(RS_SWCT_03110)*

[constr_4003] Semantics of SwcModeSwitchEvent ⌈If the value of SwcModeSwitchEvent.activation is onTransition, then SwcModeSwitchEvent shall refer to two different ModeDeclarations belonging to the same instance of ModeDeclarationGroup.

Their order defines the direction of the transition from one mode into another. In all other cases SwcModeSwitchEvent shall refer to exactly one ModeDeclaration.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[constr_1195] SwcModeSwitchEvent and the definition of ModeTransition ⌈For each pair of ModeDeclarations referenced by a SwcModeSwitchEvent with attribute activation set to onTransition a ModeTransition shall be defined in the corresponding direction (i.e. from exitedMode to enteredMode). This constraint shall only apply **at the time when the RTE is generated** if the respective ModeDeclarationGroup defines at least one modeTransition.⌋*()*

[TPS_SWCT_01379] AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode ⌈Using the second mechanism, the AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode.

That is, RTEEvents without an association in the role disabledMode are processed regularly according to their definition.

RTEEvents with the optional association disabledMode have the additional limitation that the associated RunnableEntity is *not* started when the ModeDeclaration referenced as disabledMode is active.⌋*(RS_SWCT_03110)*

The mechanisms discussed so far have to be applied for the SwcInternalBehavior on the receiver side of mode switches. Since mode switches are received via PortPrototypes the following constraints apply:

[TPS_SWCT_01380] Mode management behavior on the sender side ⌈On the sender side, a RunnableEntity shall have ModeSwitchPoints that eventually associate a RunnableEntity with the specific ModeDeclarationGroups which it manages.⌋*(RS_SWCT_03110)*

For more information, please refer to Figure 9.5.

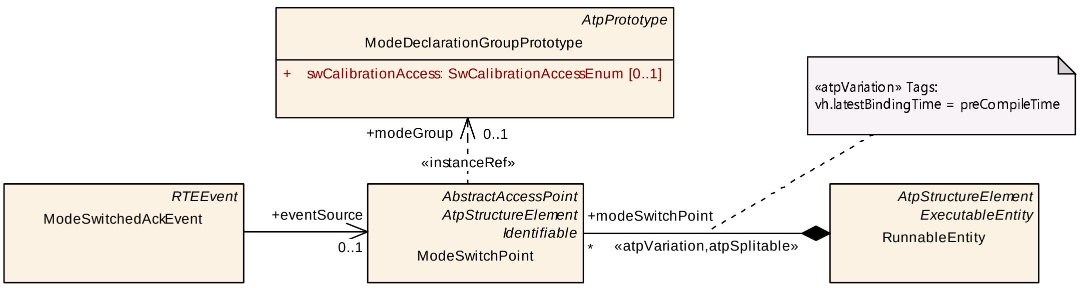


Figure 9.5: ModeSwitchPoint

[constr_1778] Value of attribute modeSwitchPoint.returnValueProvision ⌈All RunnableEntity.modeSwitchPoint that refer to the same modeGroup shall define the identical value of attribute returnValueProvision **at the time when the contract phase generation is executed**.⌋*()*

Rationale for the existence of [constr_1778]: different RunnableEntitys could aggregate ModeSwitchPoints with a different configuration of attribute returnValueProvision.

However, in such a case it would not be possible to generate the corresponding RTE API because the API exists once per software-component and it is therefore indispensable to have all affected ModeSwitchPoints agree on the configuration of attribute returnValueProvision.

This relation is exemplarily (for the case of dataSendPoint.returnValueProvision) sketched in Figure 7.22.

Class	ModeSwitchPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeSwitchPoint is required by a RunnableEntity owned a Mode Manager. Its semantics implies the ability to initiate a mode switch.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.modeSwitchPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	0..1	iref	The mode declaration group that is switched by this runnable. **InstanceRef implemented by:** PModeGroupInAtomicSwcInstanceRef

Table 9.4: ModeSwitchPoint

[TPS_SWCT_01383] ModeSwitchPoint ⌈The ModeSwitchPoint also allows for the definition of a ModeSwitchedAckEvent if this is requested by the definition of the PPortPrototype. This RTEEvent is eventually owned by a mode manager to allow for getting confirmation of a mode change.⌋*(RS_SWCT_03110)*

[TPS_SWCT_01555] ModeSwitchedAckEvent is triggered by the RTE regardless ⌈The ModeSwitchedAckEvent is triggered by the RTE (for more details please refer to [2]) regardless which RunnableEntity has requested the mode switch notification, even if the Meta-Model implies a reference from ModeSwitchedAckEvent to a specific ModeSwitchPoint in the role eventSource.⌋*(RS_SWCT_03110)*

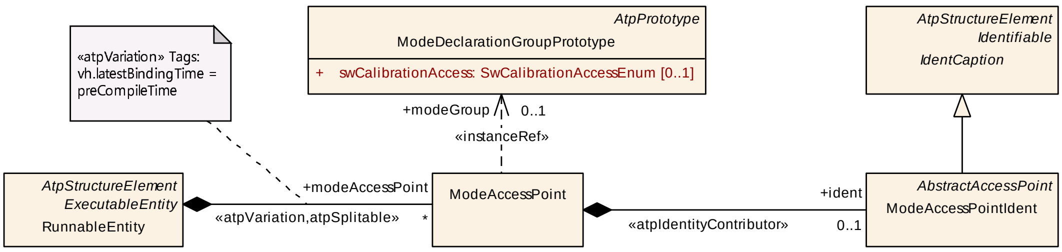


Figure 9.6: ModeAccessPoint

[constr_4012] Timeout of ModeSwitchedAckEvent ⌈The timeout value of a WaitPoint associated with a ModeSwitchedAckEvent shall be equal to the corresponding ModeSwitchedAckRequest.timeout **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01381] Read the currently active mode ⌈For *Mode Manager* and *Mode User* it might additionally be required to read the currently active mode. For that purpose, a RunnableEntity that requires read-access to the ModeDeclarationGroupPrototype's current mode has to define a ModeAccessPoint.⌋*(RS_-SWCT_03110)*

Class	ModeAccessPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeAccessPoint is required by a RunnableEntity owned by a Mode Manager or Mode User. Its semantics implies the ability to access the current mode (provided by the RTE) of a ModeDeclaration GroupPrototype's ModeDeclarationGroup.			
Base	*ARObject*			
Aggregated by	RunnableEntity.modeAccessPoint			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
ident	ModeAccessPointIdent	0..1	aggr	The aggregation in the role ident provides the ability to make the ModeAccessPoint identifiable. From the semantical point of view, the ModeAccessPoint is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-100
modeGroup	ModeDeclarationGroup Prototype	0..1	iref	The mode declaration group that is accessed by this runnable. **Tags:**xml.typeElement=true **InstanceRef implemented by:**ModeGroupInAtomicSwc InstanceRef

Table 9.5: ModeAccessPoint

[TPS_SWCT_01382] Mode switch requests are handled asynchronously by the RTE ⌈Mode switch requests are handled asynchronously by the RTE. Therefore, *Mode Managers* implementation might require reading back the current active mode to synchronize internally to the RTE. A ModeSwitchPoint does **not** automatically provide read access to the ModeDeclarationGroupPrototype's current mode.⌋*(RS_-SWCT_03110)*

[constr_1098] Mode switch and mode disabling ⌈A SwcModeSwitchEvent shall not simultaneously reference to the same ModeDeclaration in both the roles mode and disabledMode **at the time when the RTE is generated**.⌋*()*

If [constr_1098] would not apply, it might happen that a RunnableEntity would be triggered by a SwcModeSwitchEvent and, at the same time, it would be suppressed by the mode disabling.

9.3 Initialization / Finalization

The AUTOSAR standard shall support the execution of initialization code for every AtomicSwComponentType.

[TPS_SWCT_01384] Execution of initialization code for software-components ⌈Most AtomicSwComponentTypes will need to initialize by executing specific code;

this code shall complete before any other code in the component is executed. Data will be initializing to specific values before the "normal" application software is running.⌋ *(RS_SWCT_03110)*

[TPS_SWCT_01385] Execution of finalization code for software-components ⌈Most AtomicSwComponentTypes will need to finalize by calling specific code; this code shall complete before the functionality of the application software shut down (e.g. a motor drive in a start or end position).⌋ *(RS_SWCT_03110)*

[TPS_SWCT_01388] Initial modes of AtomicSwComponentTypes are defined by the initialMode ⌈The initial modes of AtomicSwComponentTypes are defined by the initialMode references of the required ModeDeclarationGroups. These modes are activated before any other mode activation has occurred. It is the responsibility of the RTE to activate all initial modes on a certain ECU.⌋ *(RS_SWCT_03110)*

For more details please refer to the specification of the SWS RTE [2].

9.4 Mode Error Behavior

With the advent of partitions in the AUTOSAR standard, it is important to consider the behavior of mode management with respect to the following scenarios:

- The partition of the mode manager is terminated.
- The partition of the mode user is terminated.

Whenever one of the two scenarios becomes reality, it is important to implement a stable reaction of both mode manager and mode user to the event. In addition, mode manager and mode user should be able to synchronize in terms of which mode shall apply as fast and seamless as possible.

For this purpose, additional modeling support has been defined such that the applicable ModeDeclarationGroup (which is part of the contract between mode manager and mode user) becomes the place where the policy towards a reaction to e.g. a partition restart is defined.

[TPS_SWCT_01530] Error behavior of mode manager and mode user ⌈The behavior in response to a mode manager getting out of sync with a mode user (because the partition of the mode user has been terminated) or vice versa (because the partition of the mode manager has been terminated) can be defined for the mode manager by means of the attribute ModeDeclarationGroup.modeManagerErrorBehavior and for the mode user by means of the attribute ModeDeclarationGroup.modeUserErrorBehavior.⌋ *(RS_SWCT_03110)*

[TPS_SWCT_01531] The semantics of ModeErrorReactionPolicyEnum ⌈The attribute ModeErrorBehavior.errorReactionPolicy shall be used to specify the behavior in the event of a mode error:

lastMode The last mode applicable before the event shall be assumed.

defaultMode This represents the ability to specify a dedicated mode that shall be made applicable. The identified ModeDeclaration could be identical to the ModeDeclarationGroup.initialMode but it can just as well be any other ModeDeclaration defined in the context of the enclosing ModeDeclarationGroup.

⌋*(RS_SWCT_03110)*

[TPS_SWCT_01532] The role of ModeErrorBehavior.defaultMode ⌈The attribute ModeErrorBehavior.defaultMode shall be used to identify the particular ModeDeclaration if ModeErrorBehavior.errorReactionPolicy is set to defaultMode.⌋*(RS_SWCT_03110)*

[constr_1263] Existence of ModeErrorBehavior.defaultMode ⌈The optional attribute ModeErrorBehavior.defaultMode **shall exist** if the value of the attribute ModeErrorBehavior.errorReactionPolicy is set to defaultMode.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

Please note that the modeling of the ModeErrorBehavior is depicted in Figure 7.11.

[TPS_SWCT_01533] ModeDeclarationGroup.initialMode shall be assumed in the absence of ModeDeclarationGroup.modeManagerErrorBehavior ⌈If the attribute ModeDeclarationGroup.modeManagerErrorBehavior is not defined it shall be assumed that the ModeDeclarationGroup.initialMode becomes applicable in case of the mode manager getting out of sync with a mode user (because the partition of the mode user has been terminated).⌋*(RS_SWCT_03110)*

[TPS_SWCT_01534] ModeDeclarationGroup.initialMode shall be assumed in the absence of ModeDeclarationGroup.modeUserErrorBehavior ⌈If the attribute ModeDeclarationGroup.modeUserErrorBehavior is not defined it shall be assumed that the ModeDeclarationGroup.initialMode becomes applicable in case of the mode user getting out of sync with a mode manager (because the partition of the mode manager has been terminated).⌋*(RS_SWCT_03110)*

Class	**ModeErrorBehavior**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	This represents the ability to define the error behavior in the context of mode handling.			
Base	*ARObject*			
Aggregated by	ModeDeclarationGroup.modeManagerErrorBehavior, ModeDeclarationGroup.modeUserErrorBehavior			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
defaultMode	ModeDeclaration	0..1	ref	This represents the ModeDeclaration that is considered the error mode in the context of the enclosing ModeDeclarationGroup.
errorReactionPolicy	ModeErrorReactionPolicyEnum	0..1	attr	This represents the ability to define the policy in terms of which default model shall apply in case an error occurs.

Table 9.6: ModeErrorBehavior

[constr_1977] Existence of attribute ModeErrorBehavior.errorReactionPolicy ⌈For each ModeErrorBehavior, the attribute errorReactionPolicy shall exist **at the time when the RTE is generated.**⌋()

Enumeration	**ModeErrorReactionPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	This represents the ability to specify the reaction on a mode error.
Aggregated by	ModeErrorBehavior.errorReactionPolicy
Literal	*Description*
defaultMode	This represents the ability to switch to the defaultMode in case of a mode error. **Tags:**atp.EnumerationLiteralIndex=0
lastMode	This represents the ability to keep the last mode in case of a mode error. **Tags:**atp.EnumerationLiteralIndex=1

Table 9.7: ModeErrorReactionPolicyEnum

[TPS_SWCT_01535] Mode manager reacts on mode error ⌈If the mode manager is getting out of sync with a mode user (because the partition of the mode user has been terminated) or vice versa (because the partition of the mode manager has been terminated) it shall be possible for the mode manager to react on such an event.

For this purpose the formal SwcModeManagerErrorEvent is defined that can be taken to e.g. trigger the execution of a RunnableEntity in response to an error with respect to mode switch communication.⌋*(RS_SWCT_03110)*

Class	**SwcModeManagerErrorEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when an error occurred during the handling of the referenced ModeDeclarationGroup Prototype.			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *RTEEvent*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.event			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
modeGroup	ModeDeclarationGroup Prototype	0..1	iref	This represents the ModeDeclarationGroupPrototype for which this SwcModeManagerErrorEvent is raised in case of an error. **InstanceRef implemented by:**PModeGroupInAtomic SwcInstanceRef

Table 9.8: SwcModeManagerErrorEvent

[constr_1978] Existence of attribute SwcModeManagerErrorEvent.modeGroup ⌈For each SwcModeManagerErrorEvent, the instance reference to ModeDeclaration in the role modeGroup shall exist **at the time when the RTE is generated.**⌋()

As mentioned in [constr_1075], it is possible to overrule the default compatibility rules by the definition of a PortInterfaceMapping.

In this case the demand for having identical definitions of ModeDeclarationGroup.modeUserErrorBehavior and ModeDeclarationGroup.modeManagerErrorBehavior is no longer valid.

However, there is one additional caveat to observe in this case. This affects the implementation of error behavior in case that several mode users are connected to a mode manager.

[TPS_SWCT_01536] Coherent behavior of all mode users in case of errors in the mode switch communication ⌈The behavior in case of errors with the communication of mode switches needs to be **coherent for all** connected mode users **especially** if the individual SwConnectors are legitimized by the existence of a PortInterfaceMapping.⌋*(RS_SWCT_03110)*

[TPS_SWCT_01541] Preferential selection of modeUserErrorBehavior ⌈The definition of mode error behavior on the provided side of shall be considered **dominant** over the definition of mode error behavior on the required side.

This means that a ModeSwitchInterface.modeGroup.type.modeUserErrorBehavior used to type an AbstractProvidedPortPrototype shall be considered **dominant** over the definition of a corresponding modeUserErrorBehavior and defined in the context of an AbstractRequiredPortPrototype.⌋*(RS_SWCT_03110)*

[TPS_SWCT_01542] Preferential selection of modeManagerErrorBehavior ⌈The definition of mode error behavior on the provided side of shall be considered **dominant** over the definition of mode error behavior on the required side.

This means that a ModeSwitchInterface.modeGroup.type.modeManagerErrorBehavior used to type an AbstractProvidedPortPrototype shall be considered **dominant** over the definition of a corresponding modeManagerErrorBehavior defined in the context of an AbstractRequiredPortPrototype.⌋*(RS_SWCT_03110)*

The consequence of [TPS_SWCT_01541] and [TPS_SWCT_01542] is that the **mode manager shall be considered the master of the definition of mode error behavior**.

Please note that the statements made in [TPS_SWCT_01541] is further underlined by [SWS_Rte_06795] and the statement made by [TPS_SWCT_01542] is further underlined by [SWS_Rte_06795].

The details of how the run-time behavior of mode manager and mode user shall look like in the event of the mode manager getting out of sync with a mode user (because the partition of the mode user has been terminated) or vice versa (because the partition of the mode manager has been terminated) as well as the applicable RTE APIs are explained in [2].

9.5 Summary Meta-Model Excerpt Related to Modes

Figure 9.7 provides an overview of all meta-model elements that have a direct relationship to the meta-classes involved in the modeling of mode switches.

To get the complete picture, it should be noted that also the concepts of `PortGroup`s (see 4.6) and `ServiceProxySwComponentType` (see 11.3) have a semantical relationship to mode management, though this is not expressed via relations in the meta-model.

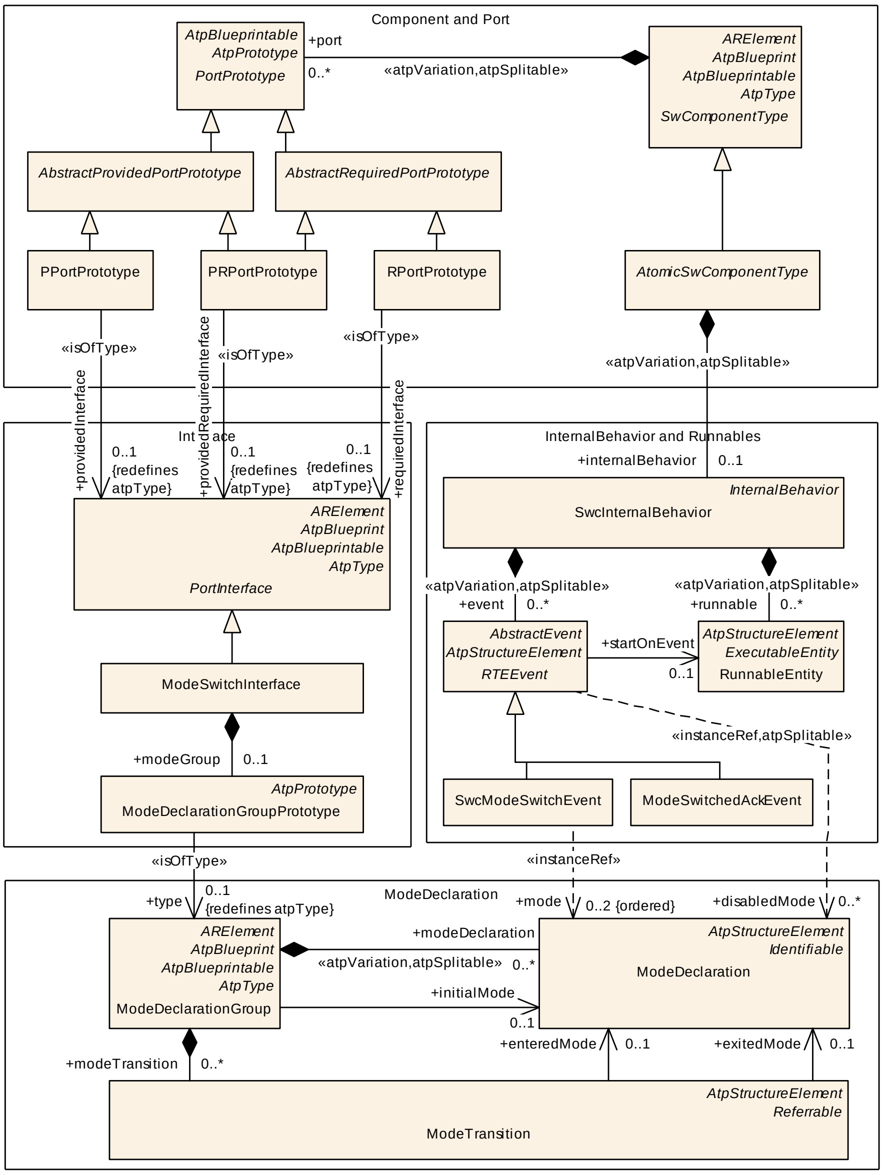


Figure 9.7: Summary meta-model excerpt related to modes

10 ECU Abstraction and Complex Drivers

10.1 Introduction

During the design of embedded systems there is one crucial point where the hardware and software have to be related. In AUTOSAR the `ECU Resource Template` describes the provided hardware resources.

On the other hand, the `Software Component Template` describes software generally without specific hardware in mind. But there are some places where both have to meet and fit.

One interface between hardware and software is discussed in the memory and execution time section of [6]. In this chapter the overall system view of the interface between sensors/actuators and software is described and the consequences for the `Software Component Template` are derived.

10.2 High Level Hardware and Software Architecture

The AUTOSAR concept defines a software architecture (see Figure 10.1) and within this layered architecture the interfaces between the hardware and the software are explicitly modeled.

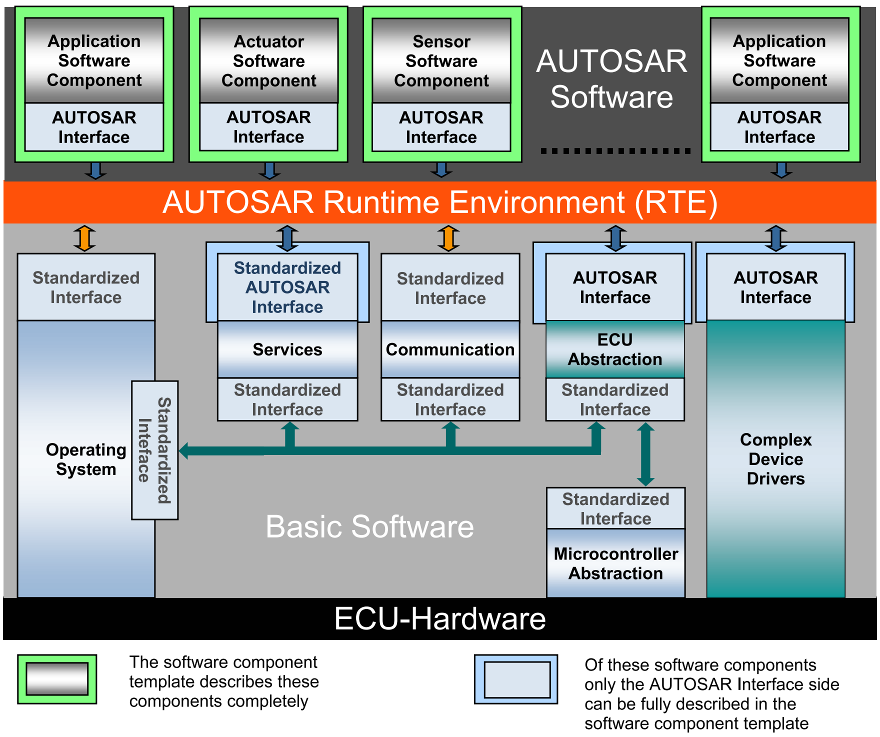


Figure 10.1: AUTOSAR ECU Software Architecture

The signal[1] flow from a hardware to software and vice versa will be described in the following sections.

A sensor[2] is converting a physical value (1) in Figure 10.2 (e.g. temperature, force, light intensity) into an electrical signal (2) which can be either a current or a voltage.

Inside the ECU generally there will be some electronics to enhance the electrical signal provided by the sensor. In AUTOSAR this is called ECU Electronics. This electronics device is also responsible for the conversion of the electrical signal into a micro-controller compatible form (3), usually a voltage.

After the electrical signal has been enhanced and converted it will be captured by the micro-controller. This can either be done by a simple digital input, an analogue to digital converter or maybe a pulse-width demodulation module. Now the electrical signal is available as a software data value (4).

This signal flow is sketched in the top part of Figure 10.2.

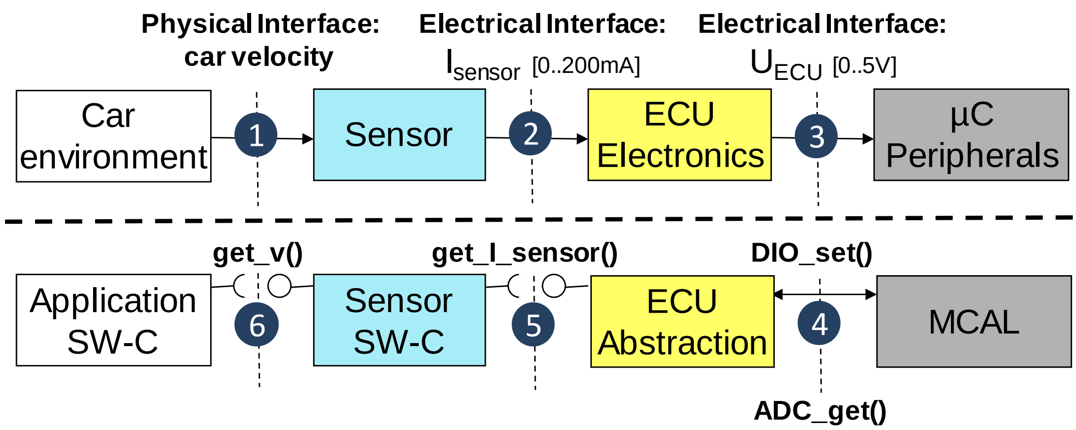


Figure 10.2: Interfaces between hardware and software

The interaction shown in Figure 10.2 shows an exemplary scenario. The usage of `ClientServerInterface` is just an example for the interaction pattern.

This signal chain is represented one-to-one in the AUTOSAR software architecture and depicted in the lower part of Figure 10.2.

In an implementation of AUTOSAR, only the micro-controller Abstraction (MCAL) has direct access to the peripheral hardware. This layer is going to be standardized and all hardware access should go through this layer. The idea of the AUTOSAR signal flow is to map the hardware to the corresponding software modules.

[1]The term “signal” is not going to be used here at its own but more specific terms will be used for the different abstractions of signals at the different stages of the signal flow.

[2]For the sake of simplicity this discussion is limited to the sensor aspects. Nevertheless, the same applies also for actuators.

So if an electrical current is the input to the micro-controller peripheral, the MCAL will deliver a data value that represents this current. As the ECU Electronics has enhanced and converted the electrical signal prior to the micro-controller, the corresponding software entity is reversing this conversion. This is performed in the ECU Abstraction layer.

So if the input to the ECU is an electrical current and the ECU Electronics has converted this current into a voltage (from 2 to 3), the ECU Abstraction will convert the data value voltage into an AUTOSAR signal representing a current (from 4 to 5). This AUTOSAR signal represents the actual current that was provided by the sensor (2).

Now the first step in the conversion has to be reversed: the sensor has converted a physical value into an electrical signal. And so the Sensor Software Component has to reverse this again. The Sensor Software Component will read the AUTOSAR signal representing the electrical value and transform it into an AUTOSAR signal representation of the physical value (from 5 to 6).

Now this physical value is available on the RTE and can be consumed or read by other SW-Components. Although the interface between the ECU Abstraction and the Sensor Software Component is also an AUTOSAR interface and could be routed through some communication-bus, it will not be practical to separate the ECU Abstraction and the corresponding SensorActuatorSwComponentType due to potentially high communication effort.

In Figure 10.3 a complete signal flow from a sensor input to an actuator output is shown.

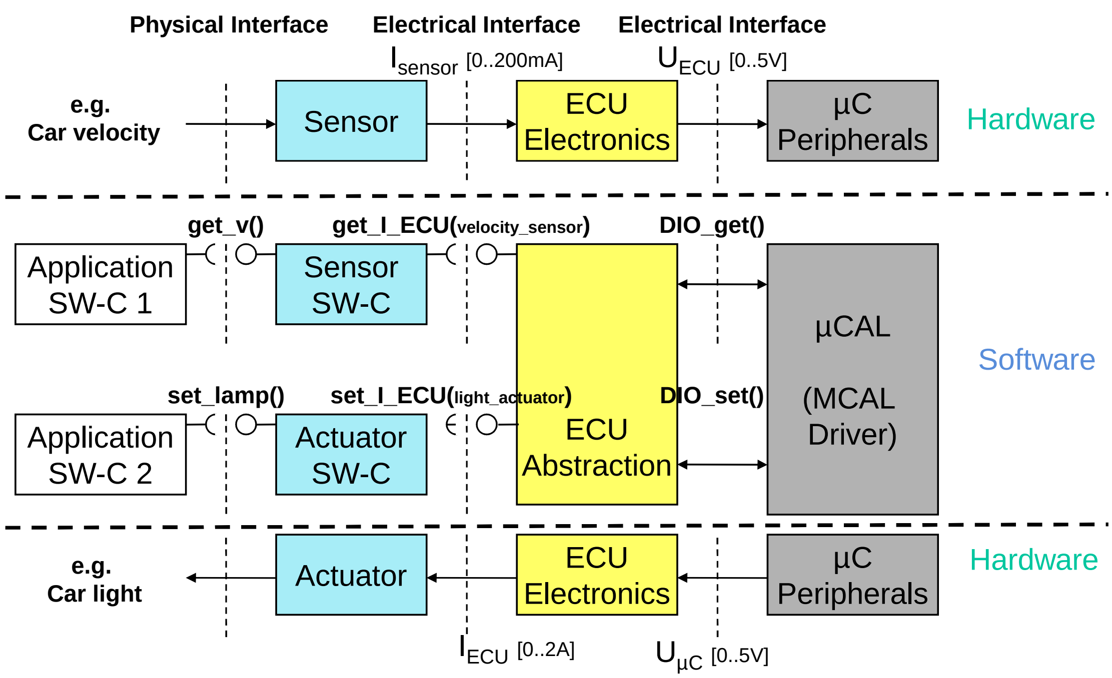


Figure 10.3: Sensor and Actuator Signal Flow

The interaction shown in Figure 10.3 shows an exemplary scenario. The usage of ClientServerInterface is just an example for the interaction pattern.

In the next section the interfaces between the involved software modules are discussed.

10.3 Interfaces and APIs

Two fundamentally different interfaces are involved when converting from sensors/actuators to software components, see markers “4” and “5” in Figure 10.2.

The interface between the micro-controller Abstraction and the ECU Abstraction is a Standardized Interface (see AUTOSAR Glossary [36]). This interface is not visible on the Virtual Function Bus and therefore the MCAL and ECU Abstraction have to be present on the same ECU.

For further description of this interface please refer to the `ECU Resource Template` documentation.

The interface to the `SensorActuatorSwComponentType`s is visible on the `Virtual Function Bus`. In general the `SensorActuatorSwComponentType` should be on the same ECU as the ECU hardware abstraction.

Also, the interface between the `SensorActuatorSwComponentType`s and the actual `AtomicSwComponentType`s representing the application is visible on the VFB. To describe the data that is going to be exchanged via this interface the standard AUTOSAR Interface description mechanisms are used (see chapter 3.4).

10.3.1 ECU Abstraction and its AUTOSAR Interfaces

Since the AUTOSAR standard is designed with the focus on the integration of software-components coming from different contractors, the interfaces between the different software-components obviously have to be compatible.

In the case of the sensors and actuators the interface is gathered in the ECU Abstraction. For each sensor and actuator there is one AUTOSAR `PortPrototype` that represents the AUTOSAR Signal that is delivered by the sensor or the AUTOSAR Signal that is consumed by the actuator. This relationship is depicted in Figure 10.4.

The interaction shown in Figure 10.4 shows an exemplary scenario. The usage of `ClientServerInterface` is just an example for the interaction pattern.

Each sensor and actuator has an AUTOSAR `PortPrototype` at the ECU Abstraction. Connected to this port is the `SensorActuatorSwComponentType`.

The `SensorActuatorSwComponentType` has one `PortPrototype` (i.e. IF_2) to the ECU Abstraction (which provides the values via IF_1) where it gets the AUTOSAR signals from the hardware, and one `PortPrototype` (i.e. IF_3) to `AtomicSwComponentType`s where it provides the actual physical value to the rest of AUTOSAR on the RTE.

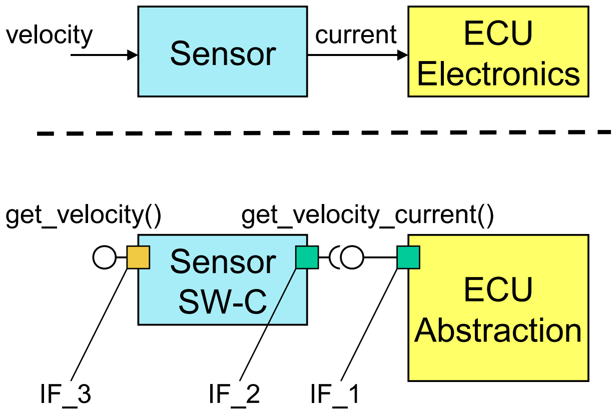


Figure 10.4: Interfaces of signals in software

In addition, the Interfaces between the ECU Abstraction and the SensorActuator-SwComponentType have to be compatible like defined in chapter 6.

10.4 Sensors/Actuators

In the layered software architecture described in [5] each hardware sensor/actuator is coupled to a SensorActuatorSwComponentType (see Figure 10.5).

[TPS_SWCT_01047] Reference from the software representation of a sensor/actuator to the actual hardware element ⌈Since the Software Component Template is going to be used to describe the SensorActuatorSwComponentType as well, there is also a reference needed from the software representation of a sensor/actuator to the actual hardware element described in the ECU Resource description.⌋ *(RS_SWCT_02080, RS_SWCT_03090)*

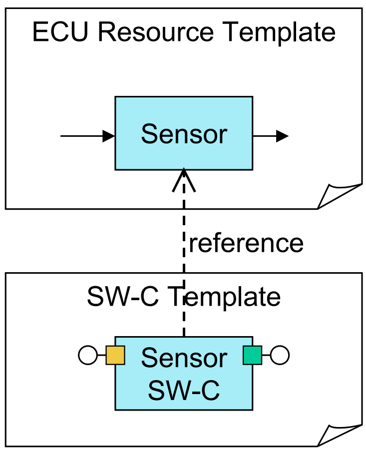


Figure 10.5: Shipment of a sensor

So each time a sensor/actuator is selected to be connected to an ECU also the corresponding SensorActuatorSwComponentType is available.

[constr_1144] SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType ⌈The attribute sensorActuator of SensorActuatorSwComponentType, the attribute hardwareElement of EcuAbstractionSwComponentType, and the attribute hardwareElement of ComplexDeviceDriverSwComponentType may **only** reference a HwType. References to other subclasses of HwDescriptionEntity are not allowed **at the time when the RTE is generated**.⌋*()*

Figure 10.6 depicts the reference of SensorActuatorSwComponentType designed as a specialization of an AtomicSwComponentType with an additional reference to a HwType.

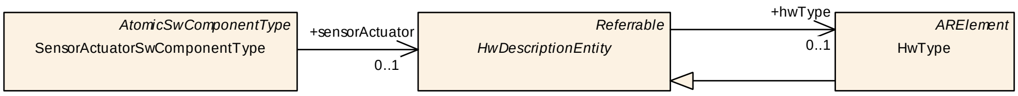


Figure 10.6: Sensor/actuator to Hardware Relationship

[constr_1109] Mapping of SwComponentPrototypes typed by a SensorActuatorSwComponentType ⌈A SwComponentPrototype typed by a SensorActuatorSwComponentType needs to be mapped and run on exactly that ECU that contains the HwElement corresponding to the HwType that its SensorActuatorSwComponentType refers to in case it accesses the hardware via the I/O hardware abstraction layer.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[TPS_SWCT_01048] SensorActuatorSwComponentType may use the I/O hardware abstraction directly ⌈In contrast to an ApplicationSwComponentType, a SensorActuatorSwComponentType may use the I/O hardware abstraction directly (via ports/connectors).⌋*(RS_SWCT_02080, RS_SWCT_03090)*

In case the sensor/actuator hardware is accessed via bus communication, e.g. is located on a LIN slave, no such mapping constraints apply (note that this is not handled via the IO hardware abstraction layer).

Class	**SensorActuatorSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The SensorActuatorSwComponentType introduces the possibility to link from the software representation of a sensor/actuator to its hardware description provided by the ECU Resource Template. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement, ARObject, AtomicSwComponentType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sensorActuator	HwDescriptionEntity	0..1	ref	Reference from the Sensor Actuator Software Component Type to the description of the actual hardware.

Table 10.1: SensorActuatorSwComponentType

10.5 I/O Hardware Abstraction

[TPS_SWCT_01389] I/O Hardware Abstraction interfaces MCAL drivers ⌈The I/O Hardware Abstraction interfaces on one side the MCAL drivers via Standardized Interfaces and on the other side the Sensor Actuator Software Component via AUTOSAR Interfaces. On the VFB[3] the I/O Hardware Abstraction is represented by the EcuAbstractionSwComponentType.⌋*()*

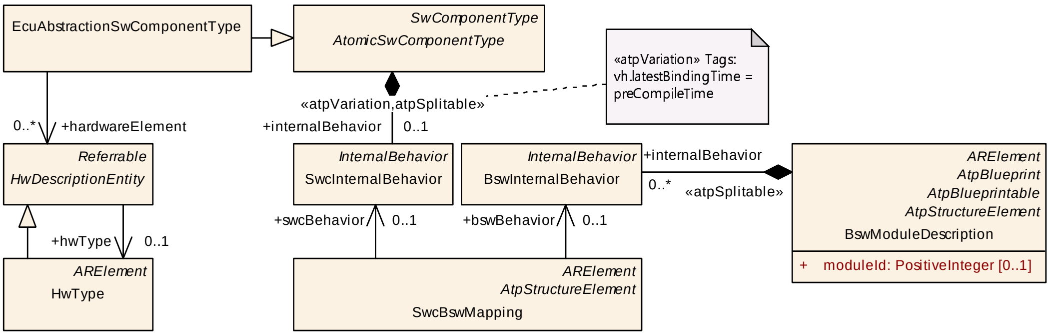


Figure 10.7: EcuAbstractionSwComponentType

[TPS_SWCT_01390] I/O Hardware Abstraction might have sub-structures ⌈Depending on the complexity of an ECU, the I/O Hardware Abstraction might have sub-structures. In this case the I/O Hardware Abstraction Layer is described by several EcuAbstractionSwComponentTypes on M1.⌋*()*

Class	**EcuAbstractionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ECUAbstraction is a special AtomicSwComponentType that resides between a software-component that wants to access ECU periphery and the Microcontroller Abstraction. The EcuAbstractionSw ComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement, ARObject, AtomicSwComponentType, AtpBlueprint, AtpBlueprintable, AtpClassifier, Atp Type, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, Sw ComponentType*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
hardware Element	HwDescriptionEntity	*	ref	Reference from the EcuAbstractionComponentType to the description of the used HwElements.

Table 10.2: EcuAbstractionSwComponentType

[TPS_SWCT_01391] I/O Hardware Abstraction abstracts from the location of peripheral I/O devices ⌈The I/O Hardware Abstraction abstracts from the location of peripheral I/O devices (on-chip or on- board) and the ECU hardware layout and has therefore dependencies to ECU Hardware described by HwElements. In addition, the EcuAbstractionSwComponentType is a hybrid concept sharing features of both software-components and basic software modules.⌋*()*

[TPS_SWCT_01392] Mapping between the EcuAbstractionSwComponentType and the corresponding BswModuleDescription ⌈The BSW part is described by the means of the Basic Software Module Template. The mapping between the EcuAbstractionSwComponentType and the corresponding BswModuleDescription is provided by the class SwcBswMapping which in addition also maps the two corresponding InternalBehaviors. This mechanism is further explained in [6].⌋()

10.6 Complex Driver

[TPS_SWCT_01393] Complex Driver ⌈A Complex Driver implements complex sensor evaluation and actuator control with direct access to the micro-controller using specific interrupts and/or complex micro-controller peripherals to fulfill the special functional and timing requirements.

In addition, it might be used to implement enhanced services / protocols or encapsulates legacy functionality of a non-AUTOSAR system.⌋()

See also document [3].

[TPS_SWCT_01394] Complex Driver is represented by the ComplexDeviceDriverSwComponentType ⌈On the VFB the Complex Driver is represented by the ComplexDeviceDriverSwComponentType. An ECU might have zero to many ComplexDeviceDriverSwComponentTypes.⌋()

Class	**ComplexDeviceDriverSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ComplexDeviceDriverSwComponentType is a special AtomicSwComponentType that has direct access to hardware on an ECU and which is therefore linked to a specific ECU or specific hardware. The ComplexDeviceDriverSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
hardwareElement	HwDescriptionEntity	*	ref	Reference from the ComplexDeviceDriverSwComponentType to the description of the used HwElements.

Table 10.3: ComplexDeviceDriverSwComponentType

[TPS_SWCT_01395] ComplexDeviceDriverSwComponentType has dependencies to ECU Hardware ⌈Similar to EcuAbstractionSwComponentType the ComplexDeviceDriverSwComponentType has dependencies to ECU Hardware described by HwElements and is a hybrid between Software Component and Basic Software Module.⌋()

[TPS_SWCT_01396] Mapping between the ComplexDeviceDriverSwComponentType and the corresponding BswModuleDescription ⌈The BSW part is described by the means of the Basic Software Module Template.

The mapping between the ComplexDeviceDriverSwComponentType and the corresponding BswModuleDescription is provided by the class SwcBswMapping which in addition also maps the two corresponding InternalBehaviors.

This mechanism is further explained in [6].⌋()

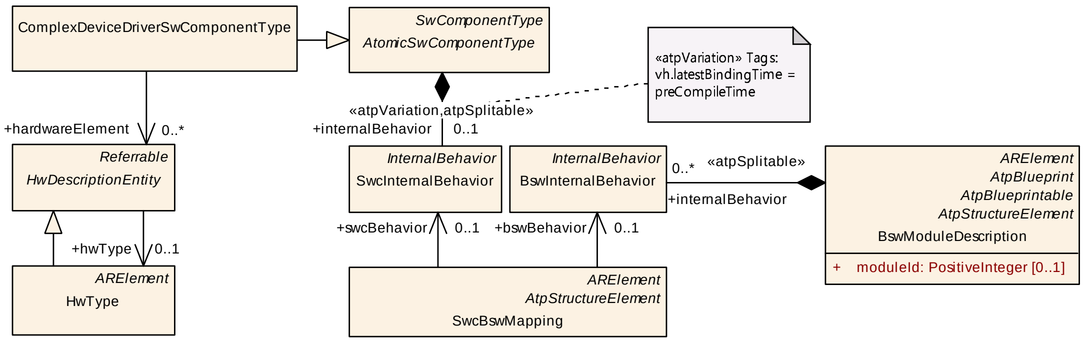


Figure 10.8: ComplexDeviceDriverSwComponentType

[constr_1979] Existence of the reference SwcBswMapping.bswBehavior ⌈For each SwcBswMapping, the reference to BswInternalBehavior in the role bswBehavior shall exist **at the time when the RTE is generated**.⌋()

[constr_1980] Existence of the reference SwcBswMapping.swcBehavior ⌈For each SwcBswMapping, the reference to BswInternalBehavior in the role swcBehavior shall exist **at the time when the RTE is generated**.⌋()

Class	**BswInternalBehavior**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the behavior of a BSW module or a BSW cluster w.r.t. the code entities visible by the BSW Scheduler. It is possible to have several different BswInternalBehaviors referring to the same BswModuleDescription.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.internalBehavior			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arTypedPerInstanceMemory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the Basic Software Module. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the Basic Software Module's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory.shortName, arTypedPerInstanceMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**BswInternalBehavior**			
bswPerInstanceMemoryPolicy	BswPerInstanceMemoryPolicy	*	aggr	Policy for a arTypedPerInstanceMemory The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswPerInstanceMemoryPolicy, bswPerInstanceMemoryPolicy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
clientPolicy	BswClientPolicy	*	aggr	Policy for a requiredClientServerEntry. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=clientPolicy, clientPolicy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
distinguishedPartition	BswDistinguishedPartition	*	aggr	Indicates an abstract partition context in which the enclosing BswModuleEntity can be executed. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=distinguishedPartition.shortName, distinguishedPartition.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
entity	BswModuleEntity	*	aggr	A code entity for which the behavior is described **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=entity.shortName, entity.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=5
event	BswEvent	*	aggr	An event required by this module behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
exclusiveAreaPolicy	BswExclusiveAreaPolicy	*	aggr	Policy for an ExclusiveArea in this BswInternalBehavior. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveAreaPolicy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedDataTypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a basic software module for its implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedDataTypeSet
includedModeDeclarationGroupSet	IncludedModeDeclarationGroupSet	*	aggr	This aggregation represents the included ModeDeclarationGroups **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedModeDeclarationGroupSet

▽

△

Class	**BswInternalBehavior**			
internal TriggeringPoint	BswInternalTriggering Point	*	aggr	An internal triggering point. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint.shortName, internal TriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=2
internal TriggeringPoint Policy	BswInternalTriggering PointPolicy	*	aggr	Policy for an internalTriggeringPoint in this BswInternal Behavior.. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPointPolicy, internal TriggeringPointPolicy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeReceiver Policy	BswModeReceiver Policy	*	aggr	Implementation policy for the reception of mode switches. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeReceiverPolicy, modeReceiver Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
modeSender Policy	BswModeSenderPolicy	*	aggr	Implementation policy for providing a mode group. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSenderPolicy, modeSender Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
parameterPolicy	BswParameterPolicy	*	aggr	Policy for a perInstanceParameter in this BswInternal Behavior. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterPolicy, parameterPolicy.variation Point.shortLabel vh.latestBindingTime=preCompileTime
perInstance Parameter	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) needed by this BswInternal Behavior. The role name perInstanceParameter is chosen in analogy to the similar role in the context of SwcInternal Behavior. In contrast to constantMemory, this object is not allocated locally by the module's code, but by the BSW Scheduler and it is accessed from the BSW module via the BSW Scheduler API. The main use case is the support of software emulation of calibration data. The aggregation is subject to variability with the purpose to support implementation variants. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceParameter.shortName, per InstanceParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45

▽

△

Class	**BswInternalBehavior**			
receptionPolicy	BswDataReception Policy	*	aggr	Data reception policy for inter-partition and/or inter-core communication. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=receptionPolicy, receptionPolicy.variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
releasedTrigger Policy	BswReleasedTrigger Policy	*	aggr	Policy for a releasedTrigger. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=releasedTriggerPolicy, releasedTrigger Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
schedulerName Prefix	BswSchedulerName Prefix	*	aggr	Optional definition of one or more prefixes to be used for the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=schedulerNamePrefix.shortName, scheduler NamePrefix.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
sendPolicy	BswDataSendPolicy	*	aggr	Policy for a providedData. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sendPolicy, sendPolicy.variationPoint.short Label vh.latestBindingTime=preCompileTime
service Dependency	BswService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation is subject to variability with the purpose to support the conditional existence of ServiceNeeds. The aggregation is splitable in order to support that ServiceNeeds might be provided in later development steps. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency.ident.shortName, serviceDependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
triggerDirect Implementation	BswTriggerDirect Implementation	*	aggr	Specifies a trigger to be directly implemented via OS calls. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=triggerDirectImplementation, triggerDirect Implementation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=15
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=variationPointProxy.shortName

Table 10.4: BswInternalBehavior

11 Services

11.1 Overview: Generation of Service-related Model Elements

This chapter covers the description and handling of AUTOSAR Service configuration.

[TPS_SWCT_01397] Hybrid concept between Basic Software Modules and a SwComponentType ⌈AUTOSAR Services can be seen as a hybrid concept between Basic Software Modules and a SwComponentType. AUTOSAR Services actually provide access to low-level and ECU-wide "standard functionalities" commonly referred to as "service".

AtomicSwComponentTypes that require AUTOSAR Services use Standardized AUTOSAR Interfaces to communicate with these. The connection of PortPrototypes of the ServiceSwComponentTypes and PortPrototypes of the AtomicSwComponentTypes implement several communication patterns.⌋*()*

I	II	III	IV
A	1..n	PPort:RPort	Distribution of data or modes to *n* software-components, e.g. used for ECU mode.
A*	1..n	RPort:PPort	Currently not used, not supported for client-server communication.
A**	1..n	PRPort:RPort	Distribution of data or modes to n software-components, e.g. used for ECU mode.
B	1..1	PPort:RPort	Software-component acts as Server, used for so called "call-backs".
B	1..1	RPort:PPort	Service acts as Server, typical Service usage.
C*	n..1	PPort:RPort	Conceptually not used to support index abstraction via PortDefinedArgumentValues.
C	n..1	RPort:PPort	Software-component acts as Server, used for so called "call-backs" invoked by more than one Service.
D	1..1	PR-Port:PRPort	I/O control data.

Table 11.1: ServiceConnectorPattern

Legend for Table 11.1:

I Pattern name

II Communication pattern (client/server, sender/receiver)

III Kind of PortPrototype at service : software-component

IV Description, use case

[TPS_SWCT_01398] Communication patterns for AUTOSAR services ⌈The communication patterns for AUTOSAR services are summarized in Table 11.1.⌋*()*

[TPS_SWCT_01403] Impact of AUTOSAR services on the methodology ⌈Due to this special nature, such AUTOSAR Services need to be handled with particular attention in the methodology [4]. That is, a number of elements need to be generated during ECU integration.⌋*()*

The following list of paragraphs presents a short overview over the steps required for the configuration of AUTOSAR Services.

Note that most of these steps are performed by tools and the model elements being created in these steps are rather specific to Service configuration and are not to be modeled manually within AUTOSAR authoring tools.

In particular, the following requirements apply:

- **[TPS_SWCT_01399] Dependency is modeled by aggregating required and provided PortPrototypes** ⌈The dependency of an AtomicSwComponentType (or more precisely, one of its non-abstract derived meta-classes) from an AUTOSAR Service is modeled by aggregating required and provided PortPrototypes.⌋*()*

  [TPS_SWCT_01400] PortInterface selected from the set of standardized Service Interfaces ⌈The PortInterface being implemented by the PortPrototypes needs to be one of a number of standardized Service Interfaces which is indicated by having its isService attribute set to true and is (via several levels of indirection) finally referenced by ServiceNeeds.⌋*()*

  Additionally, the software components and Basic Software Modules shall specify ServiceNeeds containing further input information for the later Service configuration step.

- **[TPS_SWCT_01401] Form a top-level RootSwCompositionPrototype** ⌈When defining a software system, the AtomicSwComponentType is used in the form of SwComponentPrototypes within a CompositionSwComponentType. In this step, the non-service ports of all required interfaces are being connected using AssemblySwConnectors and DelegationSwConnectors in order to eventually form a top-level RootSwCompositionPrototype which can be referenced in an AUTOSAR System.⌋*()*

- **[TPS_SWCT_01402] Mapping of all AtomicSwComponentType instances to EcuInstances** ⌈In System Configuration Phase, the mapping of all AtomicSwComponentType instances to EcuInstances is done (for the specification of EcuInstance see [10]). The ServiceNeeds may be used by tools to check for available resources on the targeted ECUs.⌋*()*

- **[TPS_SWCT_01404] Creation of the Ecu Extract** ⌈The ECU Extract is extracted from the System Configuration for each ECU. As explained in the AUTOSAR System Template [10], this contains an ECU-centric view onto the system description.

  This includes a reduced version of the system's RootSwCompositionPrototype where SwComponentPrototypes not being mapped to the ECU are being left out and all Compositions are stripped off, so that in the ECU Extract only one instance of CompositionSwComponentType remains which aggregates all SwComponentPrototypes on the ECU in a flat manner.⌋*()*

- **[TPS_SWCT_01405] Creation of the ServiceSwComponentTypes** ⌈In ECU Configuration, for each Service required on the ECU exactly one ServiceSwComponentType is created based on the needs from the AtomicSwComponentTypes: An adequate number of PortPrototypes are created on this ServiceSwComponentType for each needed port at the AtomicSwComponentType.

  Thereby the specified communication pattern A, B, C or D for a specific kind of ServicePort has to be considered.⌋*()*

  See also chapter 11.2 and table 11.1.

- **[TPS_SWCT_01406] Creation of SwComponentPrototype typed by a ServiceSwComponentType** ⌈Per Service exactly one SwComponentPrototype typed by a ServiceSwComponentType is created based on the ServiceSwComponentType. Additionally, the connectors are constructed that connect the pairs of PortPrototypes belonging to the SwComponentPrototypes requiring services and those belonging to the actual services.⌋*()*

- **[TPS_SWCT_01407] Creation of InternalBehavior typed by a ServiceSwComponentType** ⌈For each ServiceSwComponentType an SwcInternalBehavior is created or extended providing the information about PortDefinedArgumentValues, RunnableEntitys and RTEEvents necessary for RTE generation.⌋*()*

  Further, detailing of the service ports by filling in these PortDefinedArgumentValues is also done in ECU Configuration phase. See also chapter 7.6.3.

- **[TPS_SWCT_01408] Creation of SwcBswMapping** ⌈For the RTE module configuration an implementation of the AUTOSAR Service described by a Basic Software Module Description needs to be selected. The SwcBswMapping to the corresponding SwComponentPrototype needs to be created accordingly.

  For each SwcInternalBehavior one SwcImplementation is being created. The information for SwcImplementation should be generated based on the available information of BswImplementation[1].⌋*()*

- **[TPS_SWCT_01409] Update of PortDefinedArgumentValues** ⌈Depending on the configuration of the Service BSW it might be necessary to update the ValueSpecifications belonging to the PortDefinedArgumentValues generated in a previous step.⌋*()*

[1]This step does in general not require copying any attributes or elements aggregated in BswImplementation into the generated instance of SwcImplementation since the only mandatory information for the RTE configuration is the reference from SwcImplementation to the selected SwcInternalBehavior.

Class	SwcBswMapping			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Maps an SwcInternalBehavior to an BswInternalBehavior. This is required to coordinate the API generation and the scheduling for AUTOSAR Service Components, ECU Abstraction Components and Complex Driver Components by the RTE and the BSW scheduling mechanisms. **Tags:**atp.recommendedPackage=SwcBswMappings			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bswBehavior	BswInternalBehavior	0..1	ref	The mapped BswInternalBehavior
runnable Mapping	SwcBswRunnable Mapping	*	aggr	A mapping between a pair of SWC and BSW runnables. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnableMapping, runnable Mapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
swcBehavior	SwcInternalBehavior	0..1	ref	The mapped SwcInternalBehavior.
synchronized ModeGroup	SwcBswSynchronized ModeGroupPrototype	*	aggr	A pair of SWC and BSW mode group prototypes to be synchronized by the scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=synchronizedModeGroup, synchronized ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
synchronized Trigger	SwcBswSynchronized Trigger	*	aggr	A pair of SWC and BSW Triggers to be synchronized by the scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=synchronizedTrigger, synchronized Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 11.2: SwcBswMapping

11.2 Service Software Component Type

As mentioned in [TPS_SWCT_01405], AUTOSAR Services are represented by a meta-model class of their own, the ServiceSwComponentType. As can be seen in Figure 11.1, ServiceSwComponentType is a specialization of AtomicSwComponentType.

Like any other SwComponentType they can aggregate PortPrototypes.

[constr_2019] ServiceSwComponentType shall have service ports only ⌈In the case of ServiceSwComponentType, all aggregated PortPrototypes need to have an ≪isOfType≫ relationship to a PortInterface which has its isService attribute set to true **at the time when the RTE is generated**.

The exceptions described in

- [TPS_SWCT_01572],

- [TPS_SWCT_01579],
- [TPS_SWCT_01831] and
- [TPS_SWCT_01580]

apply.⌋()

[TPS_SWCT_01579] Dcm can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in AbstractProvidedPortPrototype ⌈An exception to the rule described in [constr_2019] applies: the Dcm can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in AbstractProvidedPortPrototypes (PPortPrototype in case of ParameterInterface.parameter) as long as [constr_1071] is fulfilled.

For this purpose, the ServiceSwComponentType that represents the Dcm functionality can have AbstractRequiredPortPrototypes typed by a compatible SenderReceiverInterface that may set attribute isService to false.⌋()

Please note that the inclusion of ParameterInterface.parameters in [TPS_SWCT_01579] is done under the assumption that the ServiceSwComponentType that represents the Dcm would connect to a PPortPrototype owned by a ParameterSwComponentType.

[TPS_SWCT_01831] Dcm can directly access SenderReceiverInterface.dataElements or NvDataInterface.nvDatas in AbstractRequiredPortPrototypes ⌈An exception to the rule described in [constr_2019] applies: the Dcm can directly access SenderReceiverInterface.dataElements or NvDataInterface.nvDatas in AbstractRequiredPortPrototypes as long as [constr_1071] is fulfilled.

For this purpose, the ServiceSwComponentType that represents the Dcm functionality can have AbstractProvidedPortPrototypes typed by a compatible SenderReceiverInterface that may set attribute isService to false.⌋()

Please note that the exclusion of ParameterInterface.parameters from [TPS_SWCT_01831] is a direct consequence of [constr_1137], i.e. the ServiceSwComponentType that represents the Dcm is not allowed to expose a PPortPrototype typed by a ParameterInterface.

[TPS_SWCT_01580] Dem can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in PPortPrototypes ⌈An exception to the rule described in [constr_2019] applies: the Dem can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in AbstractProvidedPortPrototypes (PPortPrototype in case of ParameterInterface.parameter) as long as [constr_1071] is fulfilled.

For this purpose, the ServiceSwComponentType that represents the Dem functionality can have RPortPrototypes typed by a compatible SenderReceiverInterface that may set attribute isService to false.⌋()

Please note that the inclusion of ParameterInterface.parameters in [TPS_SWCT_01580] is done under the assumption that the ServiceSwComponentType that represents the Dem would connect to a PPortPrototype owned by a ParameterSwComponentType.

[TPS_SWCT_01411] Use cases for a ServiceSwComponentType to express ServiceNeeds ⌈There are valid use cases for a ServiceSwComponentType to express ServiceNeeds[2]. This leads to a situation where ServiceSwComponentTypes are iteratively created in response to ServiceNeeds expressed by other ServiceSwComponentTypes. Please refer to the AUTOSAR methodology [4] for more details about how this shall be implemented into the workflow.⌋()

Similar to an EcuAbstractionSwComponentType and a ComplexDeviceDriverSwComponentType, the ServiceSwComponentType represents a hybrid concept between Software Component and Basic Software Module. The BSW part is described by the means of the BSW Module Description Template [6].

The mapping between the ServiceSwComponentType and the corresponding BswModuleDescription is provided by the class SwcBswMapping which in addition also maps the two corresponding InternalBehaviors (see [TPS_SWCT_01408]). This mechanism is further explained in [6].

[TPS_SWCT_01412] ServiceSwComponentType shall be added in ECU Configuration phase ⌈ServiceSwComponentType shall not be used when modeling application software using CompositionSwComponentType; they are only added in ECU Configuration phase where exactly one SwComponentPrototype per ServiceSwComponentType per ECU is added to the ECU Description model.

The Base ECU Config Generator tool needs to take care that for all service ports of SwComponentPrototypes mapped to the ECU service ports at the appropriate ServiceSwComponentTypes are created.

In case of pattern A for each different type of service port one port on the ServiceSwComponentType is created.

In case of pattern B and C for each service port of a SwComponentPrototype one port on the ServiceSwComponentType is created.

More explicitly, all instances of AtomicSwComponentType need to be checked for PortPrototypes of PortInterfaces with isService attribute set to true and referenced by ServiceNeeds and for each of these PortInterface instances belonging to the AUTOSAR Service to be configured one PortPrototype implementing the same or a compatible PortInterface needs to be created on the ServiceSwComponentType.⌋()

[2] Thereby the previously existing constraint 1127 becomes invalid.

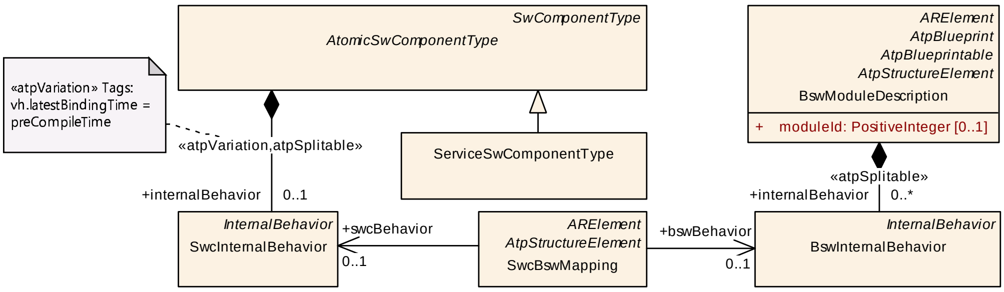


Figure 11.1: ServiceSwComponentType

Class	**ServiceSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	ServiceSwComponentType is used for configuring services for a given ECU. Instances of this class are only to be created in ECU Configuration phase for the specific purpose of the service configuration. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 11.3: ServiceSwComponentType

In the process of creating PortPrototypes the specified communication pattern A, B, or C for a specific kind of service port has to be considered, see table 11.1.

[TPS_SWCT_02500] Roles on Application/Service Components need to Match ⌈The roles of the PortPrototypes (required/provided) on the Application Component and the Service Component side obviously need to match. For example an RPortPrototype attached to an application AtomicSwComponentType matches a PPortPrototype attached to a ServiceSwComponentType.⌋*()*

11.3 Service Proxy Component Type

[TPS_SWCT_01413] Local communication with services ⌈Application software components may communicate with an instance of a ServiceSwComponentType only locally on an ECU.⌋*()*

[TPS_SWCT_01414] Mode manager needs to communicate with application software components located on other ECUs ⌈There are however use cases for the application and vehicle mode management, where a mode manager (namely the Basic Software Mode Manager, see [21]) is part of the basic software but conceptually still needs to communicate with application software components located on other ECUs.

In order to make this communication possible, the ServiceProxySwComponentType is used.

For the application software and the RTE it behaves like a "normal" AtomicSwComponentType, but it is actually a proxy for an AUTOSAR Service.⌋()

The concept of mode requests across ECU boundaries is exemplified in Figure 11.2.

[TPS_SWCT_01415] Interfaces of ServiceProxySwComponentType ⌈This means that on the one side it has to communicate over service ports with the ECU-local ServiceSwComponentType it represents. On the other side it has to offer the corresponding PortPrototypes to the ApplicationSwComponentTypes.⌋()

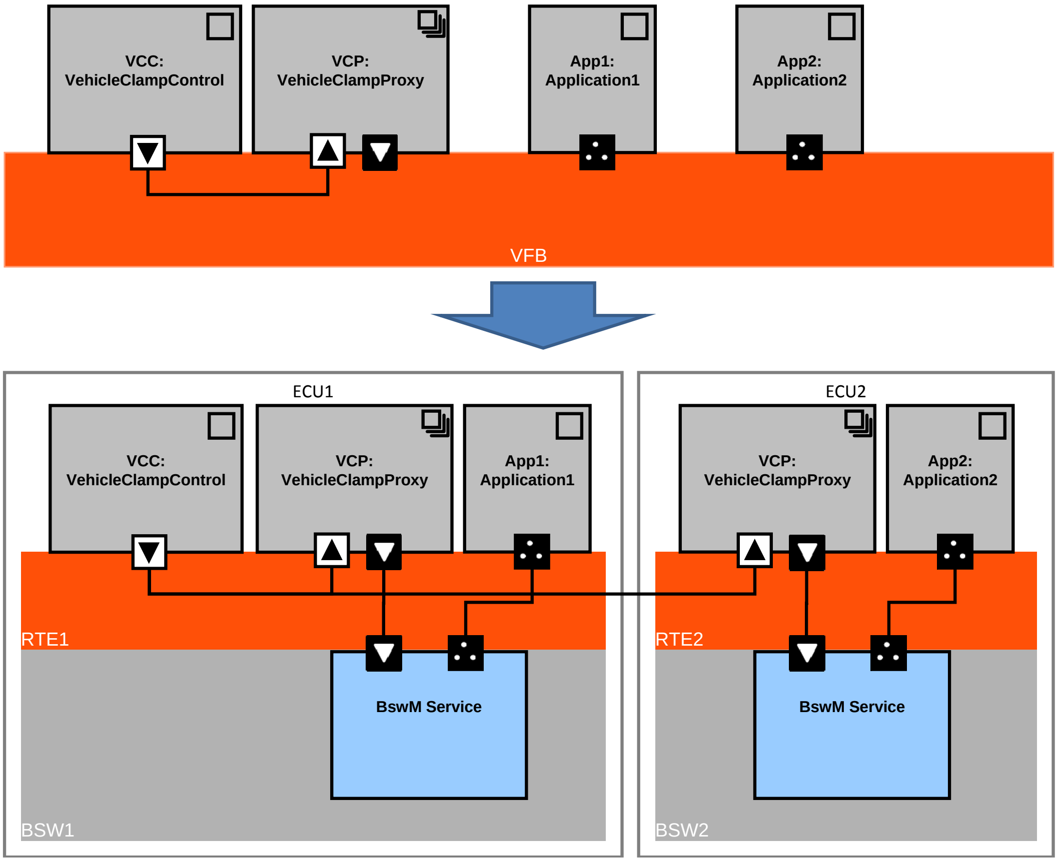


Figure 11.2: Mode request over the network [3]

In the meta-model, the ServiceProxySwComponentType does not differ from an ApplicationSwComponentType except by its class. It is up to the implementer to meet the restrictions imposed by the semantics as a proxy.

[TPS_SWCT_01416] Difference between a ServiceProxySwComponentType and an ApplicationSwComponentType ⌈The main difference between a ServiceProxySwComponentType and an ApplicationSwComponentType is on system level:

A prototype of a ServiceProxySwComponentType can be mapped to several ECUs even if it appears only once in the VFB system, because such a prototype is required on each ECU, where it has to address a local ServiceSwComponentType.

As a result of this, a ServiceProxySwComponentType can only receive but not send signals over the network. More details are explained in the class table below.⌋()

[constr_2016] Connections between SwComponentPrototypes of type ServiceProxySwComponentType ⌈A connection between PortPrototypes belonging to SwComponentPrototypes where both are typed by ServiceProxySwComponentType is not permitted **at the time when the RTE is generated**.⌋()

Class	**ServiceProxySwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This class provides the ability to express a software-component which provides access to an internal service for remote ECUs. It acts as a proxy for the service providing access to the service. An important use case is the request of vehicle mode switches: Such requests can be communicated via sender-receiver interfaces across ECU boundaries, but the mode manager being responsible to perform the mode switches is an AUTOSAR Service which is located in the Basic Software and is not visible in the VFB view. To handle this situation, a ServiceProxySwComponentType will act as proxy for the mode manager. It will have R-Ports to be connected with the mode requestors on VFB level and Service-Ports to be connected with the local mode manager at ECU integration time. Apart from the semantics, a ServiceProxySwComponentType has these specific properties: • A prototype of it can be mapped to more than one ECUs in the system description. • Exactly one additional instance of it will be created in the ECU-Extract per ECU to which the prototype has been mapped. • For remote communication, it can have only R-Ports with sender-receiver interfaces and 1:n semantics. • There shall be no connectors between two prototypes of any ServiceProxySwComponentType. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 11.4: ServiceProxySwComponentType

[constr_2017] Ports of ServiceProxySwComponentTypes ⌈ServiceProxySwComponentType is only permitted to define

- RPortPrototypes that are typed by SenderReceiverInterface or
- PortPrototypes that are typed by a PortInterface where the isService attribute is set to true.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_2018] Supported remote communication of a ServiceProxySwComponentType ⌈For remote communication, ServiceProxySwComponentType can have only RPortPrototypes typed by SenderReceiverInterfaces in a 1:n communication scenario.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

11.4 Non Volatile Memory

11.4.1 Introduction

The AUTOSAR Architecture defines two alternatives how a software component can access non-volatile memory.

- The first option is that the software component defines in its InternalBehavior a PerInstanceMemory and a NvBlockNeeds referring to the PerInstanceMemory via a RoleBasedDataAssignment.

  In this case the NVRAM Block is exclusively accessed by this software component and the NvM [37]. Therefore, the *nv data* is encapsulated inside the software component and can not be accessed directly by other software components.

  The PerInstanceMemory can be typed with AutosarDataTypes in the case of arTypedPerInstanceMemory or with C data types in the case of perInstanceMemory. For further information see section 7.7 and 13.

- The second option is that the software component uses communication based on PortPrototypes to access *nv data* provided by a NvBlockSwComponentType.

  In this case it is possible that *nv data* used by different AtomicSwComponentTypes is packed in one larger NVRAM Block to reduce the NVRAM Block management overhead or that the same *nv data* used by several software components with a reduced RAM overhead. The *nv data* of a NvBlockSwComponentType is typed with AutosarDataTypes.

More details regarding particular scenarios of interacting with the NvM [37] can be found in section 13.2.

11.4.2 NvBlockComponent

[TPS_SWCT_01142] non-volatile data are provided by a specialized AtomicSwComponentType ⌈On the VFB [3], the non-volatile data are provided by a specialized AtomicSwComponentType, the NvBlockSwComponentType.

An NvBlockSwComponentType can represent one or more NVRAM Blocks managed by the *NVRAM Manager*. The *nv data* PortPrototypes of the NvBlockSwComponentType are exclusively typed by NvDataInterfaces.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01143] Non-volatile data represented by an NvBlockSwComponentType can be read and written ⌈The non-volatile data represented by an NvBlockSwComponentType can be read and written. For this purpose the NvBlockSwComponentType is allowed to have PPortPrototypes and RPortPrototypes.⌋*(RS_SWCT_03225)*

Additionally, the NvBlockSwComponentType might have client server PortPrototypes to offer the block-related services, administrative services or notifications.

[constr_2009] Supported kinds of PortPrototypes of a NvBlockSwComponentType ⌈With respect to external communication, NvBlockSwComponentType is limited to the definition of the following kinds of PortPrototype:

- PortPrototypes typed by either NvDataInterfaces or ClientServerInterfaces
- RPortPrototypes typed by ModeSwitchInterfaces

This rule shall be imposed **at the time when the RTE is generated**.⌋()

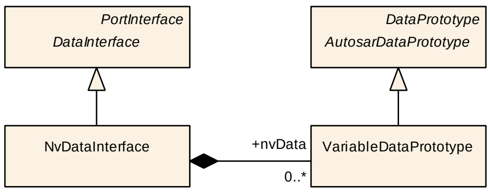


Figure 11.3: NvDataInterface

[constr_2010] Connections between SwComponentPrototypes of type NvBlockSwComponentType ⌈The existence of SwConnectors that refer to PortPrototypes belonging to SwComponentPrototypes where both are typed by NvBlockSwComponentType is not permitted **at the time when the RTE is generated**.⌋()

Class	**NvBlockSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The NvBlockSwComponentType defines non volatile data which data can be shared between SwComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**NvBlockSwComponentType**			
bulkNvData Descriptor	BulkNvDataDescriptor	*	aggr	This aggregation formally defines the bulk Nv Blocks that are provided to the application software by the enclosing NvBlockSwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bulkNvDataDescriptor.shortName, bulkNvDataDescriptor.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
nvBlock Descriptor	NvBlockDescriptor	*	aggr	Specification of the properties of exactly one NVRAM Block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=nvBlockDescriptor.shortName, nvBlockDescriptor.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 11.5: NvBlockSwComponentType

Class	**NvDataInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A non volatile data interface declares a number of VariableDataPrototypes to be exchanged between non volatile block components and atomic software components. **Tags:**atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
nvData	VariableDataPrototype	*	aggr	The VariableDataPrototype of this nv data interface.

Table 11.6: NvDataInterface

11.4.3 Software-Components using *NVRAM data* of NvBlockComponents

[TPS_SWCT_01141] `AtomicSwComponentType` may have `AbstractRequiredPortPrototype`s typed by an `NvDataInterface` ⌈An `AtomicSwComponentType` may have `AbstractRequiredPortPrototype`s typed by an `NvDataInterface`.

If such an `AbstractRequiredPortPrototype` remains unconnected the `nvData` still need to have reasonable value[3].⌋*(RS_SWCT_03225)*

[constr_2011] Connections between `SwComponentPrototype`s typed by `NvBlockSwComponentType` and `SwComponentPrototype`s typed by other `AtomicSwComponentType`s ⌈A `PortPrototype` typed by an `NvDataInterface` owned by a `SwComponentPrototype` typed by an `NvBlockSwComponentType` shall be connected to a `PortPrototype` typed by **either** an `NvDataInterface` **or**

[3]Note that it is assumed that only a subset of meta-classes that inherit from `AtomicSwComponentType` will actually apply for the definition of initial values for `nvData`. Most likely the `ApplicationSwComponentType` and the `SensorActuatorSwComponentType` will be candidates for using this feature but it will obviously not be reasonable for e.g. `NvBlockSwComponentType`.

a SenderReceiverInterface owned by a SwComponentPrototype that is typed by an other subclass of AtomicSwComponentType.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

There is no valid use case to connect a PortPrototype typed by a ParameterInterface to a PortPrototype typed by an NvDataInterface.

[constr_1148] PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes ⌈PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes as well as the PortInterfaces used in the context of NvBlockSwComponentTypes shall **always** set the value of the attribute isService to false **at the time when the RTE is generated**.⌋()

[constr_1149] PortPrototypes used for NV data management ⌈A PortPrototype typed by a ClientServerInterface used for NV data management, i.e. the interaction of ApplicationSwComponentTypes with NvBlockSwComponentTypes, shall be typed by ClientServerInterfaces that are compatible to the particular ClientServerInterfaces derived from MOD_GeneralBlueprints [30]. [constr_1148] applies.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

For details see chapter 6.4.4.

Note: In case of *nv data* which is read and written and shared between several SwComponentPrototypes the NvBlockSwComponentType establishes a not directly obvious kind of communication.

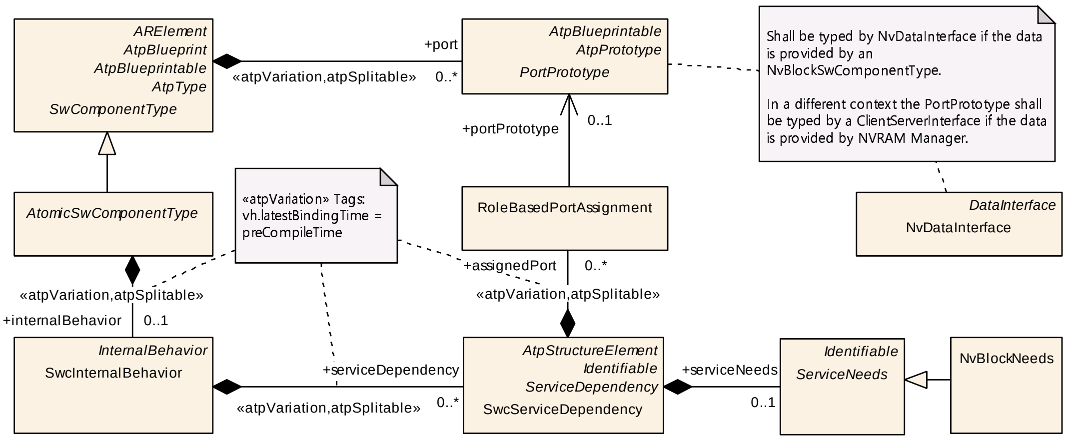


Figure 11.4: NvBlockNeeds for *nv data* PortPrototypes

Nevertheless, this is intentionally supported and it is under responsibility of the VFB designer to take care that only *nv data* is shared where the functionality of the software components is not impaired.

To determine for an VFB designer which *nv data* can be potentially by mapped into the same NVRAM Block a software-component can specify further attributes for its *nv data* PortPrototypes by the definition of SwcServiceDependency(s) with NvBlockNeeds.

In this case the role attribute of the assignedPort has to be set to the value NvDataPort. This aspect is also explained in section 13.2.4.

In contrast to the NvBlockNeeds that describe the expected configuration of a whole NVRAM Block, the NvBlockNeeds for *nv data* PortPrototypes defines only the attributes which are required from the point of view of a software-component to ensure its functionality.

This means an empty attribute has the semantic of "don't care".

Further on the VFB designer has got the freedom to specify how the requested NVRAM Block attributes are fulfilled by the created NvBlockDescriptor.

For instance, *nv data* with different writingFrequency might be mapped to one NVRAM Block. In this case the NvBlockNeeds of the NvBlockDescriptor has to indicate the worst case which is the higher frequency.

[TPS_SWCT_01675] Recommendations for attributes of NvBlockNeeds or for NvBlockDescriptor ⌈The formal modeling of a NvBlockDescriptor should follow the recommendations given in table 11.7.⌋*(RS_SWCT_03225)*

But please note that table 11.7 does not represent a binding constraint.

Attribute of NvBlockNeeds	NvBlockNeeds of different *nv data* PortPrototypes of software-components	NvBlockNeeds of NvBlockDescriptor
readonly	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.
reliability	Can be different.	Recommended to be set to the highest reliability class request by any mapped *nv data* PortPrototypes.
resistantToChangedSw	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.
restoreAtStart	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.
storeAtShutdown	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.
writeOnlyOnce	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.
writingFrequency	Can be different.	Recommended to be set to the highest requested frequency of the mapped *nv data* PortPrototypes.
writingPriority	Can be different.	Recommended to be set to the highest requested priority of the mapped *nv data* PortPrototypes.
writeVerification	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests a write verification.
calcRamBlockCrc	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests a CRC calculation.
checkStaticBlockId	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests a check of the static block ID.

▽

△

Attribute of NvBlockNeeds	NvBlockNeeds of different *nv data* PortPrototypes of software-components	NvBlockNeeds of NvBlockDescriptor
ramBlockStatusControl	Can be different.	Recommended to set to RamBlockStatusControlEnum.api if any of the *nv data* PortPrototypes requests a use of the API for accessing the block.
storeCyclic	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests cyclic writing.
storeEmergency	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests emergency writing.
storeImmediate	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests immediate writing.
storeOnChange	Can be different.	Recommended to set to true if any of the *nv data* PortPrototypes requests on-change writing.
selectBlockForFirstInitAll	Recommended to match for all connected *nv data* PortPrototypes if specified.	Recommended to be identical as requested by *nv data* PortPrototypes.

Table 11.7: NvBlockNeeds dependencies

With respect to the completeness of table 11.7 (which intentionally doesn't contain a remark regarding the value of cyclicWritingPeriod), it should be noted that (according to [TPS_SWCT_01585]) the value of NvBlockDescriptor.nvBlockNeeds.cyclicWritingPeriod shall be ignored in favor of NvBlockDescriptor.timingEvent.period.

Therefore, the missing statement for cyclicWritingPeriod in the spirit of table 11.7 is that the values of SwcServiceDependency.serviceNeeds.cyclicWritingPeriod can be different from the value of NvBlockDescriptor.timingEvent.period.

It is recommended that the value of NvBlockDescriptor.timingEvent.period shall be set to the lowest requested time value of the mapped *nv data* PortPrototypes (implemented by SwcServiceDependency.serviceNeeds.cyclicWritingPeriod).

11.4.4 Software-Components connected to NvBlockComponents

Please note that restrictions apply on the creation of AssemblySwConnectors between NvBlockSwComponentType and other AtomicSwComponentTypes.

In particular ApplicationSwComponentTypes communicating with each other used buffers generated and controlled by the RTE to exchange data. An NvBlockSwComponentType, however, maintains its own buffer in form of the ramBlock.

Thus, an ApplicationSwComponentType that reads a dataElement that may be provided by either another ApplicationSwComponentType or an NvBlockSwComponentType could not actually access the dataElement because it cannot decide whether it needs to access the buffer provided by the RTE or the ramBlock.

Therefore, scenarios like this are considered invalid by regulation of the AUTOSAR standard.

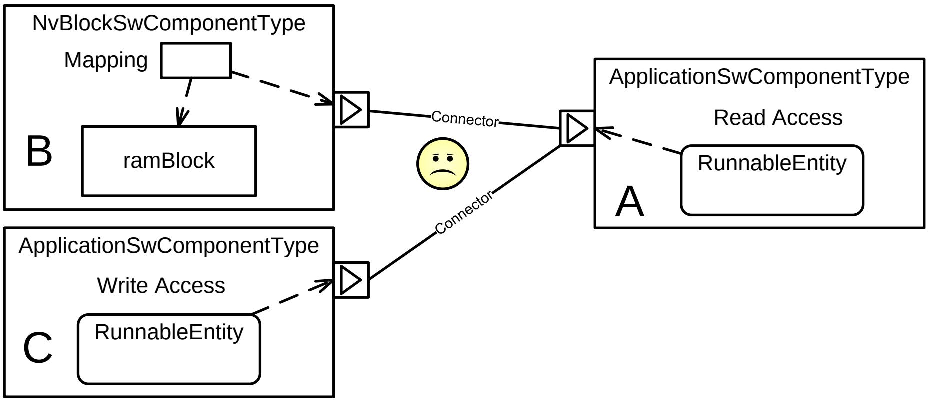


Figure 11.5: Example invalid connection between software-components (a)

[constr_1417] Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (I) ⌈A configuration where an RPortPrototype owned by an AtomicSwComponentType is simultaneously and directly connected to AbstractProvidedPortPrototypes of a collection of AtomicSwComponentTypes where at least one in the collection is an NvBlockSwComponentType for a matching set of dataElements in all these PortPrototypes shall be considered invalid **at the time when the RTE is generated**.⌋*()*

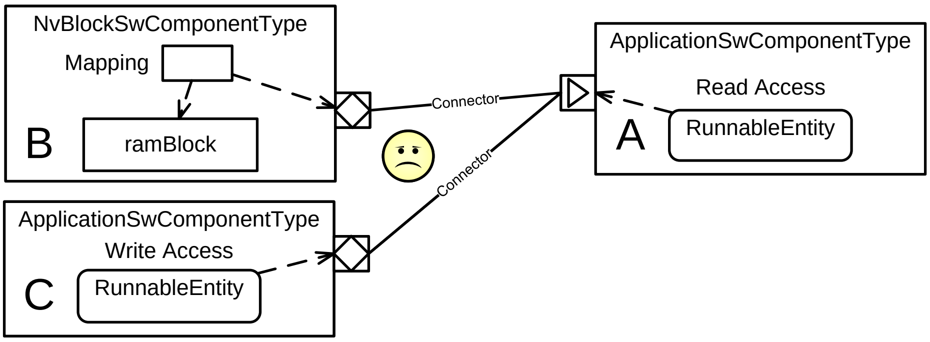


Figure 11.6: Example invalid connection between software-components (b)

The scenario covered by [constr_1417] is depicted in Figures 11.5 and 11.6.

[constr_1418] Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (II) ⌈A configuration where a PRPortPrototype owned by an AtomicSwComponentType is connected to a PPortPrototype owned by an

NvBlockSwComponentType for a matching set of dataElements in all these PortPrototypes shall be considered invalid **at the time when the RTE is generated**.⌋()

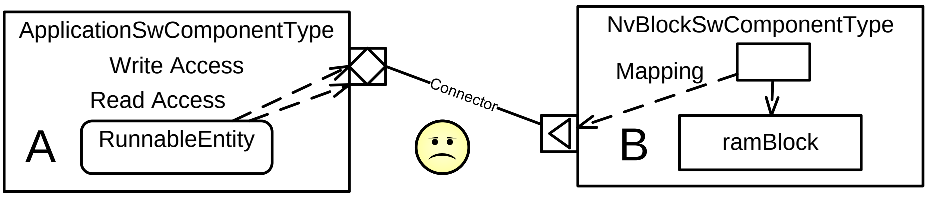


Figure 11.7: Example invalid connection between software-components (c)

The scenario covered by [constr_1418] is depicted in Figure 11.7.

11.4.5 NvBlockDescriptor

[TPS_SWCT_01144] NvBlockDescriptor specifies the properties of exactly one NVRAM Block ⌈A NvBlockDescriptor specifies the properties of exactly one NVRAM Block of a NvBlockSwComponentType.

It contains information about the requested NVRAM Block configuration of the *NVRAM Manager*, ramBlock and romBlock, the mapping between the PortPrototypes of the NvBlockSwComponentType and the data inside a ramBlock as well as the role of the clientServerPorts expressed in terms of RoleBasedPortAssignment.⌋ ()

Class	**NvBlockDescriptor**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Specifies the properties of exactly on NVRAM Block.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, NvBlockSwComponentType.nvBlockDescriptor			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clientServerPort	RoleBasedPort Assignment	*	aggr	The RoleBasedPortAssignement defines which client server port of the NvBlockSwComponentType serves for which kind of service or notification. In case of notifications one common callback function is provided by the RTE for each individual kind of notification defined by the "role". The aggregation of RoleBasedPortAssignment is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=clientServerPort, clientServerPort.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**NvBlockDescriptor**			
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for the particular NVRAM Block **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular NVRAM Block. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=dataTypeMapping
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of InstantiationDataDefProps are the refinement of some data def properties of individual instantiations within the context of a NvBlockSw ComponentType. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of ports, component internal memory objects and those attributes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch EventTriggered Activity	ModeSwitchEvent TriggeredActivity	*	aggr	This represents the collection of ModeSwitchEvent TriggeredActivities related to the enclosing NvBlock Descriptor. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSwitchEventTriggeredActivity, mode SwitchEventTriggeredActivity.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
nvBlockData Mapping	NvBlockDataMapping	*	aggr	Defines the mapping between the VariableData Prototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The aggregation of NvBlockDataMapping is subject to variability with the purpose to support the conditional existence of nv data ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=nvBlockDataMapping, nvBlockData Mapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
nvBlockNeeds	NvBlockNeeds	0..1	aggr	Specifies the abstract needs on the configuration of the NVRAM Manager for the single NVRAM Block described by this NvBlockDescriptor. In addition, it may define requirements for writing strategies in an implementation of an NvBlockSw ComponentType by the RTE. Please note that the attributes nDataSets and nRom Blocks are not relevant for this aggregation because the RTE will allocate just one block anyway. In a different context, however, they do make sense.
ramBlock	VariableDataPrototype	0..1	aggr	Defines the RAM Block of the NVRAM Block provided by NvBlockSwComponentType.
romBlock	ParameterData Prototype	0..1	aggr	Defines the ROM Block of the NVRAM Block provided by NvBlockSwComponentType.

▽

△

Class	NvBlockDescriptor			
supportDirty Flag	Boolean	0..1	attr	Specifies whether calling of NvM functions for writing and/ or status control of potentially modified RAM Blocks to NV memory shall be controlled by the RTE.
timingEvent	TimingEvent	0..1	ref	this reference can be taken to identify the TimingEvent to be used by the RTE for implementing a cyclic writing strategy for this block
writingStrategy	RoleBasedData Assignment	*	aggr	This attribute allows for assigning a specific writing strategy for an incoming AutosarDataPrototype.

Table 11.8: NvBlockDescriptor

[constr_1981] Existence of attribute `NvBlockDescriptor.nvBlockNeeds` ⌈For each `NvBlockDescriptor`, attribute `nvBlockNeeds` shall exist **`at the time when the RTE is generated`**.⌋*()*

For more explanation about the semantics of the attribute `NvBlockDescriptor.supportDirtyFlag` please refer to the SWS RTE [2].

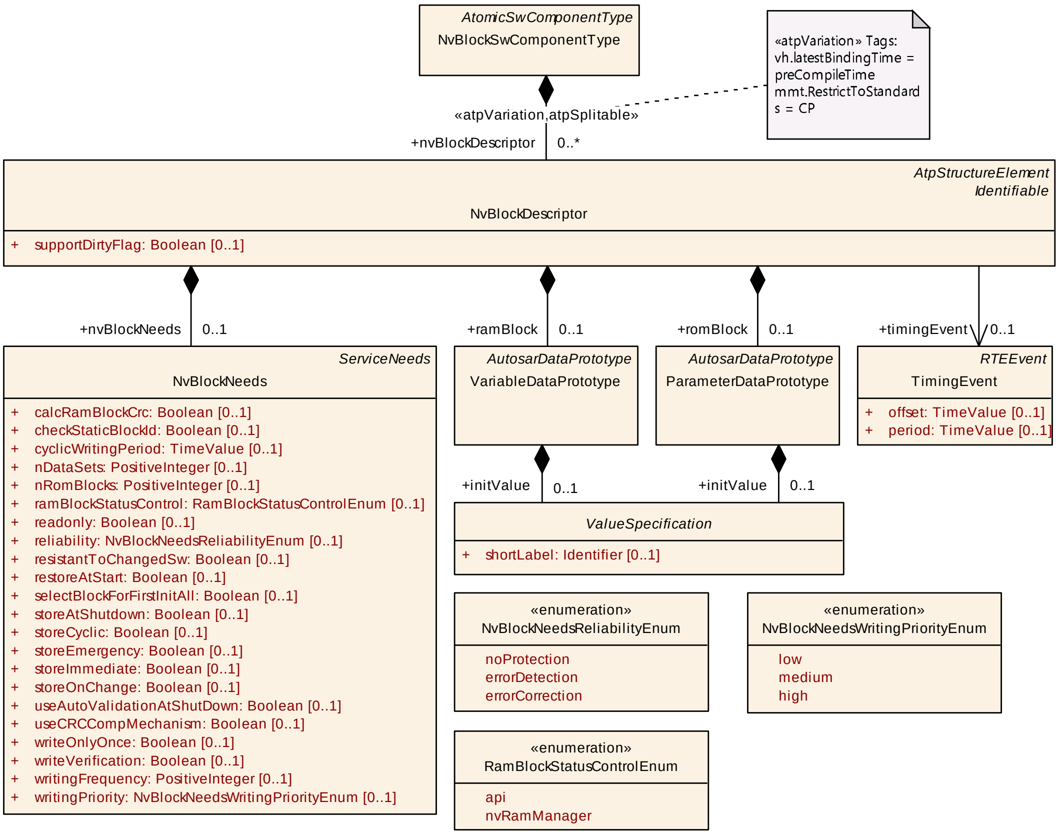


Figure 11.8: `NvBlockSwComponentType` and `NvBlockDescriptor`

[constr_1095] Values of `nDataSets` vs. `reliability` ⌈If the value of `nDataSets` is greater than 0, the value of `reliability` shall not be set to `errorCorrection` **`at the time when the RTE is generated`**.⌋*()*

The reason for the existence of [constr_1095] is that the AUTOSAR NvM [37] does not support error correction for NV data sets.

If the value of nDataSets is equal to 0, the value of reliability can take any value out of NvBlockNeedsReliabilityEnum.

11.4.5.1 Writing Strategies

[TPS_SWCT_01586] Writing strategies for *nv data* ⌈By setting certain attributes in the meta-class NvBlockDescriptor it is possible to configure different writing strategies for the values of an RAM Block to the NVRAM storage. [constr_1310] applies.

The following use cases are supported:

- Write data **cyclically**. This use case requires the existence of attribute NvBlockDescriptor.nvBlockNeeds.storeCyclic with the value true and also attribute NvBlockDescriptor.nvBlockNeeds.cyclicWritingPeriod needs to exist and have a reasonable value.

  In the context of using the attribute NvBlockDescriptor.nvBlockNeeds.cyclicWritingPeriod the constraints [constr_1308] and [constr_1309] apply.

- Write data **immediately**. This means that data send to the NvBlockSwComponentType will be written immediately to NVRAM storage.

  This use case corresponds to setting the value of attribute NvBlockDescriptor.nvBlockNeeds.storeImmediate to the value true.

- Write on **emergency**. With this setting, data shall be written to NVRAM storage if the ECU fails in some way.

  This use case corresponds to setting the value of attribute NvBlockDescriptor.nvBlockNeeds.storeEmergency to true.

  As explained in [TPS_SWCT_01589], setting the value of this attribute is not sufficient to achieve the intended semantics.

- Write at **shutdown**. Here, the data is written to NVRAM storage when the ECU shuts down.

  This use case corresponds to setting the value of attribute NvBlockDescriptor.nvBlockNeeds.storeAtShutdown to true.

- Write on **mode switch**. Here, the data is written to NVRAM in response to a mode switch configured to trigger the writing.

  This use case corresponds to the existence of attribute NvBlockDescriptor.modeSwitchEventTriggeredActivity.

- Write on **change**. Here, the data is written to NVRAM if the value is different than the respective value in the ramMirror.

  This use case corresponds to setting the value of attribute NvBlockDescriptor.nvBlockNeeds.storeOnChange to true.

⌋(*RS_SWCT_03225*)

Please refer to [TPS_SWCT_01587] and Figure 11.9 for more information about how the use case to write data cyclically can be configured.

Please refer to [TPS_SWCT_01588] and Figure 11.10 for more information about how the use case to write data immediately can be configured.

Of course, the actual implementation of the different writing strategies goes beyond setting the value of attributes and requires the existence of dedicated `RunnableEntity`s in the `SwcInternalBehavior` of the enclosing `NvBlockSwComponentType` that are triggered in response to `RTEEvent`s applicable for the particular use case.

[TPS_SWCT_01587] The cyclic writing of *nv data* requires the existence of a `TimingEvent` ⌈The implementation of cyclic writing of *nv data* requires the existence of a `TimingEvent` that can be taken to trigger a corresponding `RunnableEntity` that in turn takes care of calling the respective APIs for writing the data.⌋(*RS_SWCT_03225*)

This aspect is depicted in Figure 11.9.

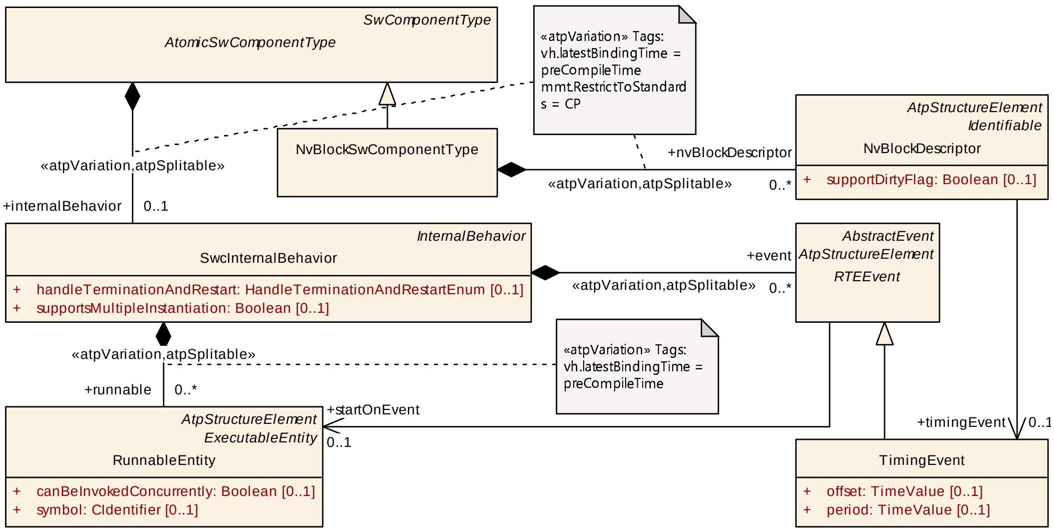


Figure 11.9: How to model a cyclic writing strategy for *nv data*

[TPS_SWCT_01588] `DataReceivedEvent` for storing *nv data* immediately ⌈The approach to store data immediately after reception by an `NvBlockSwComponentType` requires the activation of a `RunnableEntity` by a `DataReceivedEvent`.⌋(*RS_SWCT_03225*)

This approach is depicted in Figure 11.10.

[TPS_SWCT_01589] Implementation of emergency storing of *nv data* ⌈The use case for `storeEmergency` can only be implemented by means of a `Complex Driver`.

In particular, the Complex Driver is responsible for the detection of an ECU failure. If a relevant error occurs the Complex Driver should call the NvM write block operation for the emergency blocks directly.⌋*(RS_SWCT_03225)*

This consequently means that the NvM shall react to write operations coming from the Complex Driver by giving them the highest priority (re-queuing of NvM write block requests).

Please note that the behavior described in [TPS_SWCT_01587] in general is supported by AUTOSAR by requiring that NVRAM Blocks shall have to be configured with "immediate priority". The technical implications are explained in the respective SWS [37], e.g. in [SWS_NvM_00182] and [SWS_NvM_00300].

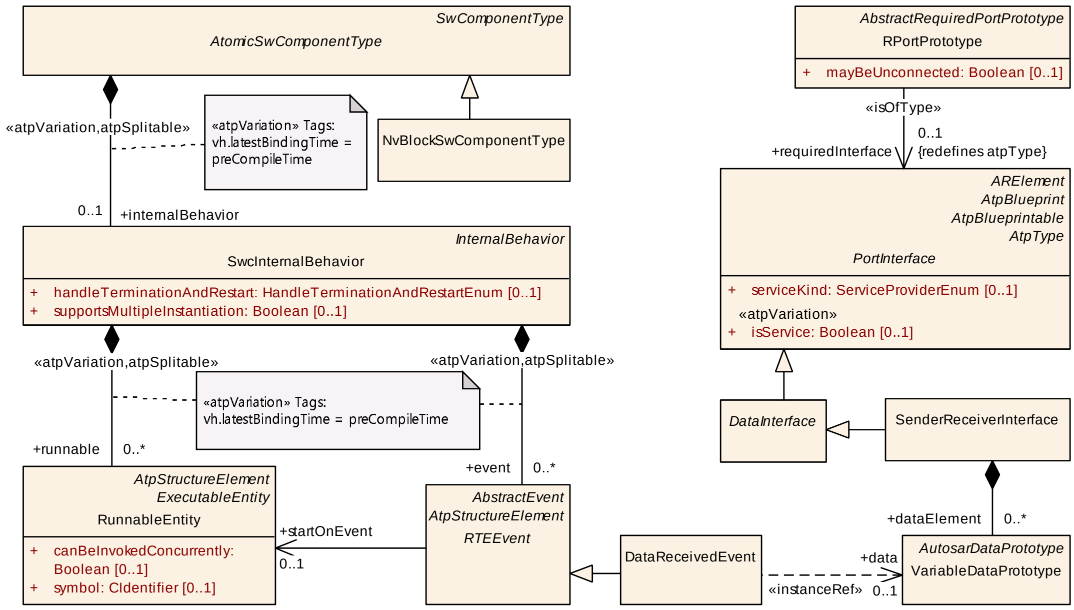


Figure 11.10: How to model an immediate writing strategy for *nv data*

[TPS_SWCT_01590] Combination of writing strategies for *nv data* is possible
⌈AUTOSAR positively supports the configuration of a combination of writing strategies for *nv data*.⌋*(RS_SWCT_03225)*

In other words, in consequence of [TPS_SWCT_01590] it is possible that (for example) both NvBlockDescriptor.nvBlockNeeds.storeImmediate and NvBlockDescriptor.nvBlockNeeds.storeCyclic may exist and set to true in the context of the same NvBlockNeeds.

[TPS_SWCT_01665] Usage of SwcModeSwitchEvent for triggering a write procedure of *nv data* ⌈The approach to manage data of an NvBlockSwComponentType in response to a mode switch notification received from a mode manager requires the activation of a RunnableEntity by a SwcModeSwitchEvent.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01666] Semantics of ModeSwitchEventTriggeredActivity.role
⌈If the role ModeSwitchEventTriggeredActivity.role is set to the value WriteBlock then NvM gets requested to write the *nv data* block after the corresponding SwcModeSwitchEvents occurs.⌋*(RS_SWCT_03225)*

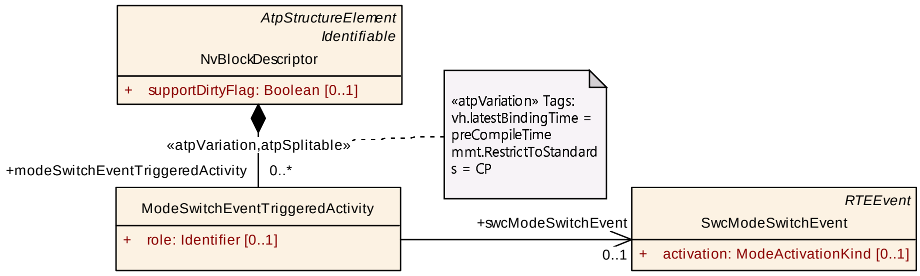


Figure 11.11: Usage of mode switch notification for the activation of a write-procedure of *nv data*

[constr_1415] Supported values of ModeSwitchEventTriggeredActivity.role ⌈The only supported value of ModeSwitchEventTriggeredActivity.role **at the time when the RTE is generated** is WriteBlock.⌋*()*

Class	**ModeSwitchEventTriggeredActivity**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	This meta-class defines an activity of the NvBlockSwComponentType for a specific NvBlock which is triggered by a ModeSwitchEvent.			
Base	*ARObject*			
Aggregated by	NvBlockDescriptor.modeSwitchEventTriggeredActivity			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
role	Identifier	0..1	attr	This attribute indicates which service of the NvM for the NvBlock shall be requested.
swcModeSwitchEvent	SwcModeSwitchEvent	0..1	ref	This reference identifies the SwcModeSwitchEvent that triggers the activity.

Table 11.9: ModeSwitchEventTriggeredActivity

[constr_1982] Existence of attribute ModeSwitchEventTriggeredActivity.role ⌈For each ModeSwitchEventTriggeredActivity, attribute role shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1983] Existence of attribute ModeSwitchEventTriggeredActivity.swcModeSwitchEvent ⌈For each ModeSwitchEventTriggeredActivity, attribute swcModeSwitchEvent shall exist **at the time when the RTE is generated**.⌋*()*

11.4.5.1.1 Existence of multiple event-driven Writing Strategies

A use case might exist where (potentially in addition to a cyclic writing strategy) multiple event-based writing strategies to NvRam exist at the same time.

Writing strategies are typically implemented by means of different RunnableEntitys that need to be triggered by the arrival of data at the NvBlockSwComponentType. In this case, however, it is necessary to be able to assign the relation between incoming data and the RunnableEntity that applies the fitting writing strategy.

In the example scenario depicted in Figure 11.12, two dataElements are received by the NvBlockSwComponentType. One dataElement shall be associated with the writing strategy to *store at shutdown* and the other dataElement shall be associated with an *immediate storage*.

Two DataReceivedEvents exist that can be taken to trigger RunnableEntitys. But without further formal specification, it is up to speculation which DataReceivedEvent shall be associated with which RunnableEntity.

To remove this degree of uncertainty, it is possible to use standardized values for the attribute RoleBasedDataAssignment.role and let the RoleBasedDataAssignment itself refer to the respective dataElement.

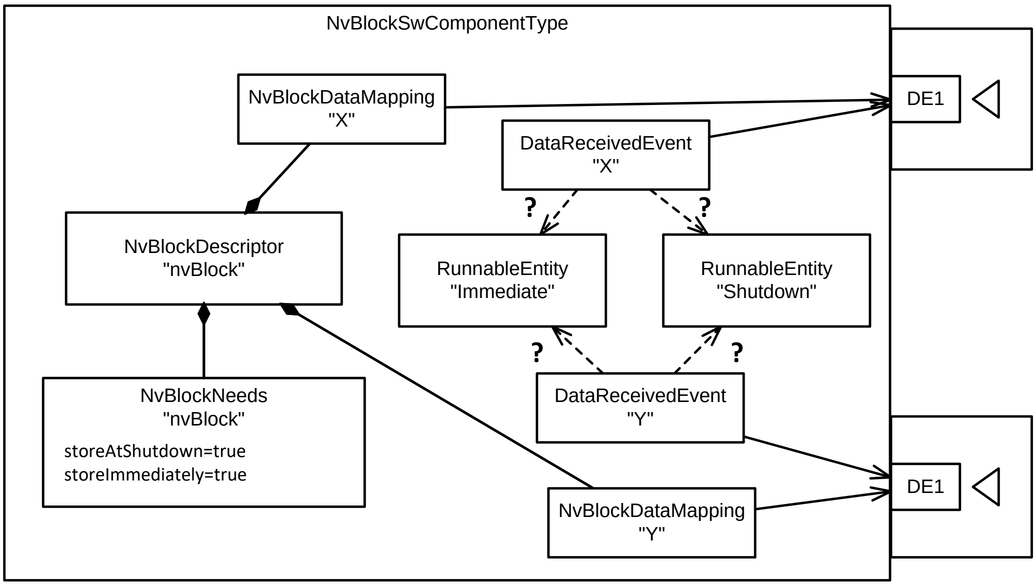


Figure 11.12: Existence of several event-based writing strategies for the same NvBlockDescriptor

The approach is sketched in Figure 11.13 and its application to the depicted scenario is sketched in Figure 11.14.

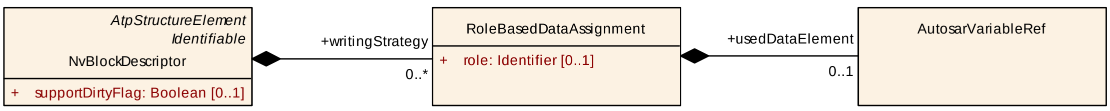


Figure 11.13: Modeling of RoleBasedDataAssignment in the context of NvBlockDescriptor

[TPS_SWCT_01795] Further specification to facilitate the association of writing strategies to the corresponding RunnableEntity ⌈NvBlockDescriptor may aggregate RoleBasedDataAssignment in the role writingStrategy and the attribute NvBlockDescriptor.writingStrategy.role can be used to assign a writing strategy to an AutosarDataPrototype (referenced by means of NvBlockDescriptor.writingStrategy.usedDataElement) in the context of a PortPrototype.⌋*(RS_SWCT_03225)*

[constr_1713] NvBlockDescriptor.writingStrategy.usedDataElement shall refer to AutosarDataPrototype ⌈The reference NvBlockDescriptor.writingStrategy.usedDataElement shall **only** refer to an AutosarDataPrototype **at the time when the RTE is generated**.⌋*()*

Rationale: the activation of a RunnableEntity that implements the respective writing strategy is triggered by the reception of an entire AutosarDataPrototype. The ability to refer to a DataPrototype inside the AutosarDataPrototype might give a false impression.

[constr_1714] AutosarDataPrototype shall only be referenced by a single NvBlockDescriptor.writingStrategy ⌈If an AutosarDataPrototype in the context of a PortPrototype is referenced from a NvBlockDescriptor.writingStrategy then this AutosarDataPrototype shall not be referenced from any other NvBlockDescriptor.writingStrategy.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

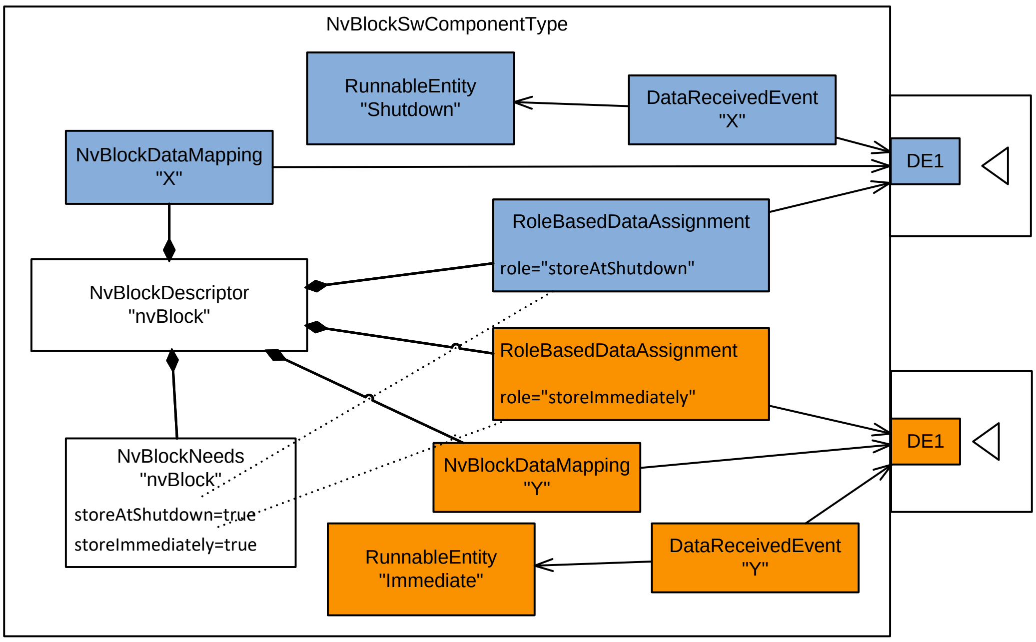


Figure 11.14: Solution for the existence of several event-based writing strategies for the same NvBlockDescriptor

[constr_1715] Possible values of attribute NvBlockDescriptor.writingStrategy.role ⌈The attribute NvBlockDescriptor.writingStrategy.role shall only have one of the following values **at the time when the RTE is generated** (see [TPS_SWCT_01586]):

- **storeAtShutdown**
- **storeImmediate**
- **storeOnChange**

⌋()

[constr_1716] Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeAtShutdown ⌈The existence of NvBlockDescriptor.writingStrategy where attribute role is set to storeAtShutdown is only supported if NvBlockDescriptor.nvBlockNeeds.storeAtShutdown exists and is set to true **at the time when the RTE is generated**.⌋()

[constr_1717] Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeImmediate ⌈The existence of NvBlockDescriptor.writingStrategy where attribute role is set to storeImmediate is only supported if NvBlockDescriptor.nvBlockNeeds.storeImmediate exists and is set to true **at the time when the RTE is generated**.⌋()

[constr_10074] Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeOnChange ⌈The existence of NvBlockDescriptor.writingStrategy where attribute role is set to storeOnChange is only supported if NvBlockDescriptor.nvBlockNeeds.storeOnChange exists and is set to true **at the time when the RTE is generated**.⌋()

11.4.5.2 NvBlockNeeds

The requested NVRAM Block configuration of the *NVRAM Manager* is described by the NvBlockNeeds of the NvBlockDescriptor.

This information can be evaluated during ECU configuration similar to the NvBlockNeeds of an atomic software component or a BSW module. For further details see section 13.

[constr_1310] Existence of attributes of meta-class NvBlockNeeds ⌈If in the context of an ApplicationSwComponentType the attribute SwcServiceDependency.serviceNeeds is implemented by an NvBlockNeeds then the following attributes

- NvBlockNeeds.storeCyclic
- NvBlockNeeds.cyclicWritingPeriod
- NvBlockNeeds.storeEmergency
- NvBlockNeeds.storeImmediate
- NvBlockNeeds.storeOnChange

shall only exist if in the context of the same SwcServiceDependency a SwcServiceDependency.assignedPort exists that has the attribute role set to the value NvDataPort.

This rule shall be imposed **at the time when the RTE is generated**.⌋()

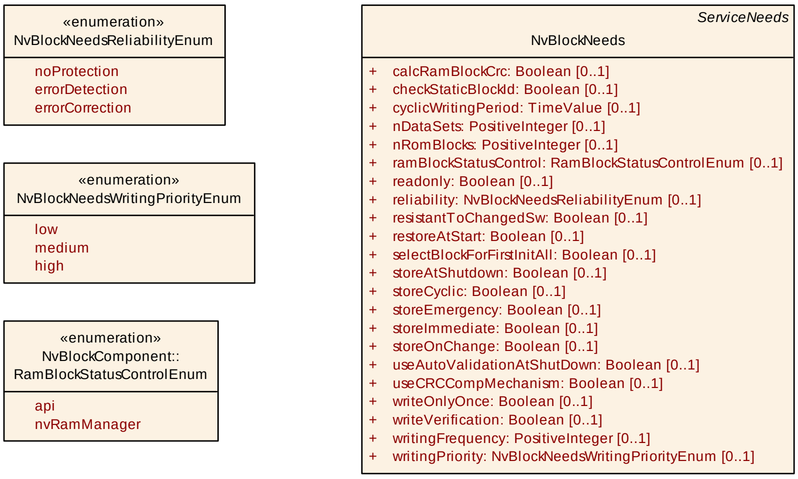


Figure 11.15: NvBlockNeeds

[constr_1308] Existence of NvBlockNeeds.cyclicWritingPeriod ⌈The attribute NvBlockNeeds.cyclicWritingPeriod shall exist if and only if the attribute NvBlockNeeds.storeCyclic exists and its value is set to true.

This rule shall be imposed **at the time when the RTE is generated**⌋()

Class	**NvBlockNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of a single NVRAM Block.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, NvBlockDescriptor.nvBlockNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
calcRamBlockCrc	Boolean	0..1	attr	Defines if CRC (re)calculation for the permanent RAM Block is required.
checkStaticBlockId	Boolean	0..1	attr	Defines if the Static Block Id check shall be enabled.
cyclicWritingPeriod	TimeValue	0..1	attr	This represents the period for cyclic writing of NvData to store the associated RAM Block.
nDataSets	PositiveInteger	0..1	attr	Number of data sets to be provided by the NVRAM manager for this block. This is the total number of ROM Blocks and RAM Blocks.
nRomBlocks	PositiveInteger	0..1	attr	Number of ROM Blocks to be provided by the NVRAM manager for this block. Please note that these multiple ROM Blocks are given in a contiguous area.
ramBlockStatusControl	RamBlockStatusControlEnum	0..1	attr	This attribute defines how the management of the RAM Block status is controlled.

▽

△

Class	NvBlockNeeds			
readonly	Boolean	0..1	attr	true: data of this NVRAM Block are write protected for normal operation (but protection can be disabled) false: no restriction
reliability	NvBlockNeeds ReliabilityEnum	0..1	attr	Reliability against data loss on the non-volatile medium.
resistantTo ChangedSw	Boolean	0..1	attr	Defines whether an NVRAM Block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, please refer to the NVRAM specification.
restoreAtStart	Boolean	0..1	attr	Defines whether the associated RAM Block shall be implicitly restored during startup by the basic software.
selectBlockFor FirstInitAll	Boolean	0..1	attr	If this attribute is set to true the NvM shall process this block in the NvM_FirstInitAll() function.
storeAt Shutdown	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored during shutdown by the basic software.
storeCyclic	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored periodically by the basic software.
store Emergency	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored in case of ECU failure (e.g. loss of power) by the basic software. If the attribute store Emergency is set to true the associated RAM Block shall be configured to have immediate priority.
storeImmediate	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored immediately during or after execution of the according SW-C RunnableEntity by the basic software.
storeOnChange	Boolean	0..1	attr	This attribute defines whether the associated RAM Block shall be stored immediately if the written value is different to the value stored in the associated RAM Block(s) during or after execution of the according SW-C RunnableEntity.
useAuto ValidationAt ShutDown	Boolean	0..1	attr	If set to true the RAM Block shall be auto validated during shutdown phase.
useCRCComp Mechanism	Boolean	0..1	attr	If set to true the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job in order to skip unnecessary NVRAM writings.
writeOnlyOnce	Boolean	0..1	attr	Defines write protection after first write: true: This block is prevented from being changed/erased or being replaced with the default ROM data after first initialization by the software-component. false: No such restriction.
writeVerification	Boolean	0..1	attr	Defines if Write Verification shall be enabled for this NVRAM Block.
writing Frequency	PositiveInteger	0..1	attr	Provides the amount of updates to this block from the application point of view. It has to be provided in "number of write access per year".
writingPriority	NvBlockNeedsWriting PriorityEnum	0..1	attr	Requires the priority of writing this block in case of concurrent requests to write other blocks.

Table 11.10: NvBlockNeeds

Enumeration	**NvBlockNeedsReliabilityEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Reliability against data loss on the non-volatile medium. These requirements give only a relative indication, for example on the required degree of redundancy for storage. They do, however, not specify by which means (e.g. software or hardware) the reliability is actually achieved.
Aggregated by	NvBlockNeeds.reliability
Literal	***Description***
errorCorrection	Errors shall be corrected **Tags:**atp.EnumerationLiteralIndex=0
errorDetection	Errors shall be detected **Tags:**atp.EnumerationLiteralIndex=1
noProtection	Data need not to be handled with protection **Tags:**atp.EnumerationLiteralIndex=2

Table 11.11: NvBlockNeedsReliabilityEnum

11.4.5.3 RAM Block and ROM Block

[TPS_SWCT_01145] `ramBlock` and the `romBlock` are described by a `VariableDataPrototype` and a `ParameterDataPrototype` ⌈The `ramBlock` and the `romBlock` are described by a `VariableDataPrototype` and a `ParameterDataPrototype` which are typed by an `AutosarDataType`.⌋*()*

[TPS_SWCT_01146] `romBlock` is optional ⌈The `romBlock` is optional. If a `romBlock` is configured, the RTE copies the `romBlock` constants into the `RAM Block` in case of a block initialization notification (*NvMNotifyInitBlock*).⌋*()*

[TPS_SWCT_01147] No `romBlock` is configured ⌈If there is no `romBlock` configured the connected software components are either required to offer this functionality by a proper implementation of block initialization notification or the `NVRAM Block` has to be configured, that no `ROM Block` is needed.⌋*()*

As a mitigation against a failed read operation from NV memory, it is recommended to always define a `romBlock` with suitable initial values to ensure the proper initialization of the corresponding `ramBlock`.

In particular, for software-components that don't define a `PortPrototype` typed by the `ClientServerInterface` with the standardized `shortName` `NotifyInitBlock` [37] it may happen that the `ramBlock` might not be properly initialized in case of failure.

[constr_2012] Compatibility of `ImplementationDataType`s used for `ramBlock` and `romBlock` ⌈The `ramBlock` and the `romBlock` shall have compatible `ImplementationDataType`s to ensure **`at the time when the RTE is generated`**, that the `NVRAM Block` default values in the `ROM Block` can be copied into the `RAM Block`.⌋*()*

[TPS_SWCT_01848] Semantics of NvBlockDescriptor.instantiationDataDefProps ⌈It is possible that NvBlockDescriptor.ramBlock and NvBlockDescriptor.romBlock shall be able to be calibrated or measured.

This represents another use case for overwriting existing SwDataDefProps (defined on data type level) in the context of NvBlockDescriptor.instantiationDataDefProps to fine-tune the respective MCD-information.⌋*()*

11.4.5.4 NvBlockDataMapping

[TPS_SWCT_01148] NvBlockDataMapping ⌈The meta-class NvBlockDataMapping specifies the mapping of (bits of) DataPrototypes of the NvBlockSwComponentType's PPortPrototypes resp. RPortPrototypes to (bits of) DataPrototypes inside the RAM Block.⌋*()*

This ensures a flexible but deterministic NVRAM Block memory structure given by the ImplementationDataType of the ramBlock and romBlock and its association to the PortPrototypes of the NvBlockSwComponentType.

[constr_2013] Compatibility of ImplementationDataTypes for NvBlockDataMapping ⌈Unless both the attribute bitfieldTextTableMaskNvBlockDescriptor and attribute bitfieldTextTableMaskPortPrototype is defined in the context of a given NvBlockDataMapping, the NvBlockDataMapping is only valid if the ImplementationDataType of the referenced VariableDataPrototype or ImplementationDataTypeElement in the role nvRamBlockElement is compatible to the ImplementationDataType used to type the DataPrototype aggregated by NvBlockDataMapping in the role writtenNvData, writtenReadNvData, or readNvData.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

[constr_1285] Applicability of roles vs. PortPrototypes ⌈The aggregation of AutosarVariableRef aggregated by NvBlockDataMapping in the roles writtenNvData, writtenReadNvData, or readNvData is subject to limitation **at the time when the RTE is generated**, depending on the applicable subclass of PortPrototype:

- The role writtenNvData shall only be used if the corresponding PortPrototype is a RPortPrototype
- The role writtenReadNvData shall only be used if the corresponding PortPrototype is a PRPortPrototype
- The role readNvData shall only be used if the corresponding PortPrototype is a PPortPrototype

⌋*()*

But nevertheless it is valid, that not all ImplementationDataTypeElements within the VariableDataPrototype aggregated by NvBlockDescriptor in the role ramBlock are mapped to a VariableDataPrototype located in a PortPrototype.

This enables to have fill elements or logistic data in the NVRAM Block which are not accessed by software components. This is exemplified by the element *x* in Figure 11.16.

Please note that the VariableDataPrototype located in the PortPrototype, in the vast majority of cases, will be typed by an ApplicationDataType which in turn (at least before the actual code generation starts) finally shall have a mapping to an ImplementationDataType. This aspect is explained in chapter 5.2.2.

[TPS_SWCT_01659] Mapping of VariableDataPrototype to a NvBlockDescriptor ⌈There are three ways to map a VariableDataPrototype (i.e. NvDataInterface.nvData in the context of a specific PortPrototype) to either an NvBlockDescriptor.ramBlock or a *sub-element* thereof:

- NvDataInterface.nvData is directly and completely mapped, i.e. AutosarVariableRef.autosarVariable shall exist and autosarVariable.targetDataPrototype shall refer to the NvDataInterface.nvData.
- **Every** *leaf* element of NvDataInterface.nvData is mapped individually. This means that either
  - AutosarVariableRef.autosarVariableInImplDatatype shall exist and autosarVariableInImplDatatype.targetDataPrototype shall refer to the respective *leaf* element of NvDataInterface.nvData.
  - AutosarVariableRef.autosarVariable shall exist and autosarVariable.targetDataPrototype shall refer to the respective *leaf* element of NvDataInterface.nvData.

  In other words: the mapping shall be defined either via the used ImplementationDataType or else via the used ApplicationDataType.
- A *sub-element* of NvDataInterface.nvData - which is **not** a *leaf* element - may be directly mapped and consequently **all** the *leaf* elements of the respective *sub-element* of NvDataInterface.nvData are **indirectly mapped** as well. This means that
  - AutosarVariableRef.autosarVariableInImplDatatype shall exist and autosarVariableInImplDatatype.targetDataPrototype shall refer to the *sub-element* element of NvDataInterface.nvData.
  - AutosarVariableRef.autosarVariable shall exist and autosarVariable.targetDataPrototype shall refer to the *sub-element* element of NvDataInterface.nvData.

  In other words: the mapping shall be defined either via the used ImplementationDataType or else via the used ApplicationDataType.

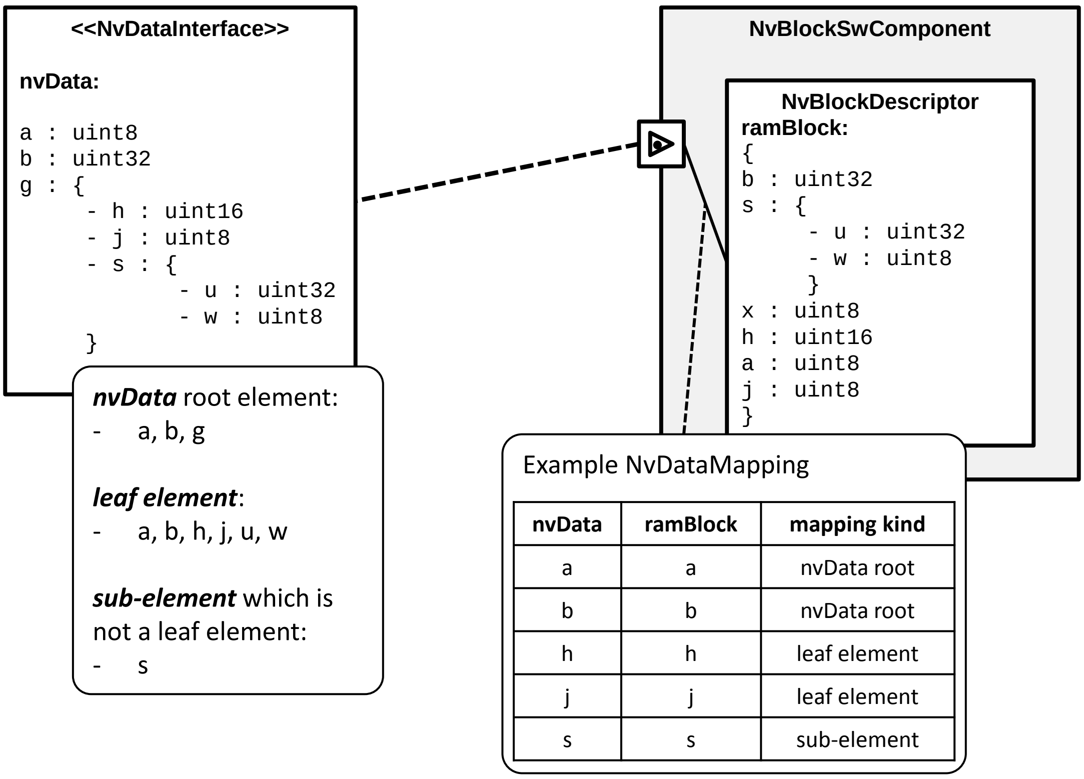
⌋()

Please note that a mixing of **mutually exclusive** mappings for entire *sub-elements* or *leaf* elements as described by [TPS_SWCT_01659] is positively supported (see Figure 11.16).

[constr_1395] NvBlockDataMapping shall be complete ⌈If an NvBlockDataMapping refers to *sub-elements* or *leaf* elements of the NvDataInterface.nvData in the context of a particular PortPrototype, then **all remaining** *sub-elements* or *leaf* elements **shall effectively be mapped** according to [TPS_SWCT_01659] by means of a collection of NvBlockDataMappings **at the time when the RTE is generated**.⌋()

[constr_1403] NvBlockDataMappings to a given nvData shall be unambiguous ⌈If an NvBlockDataMapping exists that **directly** and **completely** maps a specific NvDataInterface.nvData in the context of a particular PortPrototype, then **no** other NvBlockDataMapping which maps sub-elements of the NvDataInterface.nvData shall exist **at the time when the RTE is generated**.⌋()

The interaction with AUTOSAR services is centrally defined in the context of the SwcServiceDependency. The latter gathers a collection of PortPrototypes by means of RoleBasedPortAssignments that implement a closely related service functionality.



nvData	ramBlock	mapping kind
a	a	nvData root
b	b	nvData root
h	h	leaf element
j	j	leaf element
s	s	sub-element

Figure 11.16: Example NvBlockDataMapping to explain the three ways to map a VariableDataPrototype to either an NvBlockDescriptor.ramBlock or a sub-element thereof

In the specific case of interaction between AtomicSwComponentType and NvBlockSwComponentType (as described by [TPS_SWCT_02503]), there are PortPrototypes referenced by a RoleBasedPortAssignment with attribute RoleBasedPortAssignment.role set to NvDataPort. These PortPrototypes contain the collected *Nv Data* of the service use case.

Furthermore, there is the possibility to receive notifications when the writing of the mapped NV Block to the NvRam is finished.

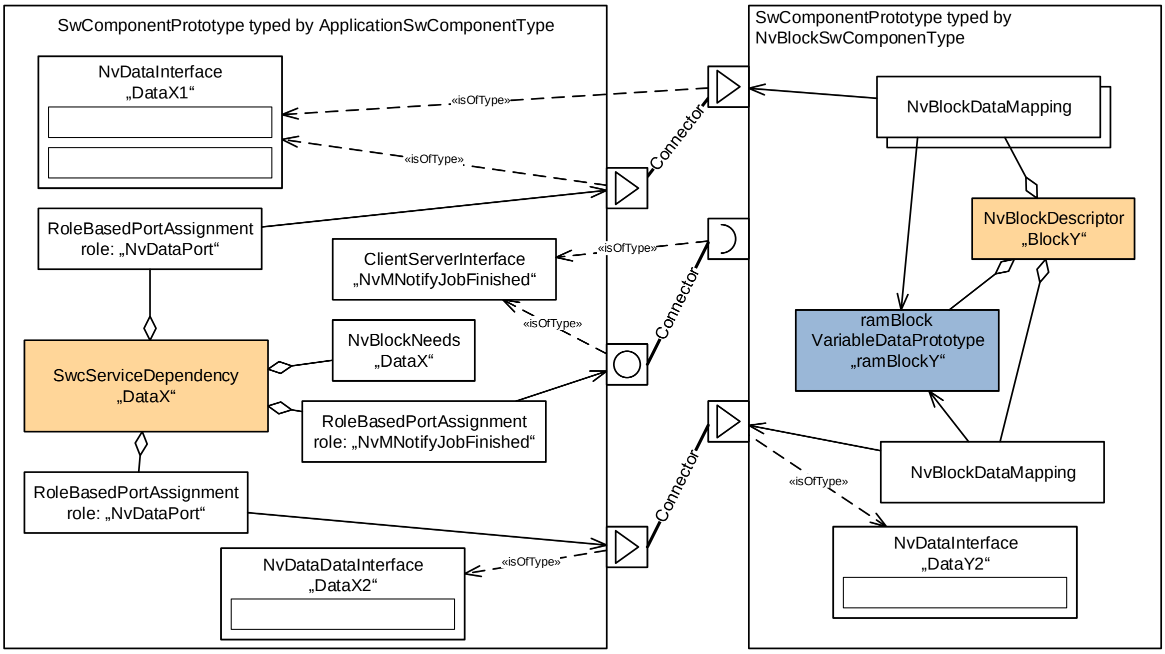


Figure 11.17: Visualization of the statement made by [constr_1404]

[TPS_SWCT_01863] Existence of VariableInAtomicSWCTypeInstanceRef.rootVariableDataPrototype in the context of NvBlockDataMapping.nvRamBlockElement ⌈In the context of NvBlockDataMapping.nvRamBlockElement, the reference in the VariableInAtomicSWCTypeInstanceRef.rootVariableDataPrototype is redundant because the information carried by this reference is already conveyed by the aggregation NvBlockDescriptor.ramBlock.

Therefore, VariableInAtomicSWCTypeInstanceRef.rootVariableDataPrototype shall be ignored in the context of NvBlockDataMapping.nvRamBlockElement.⌋*()*

In order to be able to properly assign such a notification to the content of the related *Nv Data* PortPrototypes in the scope of the same SwcServiceDependency it is necessary that the *Nv Data* of **all** these PortPrototypes is mapped to the **same** Nv Block (because the notifications are created **per block**).

This aspect represents one motivation for the existence of [constr_1404]:

[constr_1404] All NvDataInterface.nvData of PortPrototypes in the context of a specific SwcServiceDependency shall be mapped to the same NvBlockDescriptor ⌈In the context of a given SwcServiceDependency (which, in turn, is owned by an AtomicSwComponentType), **all** NvDataInterface.nvData of PortPrototypes referenced by a RoleBasedPortAssignment with attribute RoleBasedPortAssignment.role set to NvDataPort shall be connected (either directly or via the definition of suitable PortInterfaceMappings) to NvDataInterface.nvData (on the side of the NvBlockSwComponentType) that are **completely mapped** (via NvBlockDataMappings) **to the identical** NvBlockDescriptor.ramBlock.

This rule shall be imposed **at the time when the RTE is generated.**⌋*()*

The statement made by [constr_1404] is visualized in Figure 11.17. The context-defining model elements, i.e. SwcServiceDependency owned by the AtomicSwComponentType as well as NvBlockDescriptor owned by the NvBlockSwComponentType, are colored in light orange.

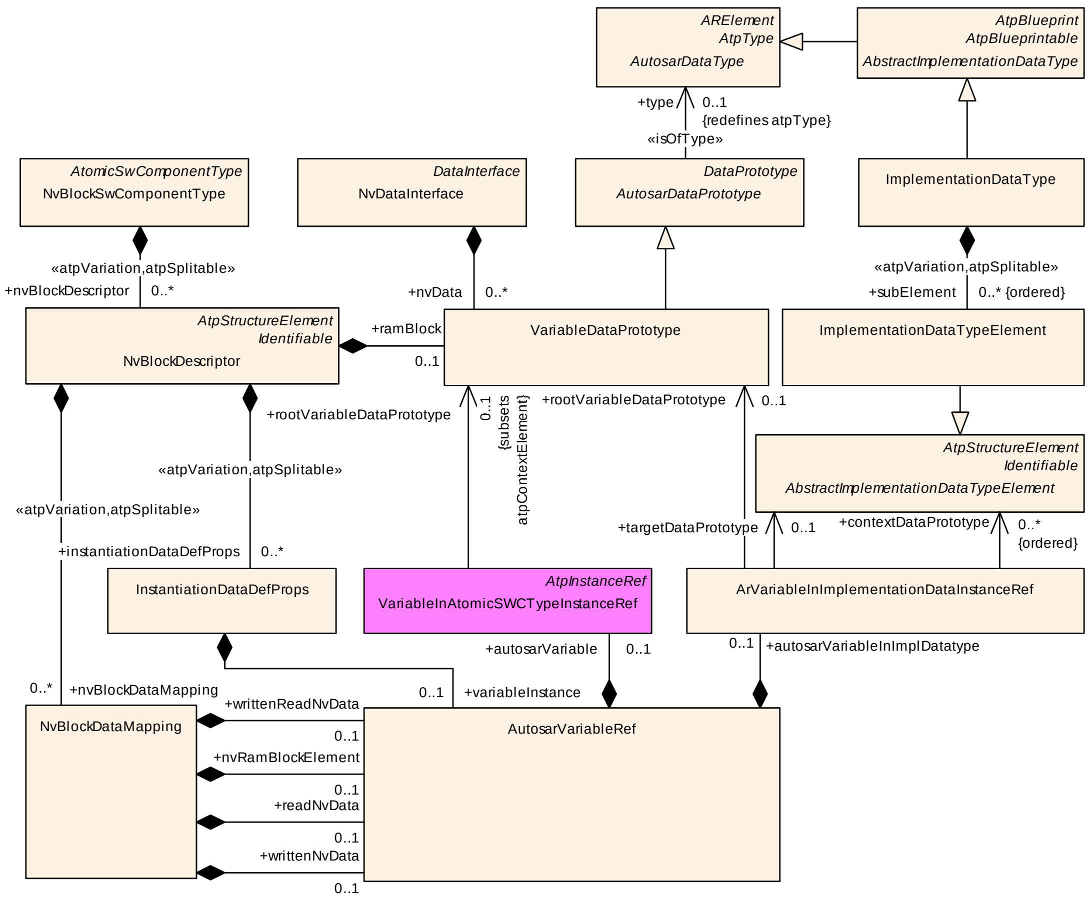


Figure 11.18: NvBlockDataMapping and InstantiationDataDefProps

The diagram is focused on the NvBlockDescriptor.ramBlock. As stressed by [constr_1404], all *Nv Data* provided by the PortPrototypes referenced by the specific SwcServiceDependency finally ends up in the one depicted ramBlock (colored in blue).

Please note that the graphical representation of the NvBlockDataMapping in Figure 11.17 has been simplified for the sake of clarity.

In summary, there is in fact a whole list of motivations that lead to the creation of [constr_1404]:

- The handling of the dirty-flag within the RTE is done in the context of a PortPrototype.
- The currently implemented approach allows for an easy understanding of the relation of data in the application to data in NV-RAM.
- All data in the context of one SwcServiceDependency have the identical storage life cycle.
- Jobs triggered in the NvM via operations in the NvMService (see [SWS_NvM_00734]) interface belong to exactly one NvBlockDescriptor.
- As already mentioned, notifications of the NvM towards software components (e.g. NvMNotifyJobFinished as described in [SWS_NvM_00735] or NvMNotifyInitBlock, see [SWS_NvM_00736]) belong to exactly one NvBlockDescriptor.

[constr_1984] Existence of instance reference NvBlockDataMapping.nvRamBlockElement ⌈For each NvBlockDataMapping, the instance reference to ModeDeclaration in the role nvRamBlockElement shall exist **at the time when the RTE is generated.**⌋*()*

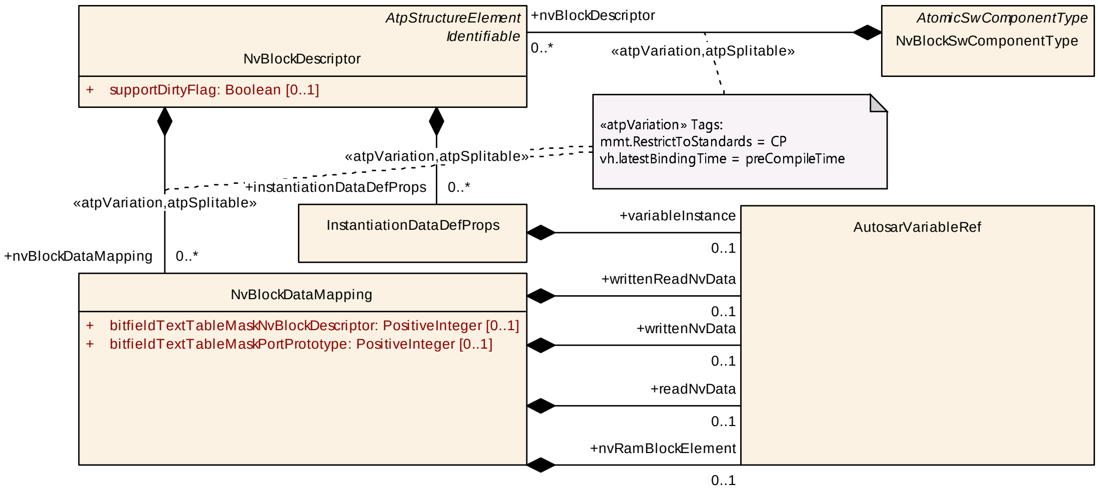


Figure 11.19: Attributes of NvBlockDataMapping

Please note that are use cases for numerous fine-grained (e.g. single bits) data in non-volatile RAM. For this purpose, it would be very inefficient to copy the value of a whole DataPrototype that is at least 8 bits long into the non-volatile RAM.

A much better approach is to allow for the mapping of single bits or bit-fields inside a DataPrototype in a PortPrototype to the ramBlock of an NvBlockDescriptor directly.

For this purpose the NvBlockDataMapping provides the attributes bitfieldTextTableMaskNvBlockDescriptor and bitfieldTextTableMaskPortPrototype[4].

[TPS_SWCT_01799] Mapping of bit-fields between NvBlockDescriptor and PortPrototype ⌈It is possible to map a bitfield contained in the DataPrototype representing the ramBlock to a bitfield contained in a DataPrototype defined in the context of a PortPrototype on the surface of the enclosing NvBlockSwComponentType by means of the definition of attributes NvBlockDataMapping.bitfieldTextTableMaskNvBlockDescriptor and NvBlockDataMapping.bitfieldTextTableMaskPortPrototype.

In this case [constr_2013] is not applicable.⌋*()*

The general approach for mapping bit-fields onto each other in the context of an NvBlockSwComponentType is sketched in Figure 11.20.

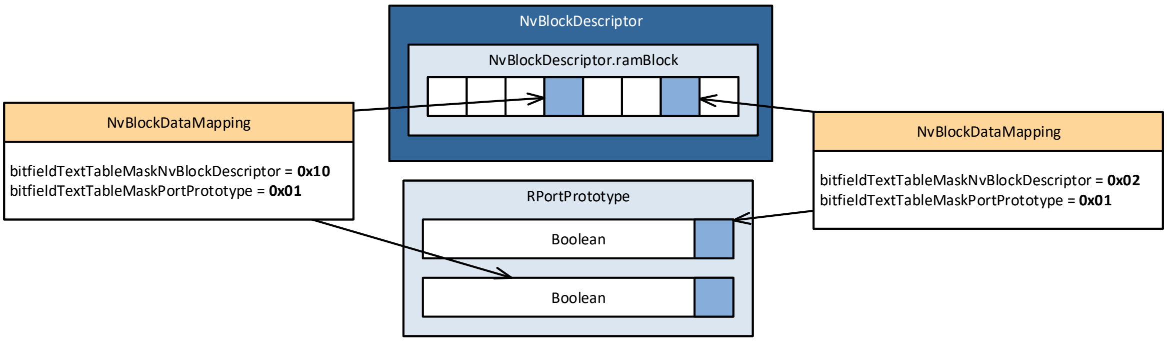


Figure 11.20: Visualization of the approach to map bit-fields inside an NvBlockSwComponentType

Class	**NvBlockDataMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Defines the mapping between the VariableDataPrototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The data types of the referenced VariableDataPrototypes in the ports and the referenced sub-element (inside a CompositeDataType) of the VariableDataPrototype representing the RAM Block shall be compatible.			
Base	*ARObject*			
Aggregated by	BulkNvDataDescriptor.nvBlockDataMapping, NvBlockDescriptor.nvBlockDataMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

[4]The alternative to this approach would have been to shift this "distillation process" into the connection between e.g. an ApplicationSwComponentType and an NvBlockSwComponentType.

Such an approach comes with considerably heavier side effects and complexity (usage in DelegationSwConnectors or as part of a network connection) and is therefore not supported.

△

Class	NvBlockDataMapping			
bitfieldTextTableMaskNvBlockDescriptor	PositiveInteger	0..1	attr	This attribute identifies the applicable bit mask on the side of the Nv Block.
bitfieldTextTableMaskPortPrototype	PositiveInteger	0..1	attr	This attribute identifies the applicable bit mask on the side of the PortPrototype.
nvRamBlockElement	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a RAM Block.
readNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a pPort of the NvBlockComponent providing read access to the RAM Block.If there is no PortPrototype providing read access (write-only) the reference can be omitted.
writtenNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a rPort of the Nv BlockComponent providing write access to the RAM Block. If there is no port providing write access (read-only) the reference can be omitted.
writtenReadNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a PRPort Prototype of the NvBlockSwComponentType providing write and read access to the RAM Block.

Table 11.12: NvBlockDataMapping

11.4.5.5 Client Server Ports

[TPS_SWCT_01149] RoleBasedPortAssignment of NvBlockDescriptor ⌈The clientServerPort of the NvBlockDescriptor describes which client/server PortPrototype of the NvBlockSwComponentType serves for which purpose. The role specifies if the port serves for block-related services, administrative services or notification.⌋*()*

[constr_2014] Limitation of NvBlockDescriptor.clientServerPort.role ⌈The value of attribute NvBlockDescriptor.clientServerPort.role shall be set to a valid name of one of the Standardized AUTOSAR (client/server) Interfaces used for the NVRAM Manager, as described by [TPS_SWCT_02501], [TPS_SWCT_02502], [TPS_SWCT_02503] and [TPS_SWCT_02504].⌋*()*

Rationale for the existence of [constr_2014]: the job of the NvBlockDescriptor.clientServerPort is to qualify the PortPrototypes on the NvBlockSwComponentType that interact with the PortPrototypes described by the service use cases documented in [TPS_SWCT_02501], [TPS_SWCT_02502], [TPS_SWCT_02503] and [TPS_SWCT_02504].

In case of notifications one common callback function is provided by the RTE for each individual kind of notification defined by the attribute role.

The PPortPrototypes related to a given NvBlockDescriptor need to be provided with the same value for a PortDefinedArgumentValue in order to make the software work correctly. The provision of the PortDefinedArgumentValue is heuristic, but with a further "trick" the reliability of this operation can be much improved.

For all NvBlockDescriptor.clientServerPort of a given NvBlockDescriptor where attribute role is set to the value NvMService or NvMAdmin collect the PortPrototype if it is a PPortPrototype. The resulting collection of PortPrototypes need to be provided with the same value (of PortDefinedArgumentValue).

In this case it is no longer necessary to explicitly model the existence of PortDefinedArgumentValues for these PortPrototypes since the applicable id value of the NvM can be derived by RTE configuration.

To make this approach work the role value needs to be standardized, see [constr_2014].

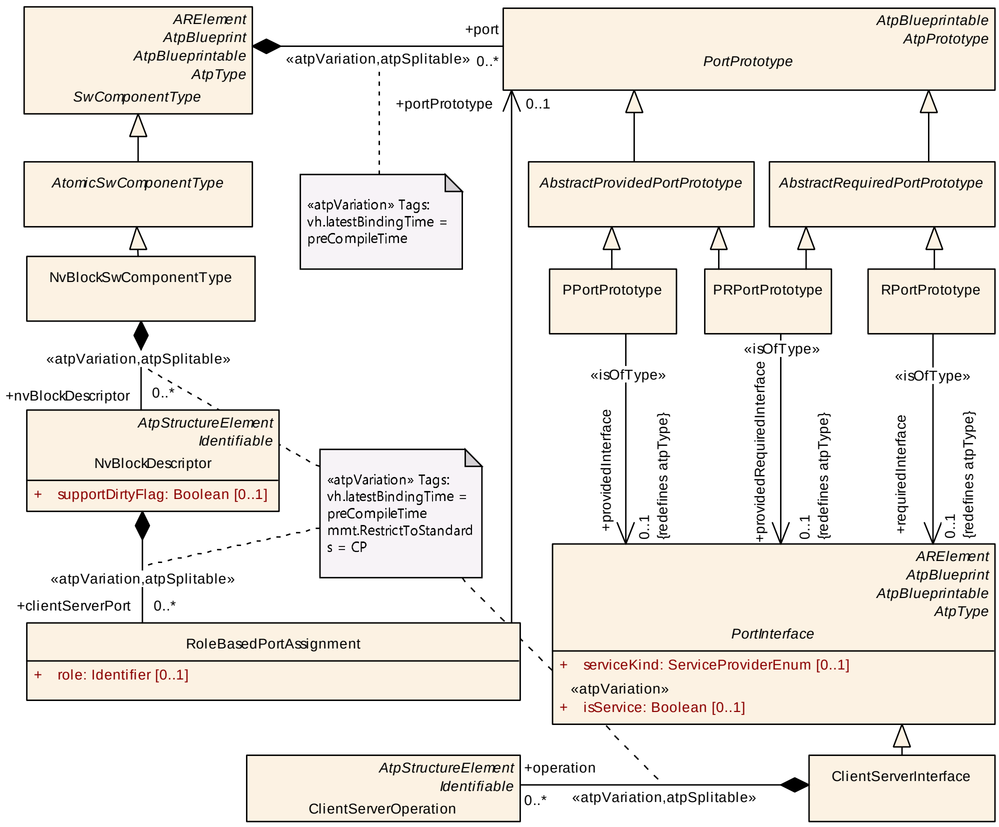


Figure 11.21: NvBlockNotification

11.4.6 BulkNvDataDescriptor

There is a strong use case where application software gets read-only access to data of significant size located in non-volatile memory. The actual data is created or updated e.g. at the end of the production line or maybe during maintenance in a garage.

If such a scenario were to be implemented using an NvBlockDescriptor then the creation of a RAM mirror of the data in the non-volatile memory would be obligatory. But as described before, the existence of **a RAM mirror would not be necessary at all** in this use case and would only waste a significant amount of precious resources.

It is therefore necessary to be clearly separate the use case for utilizing an NvBlockDescriptor from the direct read-only access to non-volatile memory in the meta-model.

For this purpose, model support is provided by the existence of meta-class BulkNvDataDescriptor, aggregated at NvBlockSwComponentType in the role bulkNvDataDescriptor. This modeling is depicted in Figure 11.22.

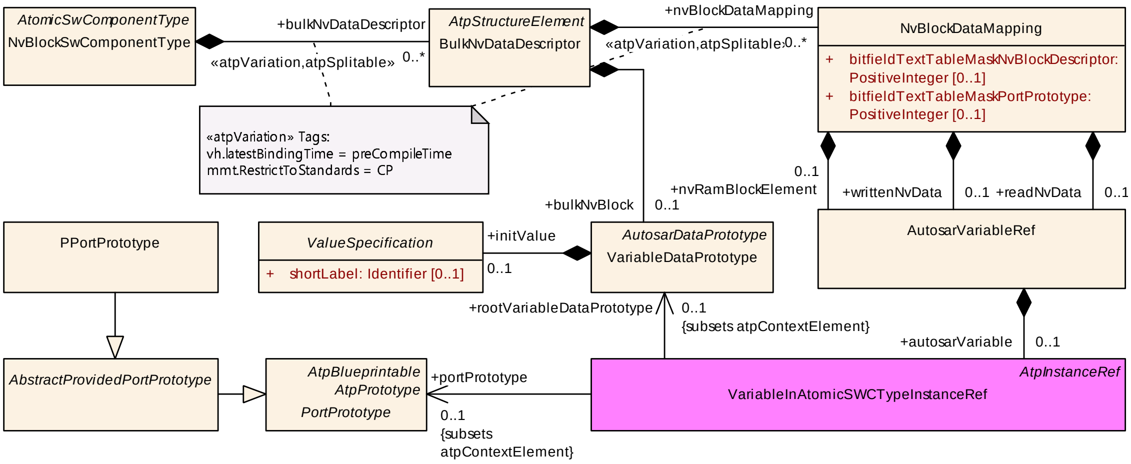


Figure 11.22: NvBlockSwComponentType and BulkNvDataDescriptor

[TPS_SWCT_01805] Semantics of the aggregation NvBlockSwComponentType.bulkNvDataDescriptor ⌈The existence of an aggregation at meta-class NvBlockSwComponentType in the role bulkNvDataDescriptor indicates that read-only access to a block of data in non-volatile memory shall be provided.

Access to this block of data is **provided directly from non-volatile memory** as opposed to via a RAM mirror.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01806] Simultaneous aggregation of NvBlockSwComponentType.bulkNvDataDescriptor and NvBlockSwComponentType.nvBlockDescriptor ⌈The simultaneous existence of aggregations in the roles bulkNvDataDescriptor and nvBlockDescriptor in the context of the same NvBlockSwComponentType is positively supported.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01807] Application of NvBlockDataMapping on BulkNvDataDescriptor ⌈Via the aggregation of NvBlockDataMapping in the role nvBlockDataMapping it is possible to make the content of the BulkNvDataDescriptor available to a PPortPrototype on the surface of the enclosing NvBlockSwComponentType.

It is also supported to make only a subset of the content of the BulkNvDataDescriptor available at the PPortPrototype.⌋*(RS_SWCT_03225)*

[constr_1735] Limitation of the aggregation of AutosarVariableRef in the context of an NvBlockDataMapping owned by a BulkNvDataDescriptor ⌈Any NvBlockDataMapping owned by a BulkNvDataDescriptor shall only aggregate an AutosarVariableRef in the role readNvData and nvRamBlockElement (that in turn refers to the BulkNvDataDescriptor.bulkNvBlock) **at the time when the RTE is generated.**⌋()

Of course, [constr_1285] also applies for the usage of BulkNvDataDescriptor.

Class	BulkNvDataDescriptor			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	This meta-class represents one bulk NV Data Block that is read-only for the application software. The purpose of a bulk NV Data Block is to provide access to information uploaded to the vehicle at e.g. the end of the production line.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, NvBlockSwComponentType.bulkNvDataDescriptor			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bulkNvBlock	VariableDataPrototype	0..1	aggr	This aggregation represents the actual bulk NVBlock.
nvBlockData Mapping	NvBlockDataMapping	*	aggr	Defines the mapping between the VariableData Prototypes in the NvBlockComponents ports and the VariableDataPrototypes of the non-volatile memory. The aggregation of NvBlockDataMapping is subject to variability with the purpose to support the conditional existence of nv data ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=nvBlockDataMapping, nvBlockData Mapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 11.13: BulkNvDataDescriptor

11.4.7 SwcInternalBehavior of an NvBlockSwComponentType

[TPS_SWCT_01150] InternalBehavior of a NvBlockSwComponentType to enable access to the NVRAM Block management API ⌈In general, the InternalBehavior of a NvBlockSwComponentType is only used for a limited scope.

The main use case is that the NvBlockSwComponentType defines PPortPrototypes typed by a ClientServerInterface to enable access to the NVRAM Block management API.

To enable the configuration of the server invocation in the RTE's ECU configuration, the NvBlockSwComponentType needs to provide the following model elements:

- OperationInvokedEvents
- server RunnableEntity
- PortDefinedArgumentValues to define the NVRAM Block ID which has to be passed to the NvM

In addition to the above list further model elements may qualify; the details are explained in [TPS_SWCT_01584].⌋()

[TPS_SWCT_01152] InternalBehavior does not have further attributes ⌈It is not expected, that such InternalBehavior do have further attributes like ExclusiveAreas, per-instance memory or inter-runnable variables, etc.⌋*()*

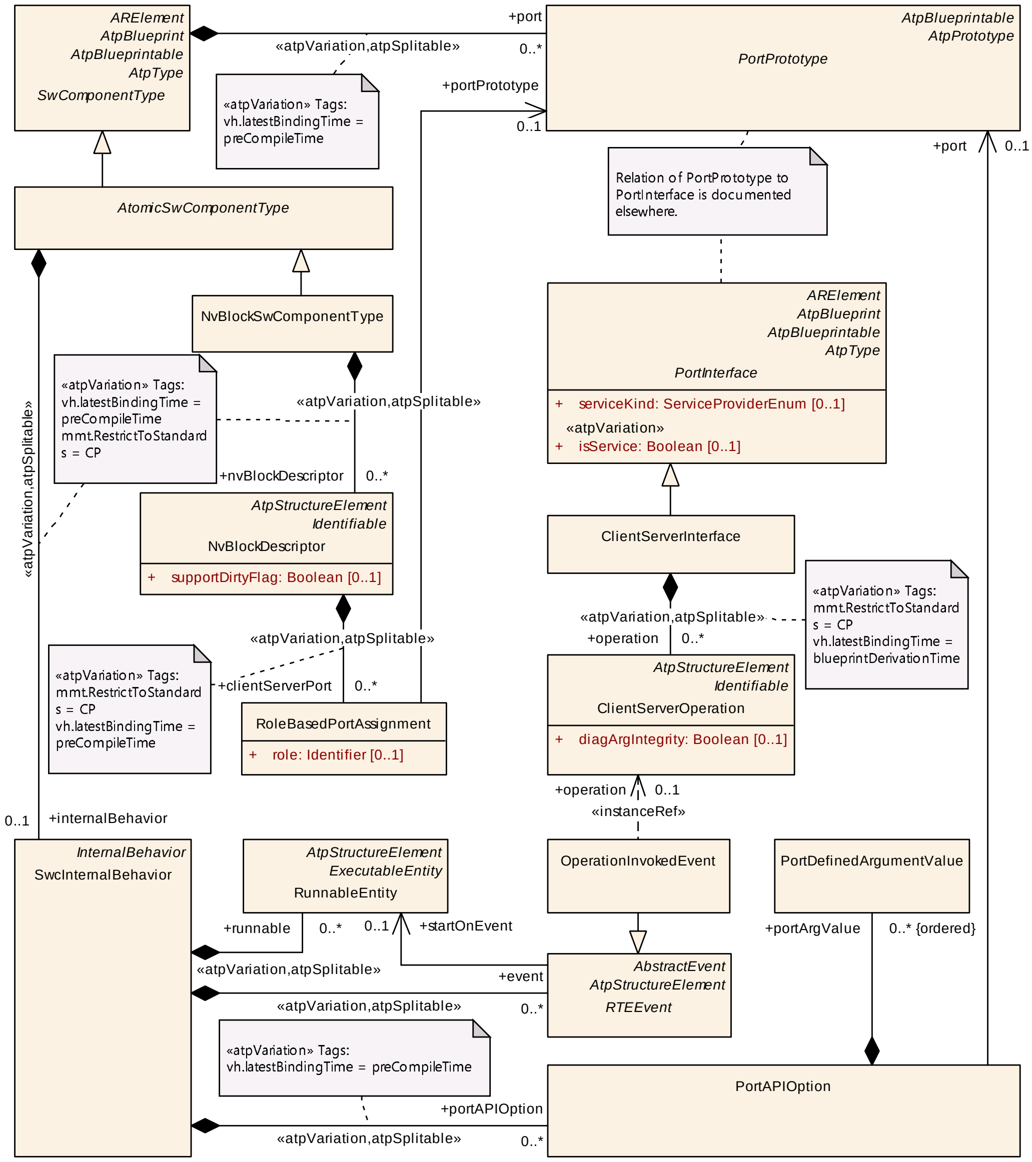


Figure 11.23: NvBlockSwComponentType and SwcInternalBehavior

[TPS_SWCT_01584] InternalBehavior of a NvBlockSwComponentType for implementing a writing strategy ⌈For the use case that NvBlockDescriptors exists that aggregate NvBlockNeeds which, in turn, define particular NV data writing strategies (by defining any of the attributes storeAtShutdown, storeImmediate, storeEmergency, or storeCyclic) the InternalBehavior of a NvBlockSwComponentType needs to support further model elements.

Particularly, In addition to the model elements listed in [TPS_SWCT_01150], the following list of model elements can be used in the InternalBehavior of a NvBlockSwComponentType for implementing writing strategies:

- TimingEvents (which may include references to ModeDeclarations in the role disabledMode)
- DataReceivedEvents (which may include references to ModeDeclarations in the role disabledMode)
- SwcModeSwitchEvents
- RunnableEntitys

⌋*(RS_SWCT_03225)*

[TPS_SWCT_01151] RunnableEntitys do not have further attributes ⌈The same condition exists for the RunnableEntitys of such InternalBehavior which shall not define further attributes, e.g. data access points (implemented by means of references from SwcInternalBehavior to VariableAccess) or ServerCallPoints.⌋*()*

[constr_1234] Value of RunnableEntity.symbol ⌈The value of a RunnableEntity.symbol owned by an NvBlockSwComponentType that is triggered by an OperationInvokedEvent shall only be taken from the set of API names associated with the NvM.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

For example, RunnableEntity.symbol owned by an NvBlockSwComponentType could rightfully be set to NvM_ReadBlock [37] but an arbitrary value like ReadThisBlock is not permitted.

The rationale for [constr_1234] is that the RunnableEntitys that are triggered by an OperationInvokedEvent are not existing as such but are mapped to the respective function calls of the NvM. For more details of how this mapping can be achieved please refer to [6].

Please note that no restriction applies for the value of attribute RunnableEntity.symbol of any RunnableEntity owned by an NvBlockSwComponentType that is triggered by an RTEEvent other than OperationInvokedEvent.

[constr_2015] Limitation of SwcInternalBehavior of a NvBlockSwComponentType ⌈The SwcInternalBehavior of a NvBlockSwComponentType is only permitted to define

- OperationInvokedEvents
- RunnableEntitys triggered by OperationInvokedEvents (server RunnableEntitys)
- RunnableEntitys which defines only the mandatory attributes symbol and canBeInvokedConcurrently
- PortAPIOptions defining PortDefinedArgumentValues

- TimingEvents (which may include references to ModeDeclarations in the role disabledMode)
- DataReceivedEvents (which may include references to ModeDeclarations in the role disabledMode)
- SwcModeSwitchEvents
- RunnableEntitys triggered by TimingEvents
- RunnableEntitys triggered by DataReceivedEvents
- RunnableEntitys triggered by SwcModeSwitchEvents
- DataTypeMappingSet

This rule shall be imposed **at the time when the RTE is generated**.⌋()

[constr_1309] Existence of NvBlockDescriptor.timingEvent ⌈The attribute NvBlockDescriptor.timingEvent shall exist **at the time when the RTE is generated** if and only if the NvBlockDescriptor.nvBlockNeeds.storeCyclic exists and is set to the value true.⌋()

Note that there is a conceptual connection between the values of the two attributes NvBlockDescriptor.timingEvent.period and SwcServiceDependency.serviceNeeds.cyclicWritingPeriod.

Specifically, the SwcServiceDependency.serviceNeeds.cyclicWritingPeriod represents a requirement and the NvBlockDescriptor.timingEvent.period is supposed to fulfill the requirement.

[TPS_SWCT_01585] Relevance of NvBlockDescriptor.timingEvent.period ⌈For any given NvBlockDescriptor, the value of the attribute NvBlockDescriptor.nvBlockNeeds.cyclicWritingPeriod shall be ignored and the value of NvBlockDescriptor.timingEvent.period shall be taken to specify the effective writing frequency for cyclic storage.⌋*(RS_SWCT_03225)*

[TPS_SWCT_01662] Applicability of DataTypeMappingSets inside an NvBlockSwComponentType ⌈The DataTypeMappingSets to be applied for a given NvBlockDescriptor is the superset of NvBlockDescriptor.dataTypeMapping and InternalBehavior.dataTypeMapping.⌋*(RS_SWCT_03225)*

12 Software Component Documentation

AUTOSAR supports documentation of software component types by adopting the principles of ASAM-FSX [38] Standard to AUTOSAR.

With AUTOSAR Release 4.0, the AUTOSAR XML schema provides support for integrated and well-structured documentation.

More details about the AUTOSAR Documentation Support Concept can be found in the AUTOSAR Generic Structure Template [11].

[TPS_SWCT_01062] Documentation of software-components ⌈The documentation of a SwComponentType is composed of several Chapters.

Some Chapters are predefined, describing the component from the perspective of different activities performed on the component, like:

- testing it (swTestDesc)
- maintaining it (swMaintenanceNotes)
- calibrating it (swCalibrationNotes)
- performing diagnostic (swDiagnosticsNotes)

⌋*(RS_SWCT_02110, RS_SWCT_03230)*

The documentation of a SwComponentType is shown in figure 12.1.

Two other predefined Chapters describe the component (swFeatureDesc) and define its physical functionality (swFeatureDef).

In order to describe additional aspects of a software component, an arbitrary number of free Chapters can be defined.

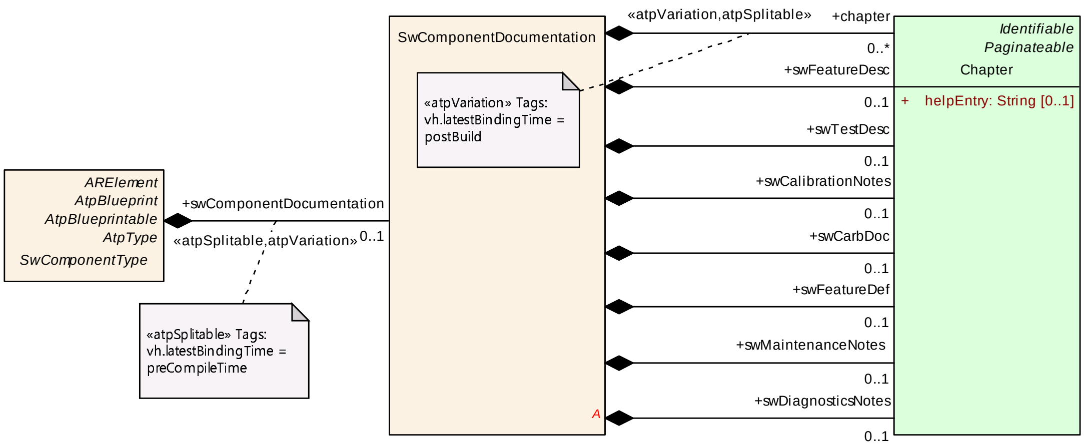


Figure 12.1: Software component documentation

The predefined Chapters typically provide informal guideline (e.g. recommendation) or documentation.

Formal information can be captured using special data groups [11] or annotating documentation construct with semantic information. This could be used to extend the predefined Chapters or in separate free Chapters.

Note that the documentation of a software component can be stored in a different file than the component itself (i.e., it is ≪atpSplitable≫ from the component).

Each of the predefined and free Chapters follows the ≪atpVariation≫ stereotype to support variant handling (see [11]) on the documentation at the Chapter level.

These VariationPoints set the latest binding time to the value AdditionalBindingTimeEnum.postBuild because the decision to include or exclude a Chapter as well as the decision which variant of this Chapter should be included can be made when the component has been built.

Class	**SwComponentDocumentation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SoftwareComponentDocumentation			
Note	This class specifies the ability to write dedicated documentation to a component type according to ASAM FSX.			
Base	*ARObject*			
Aggregated by	BswModuleDescription.bswModuleDocumentation, *SwComponentType*.swComponentDocumentation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
chapter	Chapter	*	aggr	These chapters provide additional information about the software component that do not fit in the other chapters. Note that this is subject to variation because Chapter aggregations in the role chapter are variant within the documentation in general. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=chapter.shortName, chapter.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=100 xml.typeElement=false
swCalibrationNotes	Chapter	0..1	aggr	This element contains calibration instructions and hints for a calibration engineer. **Tags:** xml.roleElement=true xml.sequenceOffset=60 xml.typeElement=false
swCarbDoc	Chapter	0..1	aggr	This element records the documentation requested by CARB. **Tags:** xml.roleElement=true xml.sequenceOffset=80 xml.typeElement=false
swDiagnosticsNotes	Chapter	0..1	aggr	This element contains general information about diagnostics issues within the component. **Tags:** xml.roleElement=true xml.sequenceOffset=75 xml.typeElement=false

▽

△

Class	**SwComponentDocumentation**			
swFeatureDef	Chapter	0..1	aggr	This element contains the definition of the physical functionality of this software component. This definition is more or less formal and is intended to be delivered from modeling tools. **Tags:** xml.roleElement=true xml.sequenceOffset=20 xml.typeElement=false
swFeatureDesc	Chapter	0..1	aggr	This element contains the textual description of the software functionality of this software component. Expert should write this description. **Tags:** xml.roleElement=true xml.sequenceOffset=30 xml.typeElement=false
swMaintenance Notes	Chapter	0..1	aggr	This element contains information regarding the software maintenance of the component. **Tags:** xml.roleElement=true xml.sequenceOffset=70 xml.typeElement=false
swTestDesc	Chapter	0..1	aggr	This element contains suggestions and hints for the test of the software functionality of this software component. **Tags:** xml.roleElement=true xml.sequenceOffset=50 xml.typeElement=false

Table 12.1: SwComponentDocumentation

Class	**Chapter**			
Package	M2::MSR::Documentation::Chapters			
Note	This meta-class represents a chapter of a document. Chapters are the primary structuring element in documentation.			
Base	*ARObject*, *DocumentViewSelectable*, *Identifiable*, *MultilanguageReferrable*, *Paginateable*, *Referrable*			
Aggregated by	ChapterOrMsrQuery.chapter, MsrQueryResultChapter.chapter, SwComponentDocumentation.chapter, SwComponentDocumentation.swCalibrationNotes, SwComponentDocumentation.swCarbDoc, SwComponentDocumentation.swDiagnosticsNotes, SwComponentDocumentation.swFeatureDef, SwComponentDocumentation.swFeatureDesc, SwComponentDocumentation.swMaintenanceNotes, SwComponentDocumentation.swTestDesc, System.systemDocumentation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
chapterModel	ChapterModel	1	aggr	This represents the overall contents of the chapter. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.typeElement=false xml.typeWrapperElement=false

▽

△

Class	Chapter			
helpEntry	String	0..1	attr	This specifies an entry point in an online help system to be linked with the parent class. The syntax shall be defined by the applied help system respectively help system generator. Maybe it is a concatenated Identifier, but as of now we leave it as an arbitrary string. **Tags:**xml.attribute=true

Table 12.2: Chapter

Enumeration	**AdditionalBindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumeration specifies the additional binding times applicable for vh.latestBindingTime of variation points.
Literal	*Description*
blueprintDerivation Time	The point in time when an object is created from a blueprint. **Tags:**atp.EnumerationLiteralIndex=0
postBuild	After the executable has been built. **Tags:**atp.EnumerationLiteralIndex=1

Table 12.3: AdditionalBindingTimeEnum

13 Service Dependencies and Service Use Cases

13.1 Overview

Meta-class SwcServiceDependency represents a powerful concept to describe the service-related capabilities of an AtomicSwComponentType.

It is still required to understand how to configure SwcServiceDependency and related meta-classes for specific service use cases.

This chapter contains a detailed description of the meta-classes related to SwcServiceDependency in the context of specific service use cases, as well as modeling hints for the configuration of the respective service use cases.

13.2 NvM Service Dependencies

The meta-class NvBlockNeeds is used to define requirements to configure the NVRAM Manager Service. In addition, it may define requirements how the RTE shall implement writing strategies of an NvBlockSwComponentType.

An SwcInternalBehavior may provide several SwcServiceDependencys that in turn aggregate an NvBlockNeeds element where each defines the requirements from one NVRAM Block (for more information on the AUTOSAR NVRAM Manager see [37]).

A diagram that explains the modeling of ServiceNeeds for the NVRAM Manager Service can be found in Figure 11.15.

There are several use cases how a software-component can interact with the NVRAM Manager service. Each use case is discussed in a separate sub-chapter.

Enumeration	**RamBlockStatusControlEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent
Note	This enumeration type defines options for how the management of the ramBlock status is controlled.
Aggregated by	NvBlockNeeds.ramBlockStatusControl
Literal	**Description**
api	The ramBlock status is controlled via service interface by usage of the SetRamBlockStatus operation. **Tags:**atp.EnumerationLiteralIndex=0
nvRamManager	The ramBlock status is controlled exclusively by the Nv Ram Manager. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.1: RamBlockStatusControlEnum

13.2.1 Nvm Use Case: Permanent RAM Block

Scenario: a AtomicSwComponentType is using an NVRAM Block with a Permanent RAM Block implemented by a PerInstanceMemory section or a VariableDataPrototype in the role arTypedPerInstanceMemory. In either case, the required memory for the Permanent RAM Block is allocated by the RTE during ECU Configuration.

In this case the following rules apply:

[TPS_SWCT_02501] Setup for Nvm Use Case: Permanent RAM Block ⌈

RoleBasedPortAssignment
For every used ClientServerInterface provided by the NvM it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used standardized ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvmService [0 .. 1]
- NvMNotifyJobFinished [0 .. 1]
- NvMNotifyInitBlock [0 .. 1]
- NvMAdmin [0 .. 1]

RoleBasedDataAssignment
RoleBasedDataAssignment shall be created that refers to either the PerInstanceMemory in the role usedPim or to the VariableDataPrototype in the role usedDataElement. The value of the attribute role of the RoleBasedDataAssignment shall be set to ramBlock.

Optionally, it is possible to create an additional RoleBasedDataAssignment to a ParameterDataPrototype in the role usedParameterElement. The value of the ParameterDataPrototype is then taken as the initial or default value for the NVRAM Block. In this case the value of the attribute role of the RoleBasedDataAssignment shall be set to defaultValue.
Therefore, the following roles are applicable:

- ramBlock [1]
- defaultValue [0 .. 1]

RepresentedPortGroup
N/A

⌋*(RS_SWCT_03225)*

For more information please refer to [SWS_NvM_00734], [SWS_NvM_00735], [SWS_NvM_00736], and [SWS_NvM_00737].

The same mechanism (see description of scenario) applies also for an NvBlockSwComponentType. For each NVRAM Block the NVRAM Manager can be configured (with the help of SwcServiceDependency.assignedData) to use the same Permanent RAM Block.

It is the responsibility of the NVRAM Manager to provide the content of the NVRAM Block in this Permanent RAM Block during startup or on explicit request and to write back the content to the storage medium during shut-down or on explicit request.

13.2.2 Nvm Use Case: Temporary RAM Block

Scenario: an AtomicSwComponentType is using some NVRAM Block with a Temporary RAM Block.

In this case the AtomicSwComponentType is responsible for allocating the allocation of sufficient memory. In other words, the AtomicSwComponentType shall provide a memory area that is available to the API call to the NVRAM Manager for storage of the NV data.

[TPS_SWCT_02502] Setup for Nvm Use Case: Temporary RAM Block ⌈

RoleBasedPortAssignment

This is mandatory for the described scenario. For every used ClientServerInterface provided by the Nvm it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvmService [1]
- NvMNotifyJobFinished [0 .. 1]
- NvMNotifyInitBlock [0 .. 1]
- NvMAdmin [0 .. 1]

RoleBasedDataAssignment

The usage of a RoleBasedDataAssignment with attribute role set to defaultValue is optional and depends on whether an initial value is required.

- defaultValue [0..1]

RoleBasedDataTypeAssignment

By this means it is possible to define the data-type of a Temporary RAM Block. The data type information can be used to calculate the NVRAM Block size. [constr_1301] applies.

- temporaryRamBlock [0..1]

RepresentedPortGroup
N/A

⌋*(RS_SWCT_03225)*

[constr_1301] Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role ⌈The usage of a RoleBasedDataTypeAssignment with attribute role set to the value temporaryRamBlock is only allowed if **no** RoleBasedDataAssignment defined with attribute role set to value defaultValue exists in the owning SwcServiceDependency.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

The rationale for [constr_1301] is that the existence of a RoleBasedDataAssignment would already provide sufficient information for the intended purpose. The parallel existence of a RoleBasedDataTypeAssignment is therefore fully redundant and could only lead to potential inconsistencies.

For more information please refer to [SWS_NvM_00734], [SWS_NvM_00735], [SWS_NvM_00736], and [SWS_NvM_00737].

13.2.3 Nvm Use Case: RAM Block with explicit synchronization using Mirror Interfaces

Scenario: an AtomicSwComponentType is using an NVRAM Block where the RAM Block uses explicit synchronization by means of mirror interfaces. In this case the RAM Block does not necessarily have to be formally described by means of a PerInstanceMemory or a VariableDataPrototype in the role arTypedPerInstanceMemory.

Consequently, the software-component itself is responsible for the allocation of memory. On the other hand, this can also mean that the software-component can use several RAM Blocks instead of just one RAM Block.

[TPS_SWCT_02504] Setup for Nvm Use Case: RAM Block with explicit synchronization using Mirror Interfaces ⌈

RoleBasedPortAssignment
This is mandatory for the described scenario. For every used ClientServerInterface provided by the Nvm it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvMService [0..1]
- NvMNotifyJobFinished [0..1]
- NvMNotifyInitBlock [0..1]

- NvMAdmin [0..1]
- NvMMirror [1]

RoleBasedDataAssignment

In this scenario the existence of a RoleBasedDataAssignment is optional. The RoleBasedDataAssignment needs to reference a ParameterDataPrototype aggregated by the enclosing SwcInternalBehavior in the role perInstanceParameter or sharedParameter.

- defaultValue [0..1]

RoleBasedDataTypeAssignment

By this means it is possible to define the data-type of a temporary RAM Block and used internal data structure in case of explicit synchronization with NvMMirror interface respectively. The data type information can be used to calculate the NVRAM Block size and minimum Permanent RAM Block size. [constr_1301] applies.

- temporaryRamBlock [0..1]

RepresentedPortGroup

N/A

⌋*(RS_SWCT_03225)*

For more information please refer to [SWS_NvM_00734], [SWS_NvM_00735], [SWS_-NvM_00736], [SWS_NvM_00737], and [SWS_NvM_00738].

13.2.4 NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType (not ServiceSwComponent of NvM)

Scenario: an AtomicSwComponentType is using an NVRAM Block managed by an NvBlockSwComponentType (see section 11.4.2, as opposed to an NVRAM Block provided by a ServiceSwComponentType). Constraints [constr_1148], [constr_1149], and [constr_2011] apply.

[TPS_SWCT_02503] Setup for NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType ⌈

RoleBasedPortAssignment

This is mandatory for the described scenario. For every used ClientServerInterface provided by the NvM it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvMService [0..1]

- NvMNotifyJobFinished [0..1]
- NvMNotifyInitBlock [0..1]
- NvMAdmin [0..1]

For every PortPrototype of a software-component typed by an NvDataInterface defining a SwcServiceDependency it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the attribute assignedPort to the value NvDataPort:

- NvDataPort [1..*]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

For supporting this use case the value of attribute SwcServiceDependency.category shall be set to **NV_BLOCK_COMPONENT**.⌋*(RS_SWCT_03225)*

For more information please refer to [SWS_NvM_00734], [SWS_NvM_00735], [SWS_NvM_00736], and [SWS_NvM_00737]. Note that NvBlockNeeds (described in Chapter 11.4.5) is not in the scope of this use case.

13.3 Watchdog Service Dependencies

The meta-class SupervisedEntityNeeds is used to define requirements to configure the Watchdog Service. For the terms related to the AUTOSAR Watchdog Manager see [39].

13.3.1 Watchdog Service use Case: Local Supervision

The service interaction with the *Watchdog Manager* consists of two aspects:

- supervised entity
- checkpoint

For each of the two aspects a separated ServiceNeeds is defined. However, the SwcServiceDependencys that own these ServiceNeeds are semantically bound and cannot be used independently of each other.

In other words, the usage of two kinds of SwcServiceDependency in concert creates a higher-level semantics. Of course, in order to express this higher-level semantics a reference between the SwcServiceDependencys has to be available.

However, since the SwcServiceDependency represents a generic concept the actual reference needs to be implemented on the level of specific subclass of ServiceNeeds, in this case the SupervisedEntityNeeds and the SupervisedEntityCheckpointNeeds.

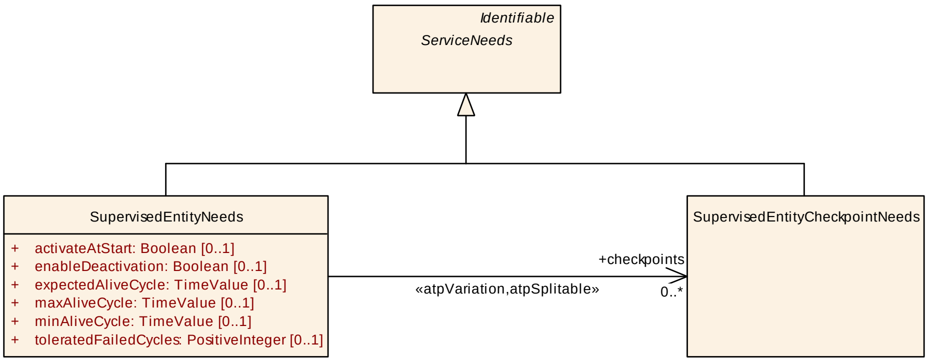


Figure 13.1: Modeling of ServiceNeeds for the watchdog

The former refers to the latter in order to express the relation of a supervised entity to its checkpoints.

Class	**SupervisedEntityNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager for one specific Supervised Entity.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
activateAtStart	Boolean	0..1	attr	true/false: supervision activation status of Supervised Entity shall be enabled/disabled at start.
checkpoints	SupervisedEntity CheckpointNeeds	*	ref	This reference indicates the checkpoints belonging to the Supervised Entity. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=checkpoints.supervisedEntityCheckpoint Needs, checkpoints.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
enable Deactivation	Boolean	0..1	attr	true: software-component shall be allowed to deactivate supervision of this SupervisedEntity false: software-component shall be not allowed to deactivate supervision of this SupervisedEntity
expectedAlive Cycle	TimeValue	0..1	attr	Expected cycle time of alive trigger of this Supervised Entity (in seconds).
maxAliveCycle	TimeValue	0..1	attr	Maximum cycle time of alive trigger of this Supervised Entity (in seconds).
minAliveCycle	TimeValue	0..1	attr	Minimum cycle time of alive trigger of this Supervised Entity (in seconds).

▽

△

Class	SupervisedEntityNeeds			
toleratedFailedCycles	PositiveInteger	0..1	attr	Number of consecutive failed alive cycles for this SupervisedEntity which shall be tolerated until the supervision status of the SupervisedEntity is set to WDGM_ALIVE_EXPIRED (see SWS WdgM for more details). Note that this value has to be recalculated with respect to the WdgM's own cycle time for ECU configuration.

Table 13.2: SupervisedEntityNeeds

[constr_1985] Existence of the reference `SupervisedEntityNeeds.toleratedFailedCycles` ⌈For each `SupervisedEntityNeeds`, the reference to `BswInternalBehavior` in the role `toleratedFailedCycles` shall exist **`at the time when the RTE is generated`**.⌋ *()*

Class	SupervisedEntityCheckpointNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager to support a Checkpoint for a Supervised Entity.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.3: SupervisedEntityCheckpointNeeds

13.3.2 Watchdog Service use Case: *Global Supervision Status* notification

Scenario: an `AtomicSwComponentType` requires to receive the *Global Supervision Status* that is combined from all individual *Supervised Entities*. In this case the following setup applies:

[TPS_SWCT_02019] Setup for `AtomicSwComponentType` which requires *Global Supervision Status* notification ⌈

RoleBasedPortAssignment valid roles:

- `WdgM_GlobalMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋ *(RS_SWCT_00030)*

For more information please refer to [SWS_WdgM_00336].

13.3.3 Watchdog Service use Case: Control global supervision or get global supervision status

Scenario: an AtomicSwComponentType either controls the global operation of the watchdog manager or gets information about the current operation status, requiring at least one of the following use cases:

- Set the current mode of Watchdog Manager
- Get the current mode of the Watchdog Manager
- Get the global supervision status of the Watchdog Manager
- Identifier of the supervised entity that first reached the expired state
- Instruct the Watchdog Manager to cause a watchdog reset

For instance, the software-component sets the current mode of the Watchdog Manager according to the operational state of the ECU or polls the global supervision status.

In this case the following setup applies:

[TPS_SWCT_01703] Setup for AtomicSwComponentType which sets or gets the Global Supervision Status ⌈

ServiceNeeds kind GlobalSupervisionNeeds

RoleBasedPortAssignment valid roles:

- WdgM_GlobalSupervision [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Class	**GlobalSupervisionNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager to get access on the Global Supervision control and status interface.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.4: GlobalSupervisionNeeds

13.3.4 Watchdog Service use Case: Software-component wants to obtain the status of a local supervision

Scenario: a software-component wants to obtain the status of a local supervision. For this purpose, it exposes an RPortPrototype typed by ClientServerInterface WdgM_LocalSupervisionStatus towards the WdgM.

[TPS_SWCT_01704] Definition of supervised entity ⌈

ServiceNeeds kind : SupervisedEntityNeeds

RoleBasedPortAssignment valid roles:

- WdgM_LocalSupervisionStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that an SwcInternalBehavior may provide several SupervisedEntityNeeds elements where each defines the requirements in relation to one supervised entity.

Note that in this situation an AtomicSwComponentType contains several *Checkpoints* that refer to a *Supervised Entity*.

In this case it is required that the *Supervised Entity* indicates to the *Watchdog Manager* the existence this *Checkpoint* for configuration and at runtime that the *Supervised Entity* has reached the *Checkpoint*.

For more information please refer to [SWS_WdgM_91004]

13.3.5 Watchdog Service use Case: Software-component wants to report a checkpoint

Scenario: A software-component want to report a checkpoint to the WdgM. For this purpose the software-component exposes an RPortPrototype to the WdgM.

[TPS_SWCT_01705] Definition of Checkpoints ⌈

ServiceNeeds kind : SupervisedEntityCheckpointNeeds

RoleBasedPortAssignment valid roles:

- WdgM_LocalSupervision [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

Please note that this scenario covers the reporting of a single checkpoint. In many cases, several checkpoints are reported in the context of a given supervised entity.

For more information please refer to [SWS_WdgM_00333].

13.4 COM Manager Service Needs

The meta-class `ComMgrUserNeeds` is used to define requirements to configure the `ComM` Service.

An `SwcInternalBehavior` may provide several `ComMgrUserNeeds` elements where each defines the requirements from one "user" of the ComM Service. Especially, it defines which `PortGroup` is associated with this "user".

Class	**ComMgrUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Communication Manager for one "user".			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maxCommMode	MaxCommModeEnum	0..1	attr	Maximum communication mode requested by this ComM user.

Table 13.5: ComMgrUserNeeds

13.4.1 ComM Use Case: read current ComM Mode

Scenario: a `AtomicSwComponentType` reads the current ComM mode.

In this case the following rules apply:

[TPS_SWCT_01019] `AtomicSwComponentType` reads the current ComM mode ⌈

RoleBasedPortAssignment valid roles:

- `ComM_CurrentMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋(*RS_SWCT_03200*)

For more information please refer to [SWS_ComM_00847].

13.4.2 ComM Use Case: request ComM Mode

Scenario: a AtomicSwComponentType requests a ComM mode. It may also check later whether the requested ComM mode has become effective.

In this case the following rules apply:

[TPS_SWCT_01020] AtomicSwComponentType requests a ComM mode. It may also check later whether the requested ComM mode has become effective ⌈

RoleBasedPortAssignment valid roles:

- ComM_CurrentMode [1]
- ComM_UserRequest [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the applicable PortGroup [0..1]

⌋*(RS_SWCT_03200)*

For more information please refer to [SWS_ComM_00848].

13.4.3 ComM Use Case: Software-Component acts as a Mode Manager that influences the ECU State

Scenario: a AtomicSwComponentType acts as a mode manager that influences the ECU state.

In this case the following rules apply:

[TPS_SWCT_01021] AtomicSwComponentType acts as a mode manager that influences the ECU state ⌈

RoleBasedPortAssignment valid roles:

- ComM_CurrentMode [0..1]
- ComM_UserRequest [0..1]
- ComM_ECUModeLimitation [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋(*RS_SWCT_03200*)

For more information please refer to [SWS_ComM_00741].

13.4.4 Read PNC ComM Mode

Scenario: an AtomicSwComponentType reads the current PNC ComM mode.

[TPS_SWCT_01811] AtomicSwComponentType reads the current PNC ComM mode ⌈

ServiceNeeds kind : ComMgrUserNeeds

RoleBasedPortAssignment valid roles:

- ComM_UserRequest [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the applicable PortGroup associated with the particular partial network.

⌋(*RS_SWCT_03200*)

For clarification, the difference between the two service use case descriptions [TPS_SWCT_01019] and [TPS_SWCT_01811] is that the latter reads the ComM mode in the context of a PNC definition while the former does not consider partial networking.

13.5 ECU State Manager Service Needs

The meta-class EcuStateMgrUserNeeds is used to define the requirements to configure the ECU State Manager Service.

An SwcInternalBehavior may provide several EcuStateMgrUserNeeds elements where each defines the requirements from one "user" of the EcuM Service (for the terms related to the AUTOSAR ECU State Manager see [40]).

Class	EcuStateMgrUserNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the ECU State Manager for one "user". This class currently contains no attributes. Its name can be regarded as a symbol identifying the user from the viewpoint of the component or module which owns this class.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.6: EcuStateMgrUserNeeds

13.5.1 EcuM Use Case: read current ECU Mode

Scenario: a `AtomicSwComponentType` reads the current ECU mode.

In this case the following rules apply:

[TPS_SWCT_01856] `AtomicSwComponentType` reads the current ECU mode ⌈

RoleBasedPortAssignment valid roles:

- `EcuM_CurrentMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*()*

13.5.2 EcuM Use Case: request a certain ECU state

Scenario: a `AtomicSwComponentType` needs to keep the ECU alive or needs to execute operations before the ECU is shut down. For this purpose the `AtomicSwComponentType` may request either the state `RUN` or `POST_RUN`.

In this case the following rules apply:

[TPS_SWCT_01857] `AtomicSwComponentType` shall keep the ECU alive ⌈

`AtomicSwComponentType` needs to keep the ECU alive or needs to execute operations before the ECU is shut down.

RoleBasedPortAssignment valid roles:

- `EcuM_StateRequest` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋()

13.5.3 EcuM Use Case: select Shutdown Target

Scenario: a AtomicSwComponentType wants to select a shutdown target.

In this case the following rules apply:

[TPS_SWCT_01016] AtomicSwComponentType wants to select a shutdown target ⌈

RoleBasedPortAssignment valid roles:

- EcuM_ShutdownTarget [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(RS_SWCT_03200)*

13.5.4 EcuM Use Case: select Boot Target

Scenario: a AtomicSwComponentType wants to select a boot target.

In this case the following rules apply:

[TPS_SWCT_01017] AtomicSwComponentType wants to select a boot target ⌈

RoleBasedPortAssignment valid roles:

- EcuM_BootTarget [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(RS_SWCT_03200)*

13.5.5 EcuM Use Case: use Alarm Clock

Scenario: a AtomicSwComponentType wants to use an alarm clock.

In this case the following rules apply:

[TPS_SWCT_01018] AtomicSwComponentType wants to use an alarm clock ⌈

RoleBasedPortAssignment valid roles:

- EcuM_AlarmClock [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(RS_SWCT_03200)*

13.6 BswM

All use cases for interaction of an application software-component with the BswM require the aggregation in the role serviceNeeds of BswMgrNeeds, a subclass of ServiceNeeds, at SwcServiceDependency.

Class	BswMgrNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Basic Software Manager for one "user".			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.7: BswMgrNeeds

13.6.1 Partial Networking

One specific use case for the existence of a SwcServiceDependency with respect to the interaction with the BswM is the support for partial networking, in particular the association of a PortGroup and the associated PortPrototypes that act as *VFC control ports* and *VFC status ports*.

For more details please refer to section 4.8.

In this case the following rules apply:

[TPS_SWCT_01126] Access to partial networking via BswM ⌈

RoleBasedPortAssignment valid roles:

- control [0 .. 1]
- status [0 .. 1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the applicable PortGroup associated with the particular partial network.

⌋*(RS_SWCT_03200, RS_SWCT_03201)*

The multiplicities of the RoleBasedPortAssignments for this case have been defined under the assumption that a given software-component may or may not have a *VFC control port*. Also, it may have a *VFC status port*. Technically, there could be several *VFC status ports* per software-component but most likely there is only one *VFC status port*.

13.6.2 Mode Manager

A software-component that acts as a mode manager exposes a PPortPrototype typed by a ModeSwitchInterface. By this means the mode manager communicates changes of the particular mode to the connected mode users.

On the side of the BswM, an RPortPrototype typed by an ModeSwitchInterface used to receive notifications of mode switches will have to be established (for more details, please refer to [SWS_BswM_00200]).

In this case the following rules apply:

[TPS_SWCT_01552] Software-component acts as a mode manager ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment valid roles:

- AppModeInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
none

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03203)*

A slight variation of this use case exists if the Application Mode Manager serves mode users that consist of both the BswM and other ApplicationSwComponentTypes.

[TPS_SWCT_01572] Application Mode Manager interacts with both BswM and other ApplicationSwComponentTypes ⌈If an Application Mode Manager interacts with both BswM and other ApplicationSwComponentTypes the following requirements on the modeling of this scenario shall be taken into account:

Mode Request For the configuration of mode requests two separate AbstractRequiredPortPrototypes shall exist:

- One AbstractRequiredPortPrototype shall be typed by a SenderReceiverInterface with attribute isService set to true. This AbstractRequiredPortPrototype shall be connected to the SwComponentPrototype typed by a ServiceSwComponentType representing the BswM.
- One AbstractRequiredPortPrototype shall be typed by a SenderReceiverInterface with attribute isService set to false. This AbstractRequiredPortPrototype shall be connected to SwComponentPrototypes typed by ApplicationSwComponentTypes that request mode changes.

Mode Switch Notification An Application Mode Manager that sends mode switch notifications to both BswM and other ApplicationSwComponentTypes shall expose a single AbstractProvidedPortPrototype for sending the mode switch notification to both the BswM and ApplicationSwComponentTypes.

The value of the attribute ModeSwitchInterface.isService shall be set to false).

⌋*(RS_SWCT_03200, RS_SWCT_03202)*

Rationale for [TPS_SWCT_01572]: technically, the existence of two separate AbstractProvidedPortPrototype for sending the mode switch notification to both the BswM and ApplicationSwComponentTypes would end up in two separate mode machines in the RTE and it would be a tough challenge to keep both mode machines perfectly synchronized.

Therefore, the exception regarding the usage of the attribute isService is justified to mitigate this effect.

On the mode request side, however, the situation is entirely different because the mode requests need arbitration by the Application Mode Manager anyway. This is completely in the scope of the implementation of the Application Mode Manager and AUTOSAR has no stakes in further standardizing this aspect.

Therefore, there is no motivation for a further exception with respect to the value of isService.

[TPS_SWCT_01664] BswM acts as a mode requester towards an application mode manager ⌈The SwcServiceDependency that covers this use case shall refer to an RPortPrototype for the reception of the mode request and optionally to a PPortPrototype for the sending of the mode switch notification.

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment valid roles:

- AppModeInterface [0..1]

RoleBasedDataAssignment valid roles:

- AppModeRequestInterface [1]

RepresentedPortGroup
none

⌋(*RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03203*)

[constr_1680] Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = AppModeRequestInterface ⌈If the attribute RoleBasedDataAssignment.role is set to the value AppModeRequestInterface, then the reference RoleBasedDataAssignment.usedDataElement.localVariable shall not exist **at the time when the RTE is generated**.⌋()

For explanation of the existence of [constr_1680], it is not intended to provide access to local variables inside the SwcInternalBehavior.

13.6.3 Mode User

A software-component that acts as a mode user exposes an RPortPrototype typed by a ModeSwitchInterface. By this means the software-component can be notified by mode switches executed at the mode manager (in this case the BswM).

On the side of the BswM, a PPortPrototype typed by an ModeSwitchInterface used to send out notifications of mode switches will have to be established (for more details, please refer to [SWS_BswM_00202]).

In this case the following rules apply:

[TPS_SWCT_01553] Software-component acts as a mode user ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment valid roles:

- AppModeInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
none

⌋(*RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03203*)

13.6.4 Mode Requester

A software-component that acts as a mode requester exposes an PPortPrototype typed by a SenderReceiverInterface. By this means the software-component can send mode requests towards the mode manager (in this case the BswM).

On the side of the BswM, an RPortPrototype typed by an SenderReceiverInterface used to requests for mode switches will have to be established (for more details, please refer to [SWS_BswM_00201]).

In this case the following rules apply:

[TPS_SWCT_01554] Software-component acts as a mode requester ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- AppModeRequestInterface [1]

RepresentedPortGroup
none

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03202)*

Please note that [constr_1680] applies for this service use case.

13.6.5 Service Oriented Use Cases

Support for service oriented communication is represented by a set of service use cases utilizing SwcServiceDependency in the interaction with the BswM. Specifically the interaction of application software-components with ServiceDiscovery (via the BswM) is handled in this section.

The AUTOSAR ServiceDiscovery can be configured to run autonomously (without the need of application software-components to initiate the ServiceDiscovery) during the startup of the ECU. In such cases there is no need to define any service oriented use cases at the respective application software-components.

This approach has the advantage that the ServiceDiscovery is performed without knowledge of the application software-components. But it has the disadvantage that all the ServiceDiscovery is performed at startup, even if some application software-components might not require that service functionality right after startup.

Providing the application software-components with the possibility to individually control the ServiceDiscovery only when the functionality actually needs that information may allow a fine grained control of communication amount.

Whether a specific service instance is controlled autonomously or whether there is a dedicated application interaction to control the availability or subscription of a service can be defined individually per service instance in the TPS System Template [10] (ProvidedServiceInstance.autoAvailable, ConsumedServiceInstance.autoRequire, and ConsumedEventGroup.autoRequire).

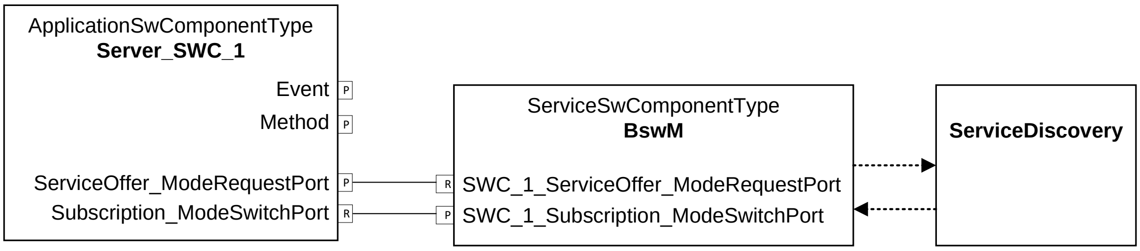


Figure 13.2: General interaction of Application - BswM - ServiceDiscovery

13.6.5.1 PortGroup supports Relation between Software-Component and service Instance

A service can be composed of an arbitrary number of

- SenderReceiverInterfaces representing events, field notifiers, and fire-and-forget methods
- ClientServerInterfaces representing methods, field getters, and field setters
- TriggerInterfaces representing fire-and-forget methods without arguments.

The TPS System Template [10] provides means to define which SenderReceiverInterfaces and ClientServerInterfaces eventually build up the service in section titled "Description of Service Discovery Services in Classic Platform".

This definition applies to the PortInterface level, thus it is not focusing on individual PortPrototypes.

If an application software-component would like to consume two instances of the same service, it would not be possible to identify which set of PortPrototypes belongs to one service instance and which set of PortPrototypes belongs to the other service instance.

The relation between PortPrototypes and service instances is defined using a PortGroup. In the example depicted in Figure 13.3, a server software-component provides two PortPrototypes, an event and a method.

A PortGroup is used to group these two PortPrototypes. Service use cases described later in this section utilize the SwcServiceDependency.representedPortGroup reference to define for which PortGroup (service instance) a dedicated service use case applies.

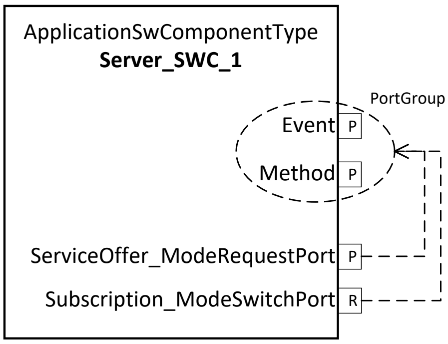


Figure 13.3: Usage of PortGroup to define which PortPrototypes belong to a service instance

The TPS System Template [10] also provides means for defining event groups (ProvidedServiceInstance.eventHandler and ConsumedServiceInstance.consumedEventGroup).

Those event groups are not directly related to the PortGroups discussed in this section. They may be defined in a consistent way, but it is also possible that the event groups are defined in a different way that the PortGroups of a specific service instance.

It is the duty of the BswM configuration to define one or more adequate BswMRules to associate the event groups to the respective events collected in the PortGroups.

13.6.5.2 General Server interaction

The server has 2 use cases defined for the interaction with service discovery:

- service offer (section 13.6.5.5)
- subscription status for events (section 13.6.5.6).

The service offer makes this service instance available in the system.

As soon as the service offer has been made, every client can directly call the methods offered by this service instance. There is no way to place a dedicated subscription for calling the methods (including fire-and-forget methods).

The server may be interested in the information about the the subscription of events (or groups of events) by any client. The granularity of the status information can be defined by the server application software-component.

The servers depicted in the example in Figure 13.4 illustrate some aspects of the server service discovery approach:

The server *Server_SWC_1* is simple and just provides two events and one method. One PPortPrototype has been defined to offer this service and another RPortPrototype is used to receive information about the subscription status .

The server *Server_SWC_2* has some more features, it also provides two events and one method, and additionally receives a “call” of a fire-and-forget method. One PPortPrototype offers this service and two RPortPrototypes are used to receive information about the subscription status.

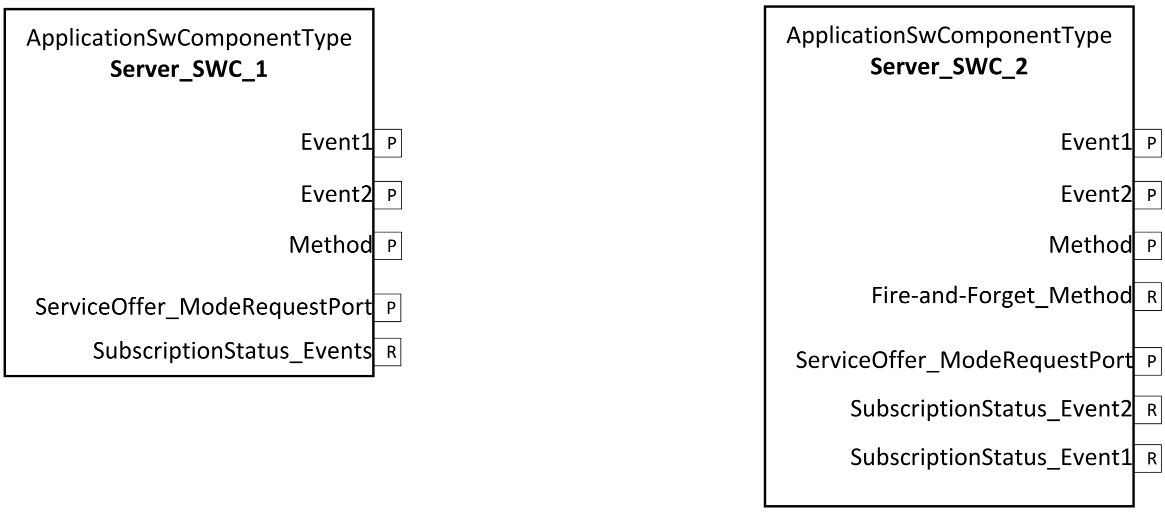


Figure 13.4: Server examples

13.6.5.3 General Client interaction

The client has several use cases defined for the interaction with service discovery:

- event (and method) subscription (section 13.6.5.7)
- event (and method) subscription status (section 13.6.5.8)

The service discovery for clients in AUTOSAR is specified to consist of two steps from the clients perspective:

- *finding* a service instance and
- *subscribing* to an event (group).

On the level of application software-components and service definitions, the event groups are not visible. Event groups get introduced in the system description (see TPS System Template [10]) in the context of definition of the communication matrix.

The event subscription in the service use cases are defined on the level of events. It is the duty of the BswM configuration to define one or more adequate BswMRules to derive the corresponding event groups out of the individual service instance events for subscriptions in the service discovery.

The application software-components on the *AUTOSAR classic Platform* do not have any means to influence the choice on the offered service instances during the service discovery *find*. This is the reason why there is no explicit *find* service use case defined in this section. The *find* is included in the respective subscription use cases.

From the client's perspective, there is no difference between subscribing to an event or "subscribing" to a method. The use case defined in section 13.6.5.7 does not distinguish whether there are just

- events,
- events and methods,
- or just methods

involved. It is again the BswM which is configured to represent the corresponding actions for the service discovery.

The example clients in figure 13.5 illustrate some aspects of the client service discovery approach:

The exemplary client *Client_SWC_1* is simple and just consumes two events and one method. One PPortPrototype is defined to subscribe for all of the events and methods of this service.

One RPortPrototype is used to inform the client whether the subscription to the whole service was successful. This means: if there is a service instance *found* (this enables the usage of the method) **and** the *subscription* to **both** events is successful, then the RPortPrototype indicates that the subscription is successful.

This also means, that if the *subscription* to at least one of the events is not successful, the indication for the whole service is not successful.

The client *Client_SWC_2* has some more features: it requires two events and one method, and additionally sends a fire-and-forget method. One PPortPrototype is defined to subscribe to this service's events and a second PPortPrototype exists to "subscribe" to the service's methods. Two RPortPrototypes are defined to receive the subscription status of the events and methods.

For the PPortPrototype, to "subscribe" to this service's methods means that (if not already performed by the other subscription to this service) a *find* is issued to the service discovery. If the *find* is successful, the notification on the *subscription* status at the RPortPrototype for the methods is performed with a successful indication. Thus the application software-component can start calling the methods of that service.

For the PPortPrototype, to subscribe to this service's events means that (if not already performed by the other subscription to this service) a *find* is issued to the service discovery.

If the *find* is successful, the *subscription* to the events is issued to the service discovery. If then the *subscription* to **both** events is successful, the notification on the *subscription* status at the RPortPrototype for the events is performed with a successful indication.

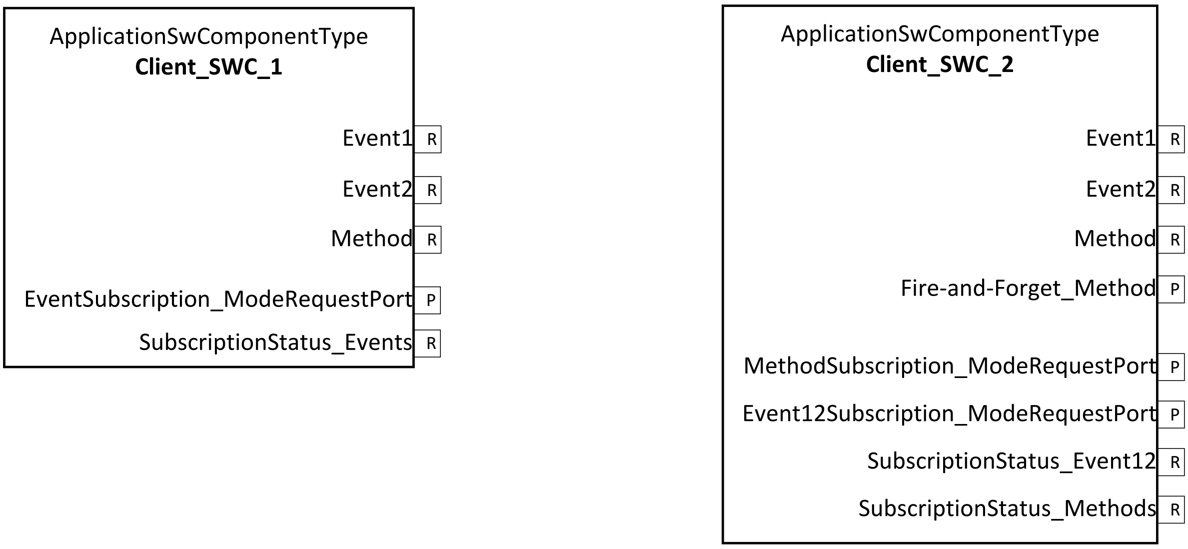


Figure 13.5: Client examples

Another example of the client interaction is given in figure 13.8.

13.6.5.4 Event Reception Without Subscription

A side effect, which needs to be taken into account by the application software-components, arises from the separate handling of service discovery mode requests and the event communication.

As soon as an event is subscribed by one application software-component inside the ECU, this event is forwarded to all receivers on that ECU, regardless whether they actually have subscribed to that event or not.

The communication path is not restricted by the service discovery or the BswM. Thus, there is no way to prevent the reception of events that have not been subscribed to by each individual receiving application software-component.

One cause for the unintended reception of events is sketched in figure 13.6. Two independent application software-components are interested in the same event of the same service instance.

Each application software-component has an RPortPrototype to actually receive the event. Each of the application software-components has a PPortPrototype to initiate the subscription to that event and an RPortPrototype to get notification on the event subscription status.

When one of the application software-component starts the subscription of the event (e.g. *Client_SWC_1*), then the BswM will forward the subscription request to the service discovery.

When the service discovery eventually receives the subscription acknowledgement the BswM gets notified and forwards that subscription status notification to **all** application software-component which have a corresponding event subscription status notification RPortPrototype defined.

Thus, *Client_SWC_1* gets a notification on the successful subscription of the event. But **also** *Client_SWC_2* get a notification on the successful subscription of that event, although *Client_SWC_2* didn't subscribe to that event!

Now, when the event messages actually arrive at that ECU, the events are indicated to be received to **all** the application software-components which have an RPortPrototype to receive the event. Thus *Client_SWC_1* will receive that event but also *Client_SWC_2* will receive the event.

One possible solution to that issue is to guard any subscription status notification and event reception with the application software-component own event subscription request: Only if the application software-component actually did subscribe to that event, then incoming subscription status notifications and event receptions will be considered by the application software-component.

Another approach might be to use RTEEvent.disabledMode configuration to disable the notification in a specific mode.

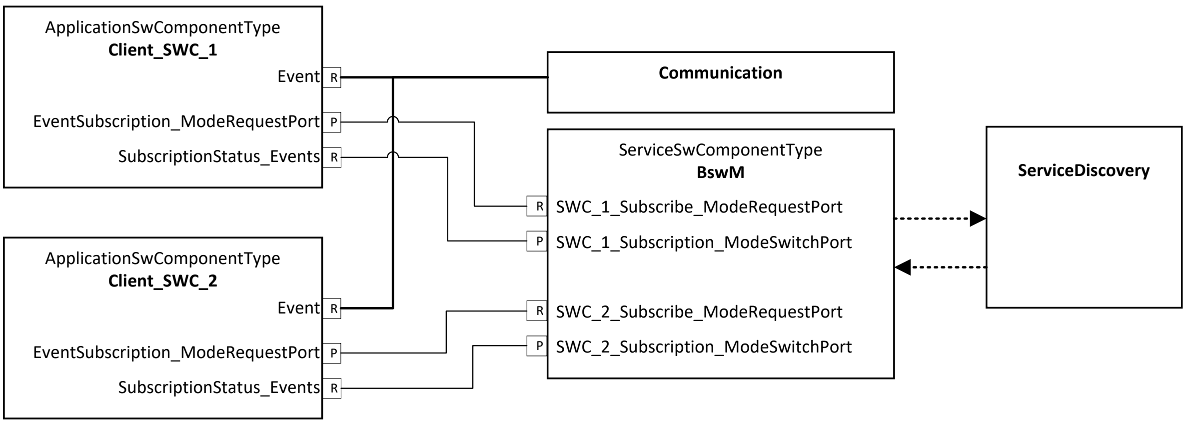


Figure 13.6: Unintended event reception

Another source of unintended event reception may be the definition of the communication matrix and the assignments of events to event groups.

If two events on the client application software-component are handled via two separate PPortPrototypes for the subscription of these events, but the events are defined to be part of the same event group in the communication matrix, then both events will be received by the application software-component, even it asked only for the subscription of one event.

13.6.5.5 Use Case: Server Service Offer

A software-component that implements the server functionality acts as a mode requester and exposes a PPortPrototype typed by a SenderReceiverInterface. By this means, the software-component can send mode requests towards the BswM in order to *offer* and *stop offer* that service instance.

On the side of the BswM, a RPortPrototype typed by the same SenderReceiverInterface is used to requests for mode switches.

In this case the following rule applies:

[TPS_SWCT_03502] Software-component acts as a server and offers the service ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- ServerServiceOffer [1]

RepresentedPortGroup
Reference to the service instance PortGroup representing this server's PortPrototypes [1]

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03202)*

[constr_3688] Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = ServerServiceOffer ⌈If the attribute RoleBasedDataAssignment.role is set to the value ServerServiceOffer, then the reference RoleBasedDataAssignment.usedDataElement.localVariable shall not exist **at the time when the RTE is generated**.⌋*()*

For explanation of the existence of [constr_3688], it is not intended to provide access to local variables inside the SwcInternalBehavior.

13.6.5.6 Use Case: Server Service Subscription Status

A server instance sending events or field notifications might not need to produce and send those events if there are no subscriptions for that server instance events.

In order to notify the server instance whether there is at least one subscriber interested in the server's events a ModeSwitchInterface is available to receive the subscription notifications.

A software-component that acts as a server and is interested in the subscription status for the service instance events exposes an RPortPrototype typed by a ModeSwitchInterface. By this means, the software-component can be notified by the BswM on subscription changes.

On the side of the BswM, a PPortPrototype typed by the ModeSwitchInterface is used to send out notifications.

In this case, the following rules apply:

[TPS_SWCT_03503] Software-component receives notification on server event subscription status changes ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment valid roles:

- ServerEventSubscriptionStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the service instance PortGroup representing this server's PortPrototypes (or the subset relevant for this notification) [1]

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03203)*

The granularity of the referenced representedPortGroup PortGroup may vary depending on the desired notification granularity.

One approach can be to use one PortGroup for the entire service instance. This may be the same PortGroup which is also used for the service offer in [TPS_SWCT_03502].

In this case any included PortPrototypes typed by a ClientServerInterface or TriggerInterface are ignored, as they do not participate in the subscription of events. This approach is sketched in the example *Service_SWC_1* in figure 13.7.

Another approach can be to group one or some PortPrototypes in separate PortGroups and define a dedicated SwcServiceDependency for each of these PortGroups.

This approach is illustrated in the example *Service_SWC_2* in Figure 13.7, where the *Service_SWC_2* has two RPortPrototypes to receive individual notification for *PortGroup1* and *PortGroup2*.

It is the duty of the BswM configuration to define one or more adequate BswMRules, which take the subscription status from the ServiceDiscovery and combine them to get the status which represents the subscription status of the events included in the specific PortGroup.

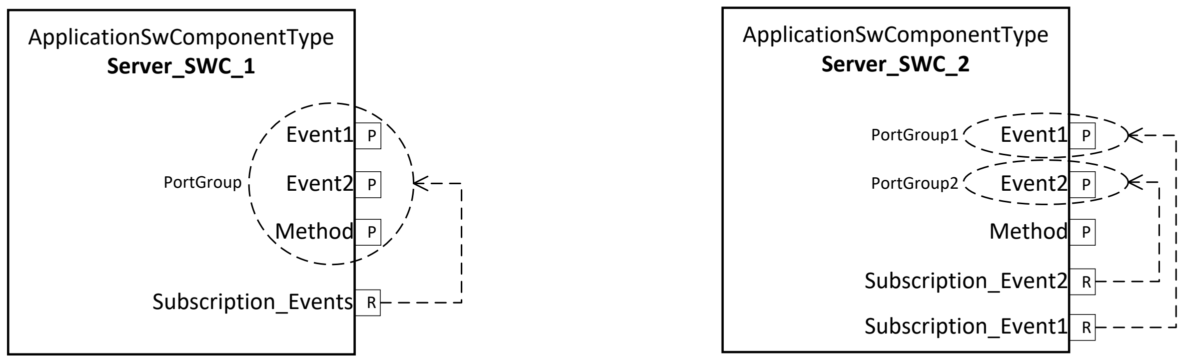


Figure 13.7: Service subscription notification examples

The server event subscription status use case ([TPS_SWCT_03503]) is not limited to servers where the offer is controlled by the application software-component. It is also possible to get subscription status information on a server's events where the server is autonomously offered.

13.6.5.7 Use Case: Client Event / Method Subscription

A software-component that implements the client functionality acts as a mode requester and exposes a PPortPrototype typed by a SenderReceiverInterface.

By this means, the software-component can send mode requests towards the BswM in order to *subscribe* and *stop subscribe* of dedicated events and methods.

On the side of the BswM, an RPortPrototype typed by the same SenderReceiverInterface is used to requests for mode switches.

In this case the following rule applies:

[TPS_SWCT_03504] Software-component acts as a client and subscribe to events and methods ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- ClientEventSubscription [1]

RepresentedPortGroup
Reference to the service instance PortGroup representing this client's PortPrototypes (or the subset relevant for this notification) [1]

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03202)*

[constr_3689] Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = ClientEventSubscription ⌈If the attribute RoleBasedDataAssignment.role is set to the value ClientEventSubscription, then the reference RoleBasedDataAssignment.usedDataElement.localVariable shall not exist **at the time when the RTE is generated**.⌋*()*

For explanation of the existence of [constr_3689], it is not intended to provide access to local variables inside the SwcInternalBehavior.

13.6.5.8 Use Case: Client Event and Method Subscription Status

A software-component that acts as a client and is interested in the subscription status for the events and methods exposes an RPortPrototype typed by a ModeSwitchInterface. By this means the software-component can be notified by the BswM whether subscription was successful.

On the side of the BswM, a PPortPrototype typed by the ModeSwitchInterface is used to send out notifications.

In this case the following rules apply:

[TPS_SWCT_03505] Software-component receives notification on client event and method subscription status changes ⌈

ServiceNeeds kind BswMgrNeeds

RoleBasedPortAssignment valid roles:

- ClientEventSubscriptionStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the service instance PortGroup representing this client's PortPrototypes (or the subset relevant for this notification) [1]

⌋*(RS_SWCT_03110, RS_SWCT_03200, RS_SWCT_03203)*

Generally, it is up to the definition of the PortGroups which PortPrototypes are included in the event status notification use case. A different definition of PortGroups for the subscription status notification from the definition of PortGroups for the subscription itself is possible.

Figure 13.8 shows two setups for the event subscription status notification of the same client service instance which consists of two events and one method.

In the illustrative *Client_SWC_1*, the *SubscriptionStatus_EventMethod* uses a PortGroup to include both events and the method in one status notification.

This setup might be a good approach for the client subscription (see section 13.6.5.7), as it requests the two events and the method to become available in one go.

But for the status notification this results in a combined notification status: Only when both events **and** the method are subscribed the notification will result in a successful subscription.

The method could be already called when the first part of the service discovery (the service instance *find*) was successful. If this information is required by the application software-component, an alternative definition of subscription notifications and PortGroups could be more suitable:

In the illustrative *Client_SWC_2*, two notifications have been defined: one *SubscriptionStatus_Events* and one *SubscriptionStatus_Method*.

While the notification for *SubscriptionStatus_Events* still has to wait until there is a successful subscription acknowledgement for both events available, the notification on the availability of the method can already be given when the service instance *find* was successful. Thus the method can be called already even though the events are not yet available.

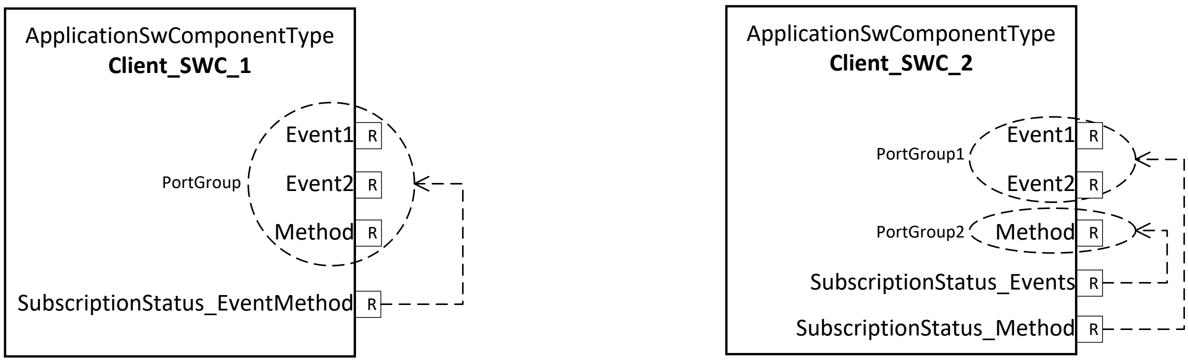


Figure 13.8: Event and Method subscription notification examples

13.7 Crypto Service Dependencies

13.7.1 Overview

The meta-classes CryptoServiceNeeds and CryptoServiceJobNeeds are used to define requirements for the configuration of the CryptoServiceManager respectively the crypto stack.

The usage of meta-class CryptoKeyManagementNeeds is described in section 13.7.4.

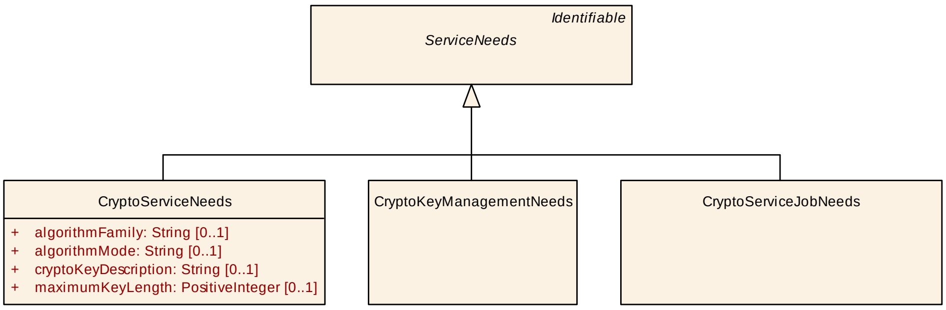


Figure 13.9: Modeling of subclasses of `ServiceNeeds` for cryptographic use cases

Please note that there are cryptographic APIs that build upon the creation of jobs that run asynchronously. The reason for this policy is that cryptographic operations - in many cases by design - tend to run for comparatively long time for each call. This behavior is visualized in Figure 13.10.

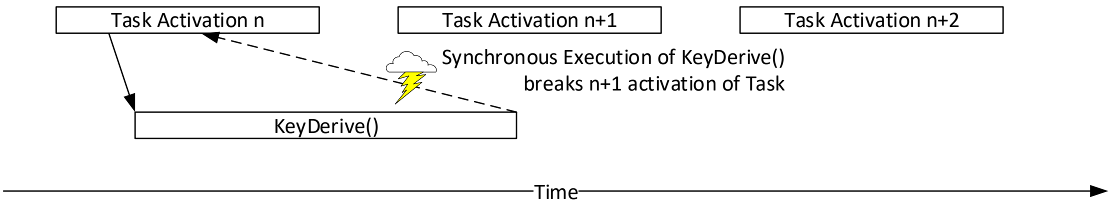


Figure 13.10: Cryptographic operation that requires a too-long execution time

Execution of these operations synchronously in the main function would block the respective module intolerably and therefore the job API is an important measure to keep the execution of software manageable. This behavior is visualized in Figure 13.11.

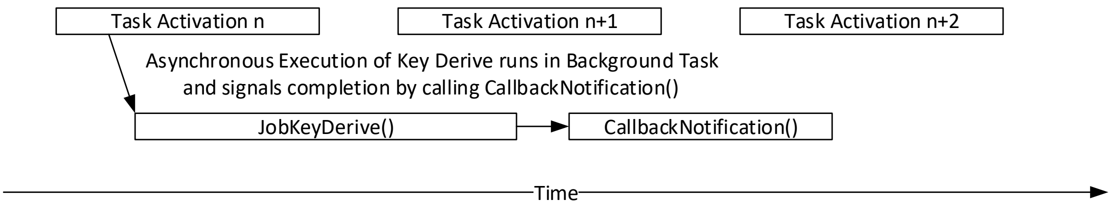


Figure 13.11: Execution of cryptographic operation using the job API

It is important to note that the asynchronous character of the execution is implemented on the server side and has nothing to do with asynchronous calling behavior on the client side (for more explanation about client-side calling behavior, please refer to section 7.5.2.1).

Class	**CryptoServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the needs on the configuration of the CryptoServiceManager for one ConfigID (see Specification AUTOSAR_SWS_CSM.doc). An instance of this class is used to find out which ports of a software-component belong to this ConfigID.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
algorithmFamily	String	0..1	attr	This attribute represents a description of the family (e.g. AES) of crypto algorithm implemented by the crypto service use case.
algorithmMode	String	0..1	attr	This meta-class has the ability to represent a crypto service use case.
cryptoKey Description	String	0..1	attr	This attribute allows for a verbal description of the applicable cryptographic key. The goal is to pass a hint for the integrator about how to treat the corresponding service use case.
maximumKey Length	PositiveInteger	0..1	attr	The maximum length of a cryptographic key, that is used by the software-component or module for this configuration. Unit: bit.

Table 13.8: CryptoServiceNeeds

Class	**CryptoServiceJobNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class shall be taken to indicate that the service use case modeled with this kind of Service Needs assumes the usage of the crypto job API.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.9: CryptoServiceJobNeeds

[TPS_SWCT_01727] Suffix used for the resulting name of the `PortInterface` for crypto `PortInterfaces` ⌈The _*{Config}* or (where applicable)_*{Primitive}* suffix used for the resulting name of the `PortInterface` for the respective crypto service shall be taken from the `shortName` of the applicable `SwcServiceDependency`.⌋*()*

13.7.2 Crypto Service Use Cases

13.7.2.1 Crypto Service Use Case: Hash calculation

Scenario: a `AtomicSwComponentType` uses the hash calculation of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02020] `AtomicSwComponentType` uses the hash calculation of the Crypto Service ⌈

ServiceNeeds kind : `CryptoServiceNeeds`

RoleBasedPortAssignment valid roles:

- CsmHash [0..1]
- CsmHash_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmHash_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.2 Crypto Service Use Case: MAC calculation

Scenario: a AtomicSwComponentType uses the message authentication code (MAC) calculation of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02021] AtomicSwComponentType uses the message authentication code (MAC) calculation of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmMacGenerate [0..1]
- CsmMacGenerate_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmMacGenerate_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.3 Crypto Service Use Case: MAC verification

Scenario: a AtomicSwComponentType uses the message authentication code (MAC) verification of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02022] AtomicSwComponentType uses the message authentication code (MAC) verification of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmMacVerify [0..1]
- CsmMacVerify_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmMacVerify_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.4 Crypto Service Use Case: generation of random numbers

Scenario: a AtomicSwComponentType uses the generation of random numbers of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02024] AtomicSwComponentType uses the generation of random numbers of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmRandomGenerate [0..1]
- CsmRandomGenerate_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(RS_SWCT_00030)

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmRandomGenerate_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.5 Crypto Service Use Case: Encryption with Authenticated Encryption with Associated Data (AEAD)

Scenario: a AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02025] AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmAEADEncrypt [0..1]
- CsmAEADEncrypt_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(RS_SWCT_00030)

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmAEADEncrypt_{Config} shall be resolved according to [TPS_SWCT_01727].

13.7.2.6 Crypto Service Use Case: Decryption with Authenticated Encryption with Associated Data (AEAD)

Scenario: a AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02026] AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmAEADDecrypt [0..1]

- CsmAEADDecrypt_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmAEADDecrypt_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.7 Crypto Service Use Case: encryption

Scenario: a AtomicSwComponentType uses the encryption of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02027] AtomicSwComponentType uses the encryption of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmEncrypt [0..1]
- CsmEncrypt_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmEncrypt_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.8 Crypto Service Use Case: decryption

Scenario: a AtomicSwComponentType uses the decryption of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02028] AtomicSwComponentType uses the decryption of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmDecrypt [0..1]
- CsmDecrypt_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmDecrypt_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.9 Crypto Service Use Case: signature generation

Scenario: a AtomicSwComponentType uses the signature generation of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02031] AtomicSwComponentType uses the signature generation of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmSignatureGenerate [0..1]
- CsmSignatureGenerate_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmSignatureGenerate_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.10 Crypto Service Use Case: signature verification

Scenario: a AtomicSwComponentType uses the signature verification of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_02032] AtomicSwComponentType uses the signature verification of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmSignatureVerify [0..1]
- CsmSignatureVerify_{Config} [0..1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmSignatureVerify_{Config} is regulated by [TPS_SWCT_01727].

13.7.2.11 Crypto Service Use Case: usage of key management

Scenario: a AtomicSwComponentType uses the key management of the Crypto Service. In this case the following setup applies:

[TPS_SWCT_01726] AtomicSwComponentType uses the key management of the Crypto Service ⌈

ServiceNeeds kind : CryptoServiceNeeds

RoleBasedPortAssignment valid roles:

- CsmKeyManagement_{Config} [1]
- CallbackNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Config} that appears in the shortName of the ClientServerInterface CsmKeyManagement_{Config} is regulated by [TPS_SWCT_01727].

13.7.3 Crypto Service Job Use Cases

13.7.3.1 Crypto Service Use Case: usage of job API to set key valid

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to set a key valid. In this case the following setup applies:

[TPS_SWCT_01776] AtomicSwComponentType uses the API of the Crypto Service to set a key valid ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobKeySetValid_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobKeySet-Valid_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.2 Crypto Service Use Case: usage of job API to create a random seed

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to create a random seed. In this case the following setup applies:

[TPS_SWCT_01777] AtomicSwComponentType uses the API of the Crypto Service to create a random seed ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobRandomSeed_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobRandomSeed_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.3 Crypto Service Use Case: usage of job API to generate a key

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to generate a key. In this case the following setup applies:

[TPS_SWCT_01778] AtomicSwComponentType uses the API of the Crypto Service to generate a key ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobKeyGenerate_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobKeyGenerate_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.4 Crypto Service Use Case: usage of job API to derive a key

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to derive a key. In this case the following setup applies:

[TPS_SWCT_01779] AtomicSwComponentType uses the API of the Crypto Service to derive a key ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobKeyDerive_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobKeyDerive_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.5 Crypto Service Use Case: usage of job API to execute calculation of the public value for key exchange

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to execute calculation of the public value for key exchange. In this case the following setup applies:

[TPS_SWCT_01780] AtomicSwComponentType uses the API of the Crypto Service to execute calculation of the public value for key exchange ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobKeyExchangeCalcPubVal_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobKeyExchangeCalcPubVal_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.6 Crypto Service Use Case: usage of job API to execute calculation of shared secret for key exchange

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to execute calculation of shared secret for key exchange. In this case the following setup applies:

[TPS_SWCT_01781] AtomicSwComponentType uses the API of the Crypto Service to execute calculation of shared secret for key exchange ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobKeyExchangeCalcSecret_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobKeyExchangeCalcSecret_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.7 Crypto Service Use Case: usage of job API to execute certificate parsing

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to execute certificate parsing. In this case the following setup applies:

[TPS_SWCT_01782] AtomicSwComponentType uses the API of the Crypto Service to execute certificate parsing ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobCertificateParse_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobCertificateParse_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.3.8 Crypto Service Use Case: usage of job API to execute certificate verification

Scenario: a AtomicSwComponentType uses the **job API** of the Crypto Service to execute certificate verification. In this case the following setup applies:

[TPS_SWCT_01783] AtomicSwComponentType uses the API of the Crypto Service to execute certificate verification ⌈

ServiceNeeds kind : CryptoServiceJobNeeds

RoleBasedPortAssignment valid roles:

- CsmJobCertificateVerify_{Primitive} [1]
- CallbackNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Please note that the resolution of the name fragment {Primitive} that appears in the shortName of the ClientServerInterface CsmJobCertificateVerify_{Primitive} is regulated by [TPS_SWCT_01727].

13.7.4 Crypto Key Management Use Cases

The service use cases for cryptographic key management are indicated by the presence of a CryptoKeyManagementNeeds aggregated at SwcServiceDependency in the role serviceNeeds.

Class	**CryptoKeyManagementNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class can be used to indicate a service use case for key management.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.10: CryptoKeyManagementNeeds

13.7.4.1 KeyM Service Use Case: Software-Component wants check a certificate on KeyM

Scenario: A software-component wants check a certificate on KeyM. The software-component sends the certificate to KeyM (SetCertificate) and then initiates a verification of the certificate (VerifyCertificate).

While the verification is still running it is possible to obtain the status of the verification (GetStatus).

[TPS_SWCT_01813] Software-Component wants check a certificate on KeyM ⌈

ServiceNeeds kind : CryptoKeyManagementNeeds

RoleBasedPortAssignment valid roles:

- KeyM_Certificate [1]
- KeyMCertificateNotification [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

13.7.4.2 KeyM Service Use Case: Software-Component wants to retrieve a certificate from KeyM

Scenario: A software-component wants to retrieve a certificate on KeyM. The software-component request the certificate by using GetCertificate. This step represents a synchronous operation.

[TPS_SWCT_01814] AtomicSwComponentType wants to retrieve a certificate from KeyM ⌈

ServiceNeeds kind : CryptoKeyManagementNeeds

RoleBasedPortAssignment valid roles:

- KeyM_Certificate [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

13.7.4.3 KeyM Service Use Case: Software-Component wants to retrieve elements of a certificate from KeyM

Scenario: A software-component wants to retrieve elements of a certificate on KeyM (CertificateElementGet, CertificateElementGetByIndex, CertificateElementGetCount). This step represents a synchronous operation.

[TPS_SWCT_01815] AtomicSwComponentType wants to retrieve elements of a certificate from KeyM ⌈

ServiceNeeds kind : CryptoKeyManagementNeeds

RoleBasedPortAssignment valid roles:

- KeyMCertificateElement [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

13.7.4.4 KeyM Service Use Case: Software-Component wants to check the existence of a certificate from KeyM

Scenario: A software-component wants to check the existence of a certificate from KeyM (GetStatus).

[TPS_SWCT_01816] AtomicSwComponentType wants to check the existence of a certificate from KeyM ⌈

ServiceNeeds kind : CryptoKeyManagementNeeds

RoleBasedPortAssignment valid roles:

- KeyM_Certificate [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

13.7.4.5 KeyM Service Use Case: Software-Component wants to store a (derived) key in KeyM

Scenario: A software-component wants to store a (derived) key in `KeyM`.

[TPS_SWCT_01817] `AtomicSwComponentType` wants to store a (derived) key in `KeyM` ⌈

ServiceNeeds kind : `CryptoKeyManagementNeeds`

RoleBasedPortAssignment valid roles:

- `KeyMCryptoKey` [1]
- `KeyMCryptoKeyNotification` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

13.7.4.6 KeyM Service Use Case: Software-Component wants to store a container with (encrypted) keys in KeyM

Scenario: A software-component wants to store a container with (encrypted) keys in `KeyM`. The processing of the container is done in the context of a session that enables the usage of the `Start` and `Finish` operations.

By calling the `Prepare` operation the software-component passes the container to the KeyM, which reads the content of the container and extracts the contained cryptographic keys. The next step is that the KeyM updates every key extracted from the container.

[TPS_SWCT_01818] `AtomicSwComponentType` wants to store a container with encrypted keys (e.g. She-keys) in `KeyM` ⌈

ServiceNeeds kind : `CryptoKeyManagementNeeds`

RoleBasedPortAssignment valid roles:

- `KeyMCryptoKey` [1]
- `KeyMCryptoKeyNotification` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

13.7.4.7 KeyM Service Use Case: Software-Component wants to verify if cryptographic operation was executed using a specific key

Scenario: A software-component wants to verify if cryptographic operation was executed using a specific key (`Verify`).

[TPS_SWCT_01819] AtomicSwComponentType wants to verify if cryptographic operation was executed using a specific key ⌈

ServiceNeeds kind : `CryptoKeyManagementNeeds`

RoleBasedPortAssignment valid roles:

- `KeyMCryptoKey` [1]
- `KeyMCryptoKeyNotification` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

13.8 Diagnostic Service Dependency

This chapter describes the usage of the specific diagnostic meta-classes derived from `ServiceNeeds` within an atomic software-component. An overview of common diagnostic service needs has already been introduced in figure 7.39 and can be divided into four main parts:

- Function Inhibition Needs in chapter 13.8.2
- Diagnostic Event Needs in chapter 13.8.3
- Diagnostic Communication Needs in chapter 13.8.4
- Service Needs to fulfill the OBD related requirements in chapter 13.8.5

Please note that for the described use cases of the Diagnostic Services the following rule applies:

[TPS_SWCT_01129] Express diagnostic capabilities ⌈For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`. Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`.

The possible role attribute values and the multiplicity of the related PortPrototypes are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.⌋ *(RS_SWCT_00170, RS_SWCT_03190)*

[constr_1640] No use of Optional Element Structure for interaction with the diagnostic stack ⌈An SwcServiceDependency that aggregates a diagnostic-related subclass of ServiceNeeds shall not refer to any PortPrototype by means of either a RoleBasedPortAssignment or RoleBasedDataAssignment where the respective PortInterface contains any DataPrototype typed by an Optional Element Structure.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

13.8.1 Development Approach

AUTOSAR supports in many cases the implementation of a bottom-up or top-down configuration for various aspects of the development workflow.

Before the advent of the Diagnostic Extract [23], AUTOSAR officially supported a modeling approach where the configuration of an ApplicationSwComponentType might contain contributions to the external configuration of the diagnostic stack. This was considered an approach to implement a top-down configuration approach for diagnostics.

However, the development approach introduced with the Diagnostic Extract turned out to be superior and less prone to mistakes and limitations in comparison to the configuration that each developer contributed without necessarily having the knowledge about the greater scope of diagnostic configuration.

Therefore, the usage of the Diagnostic Extract has become the canonical approach to the configuration of the external behavior of the AUTOSAR diagnostic stack and the respective configuration attributes available in the scope of SwcServiceDependency have been removed from the AUTOSAR methodology in order to reduce potential confusion in the audience.

In particular, a top-down approach using a Diagnostic Extract can be implemented if the Diagnostic Extract also provides the mappings between diagnostic elements and elements of the application model. If the mappings are not wanted or not available, it is also possible to use the Diagnostic Extract to derive a bottom-up configuration of the diagnostic stack without the relations to application software.

13.8.2 Function Inhibition Needs

The meta-class FunctionInhibitionNeeds is used to define requirements in order to configure the Diagnostic Event Manager Service.

An SwcInternalBehavior may provide FunctionInhibitionNeeds as well as FunctionInhibitionAvailabilityNeeds elements in the context of an SwcServiceDependency. Each FunctionInhibitionNeeds and FunctionInhibitionAvailabilityNeeds defines the requirements related to one function inhibition ID (for the terms related to the AUTOSAR Function Inhibition Manager, see [41]).

Class	**FunctionInhibitionNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Function Inhibition Manager for one Function Identifier (FID). This class currently contains no attributes. Its name can be regarded as a symbol identifying the FID from the viewpoint of the component or module which owns this class.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.11: FunctionInhibitionNeeds

Class	**FunctionInhibitionAvailabilityNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Function Inhibition Manager to provide the control function for one Function Identifier (FID).			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
controlledFid	FunctionInhibitionNeeds	0..1	ref	This reference represents the controlled FID

Table 13.12: FunctionInhibitionAvailabilityNeeds

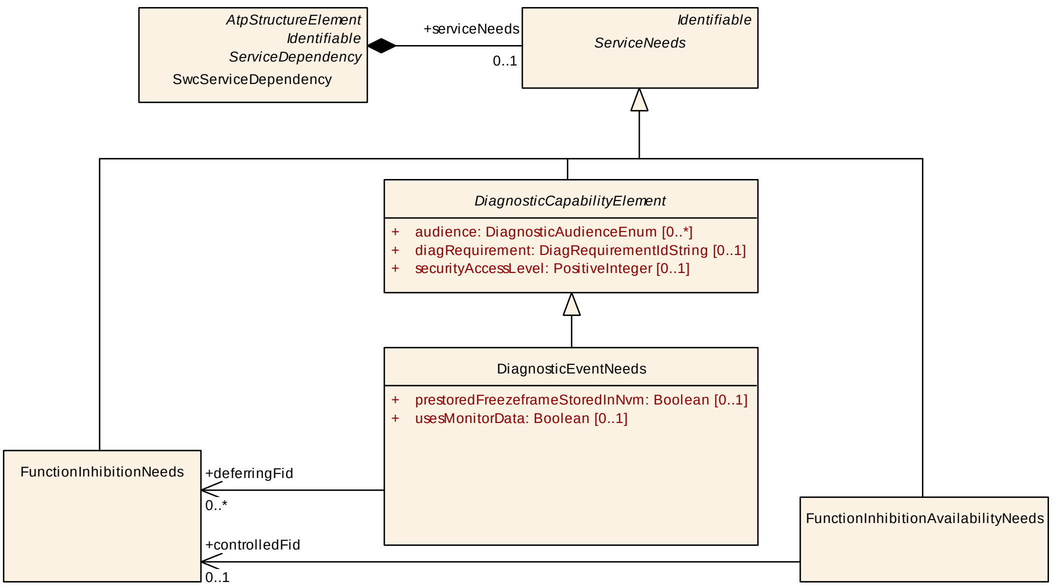


Figure 13.12: Modeling of FunctionInhibitionNeeds and FunctionInhibitionAvailabilityNeeds

13.8.2.1 Function Inhibition Manager Service use Case: read function permission

[TPS_SWCT_02505] Setup for Function Inhibition Manager Service use Case: read function permission ⌈Scenario: a AtomicSwComponentType read the function permission from FiM in order to enable or disable a functionality. In this case the following setup apply:

ServiceNeeds kind FunctionInhibitionNeeds

RoleBasedPortAssignment valid roles:

- FunctionInhibition [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Fim_00090].

13.8.2.2 Function Inhibition Manager Use Case: react on suppressed or unavailable events

[TPS_SWCT_01739] Function Inhibition Manager Use Case: react on suppressed or unavailable events ⌈Scenario: an AtomicSwComponentType wants to react on suppressed or unavailable events and disable the permission to run for a FID. In this case, the following setup applies:

ServiceNeeds kind FunctionInhibitionAvailabilityNeeds

RoleBasedPortAssignment valid roles:

- ControlFunctionAvailable [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Note: for variant coding ClientServerInterface ControlFunctionAvailable is used to deactivate a certain functionality (e.g. to set the FID to not available).

For more information, please refer to [SWS_Fim_00107].

13.8.3 Diagnostic Event Needs

The meta-classes DiagnosticEventManagerNeeds is used to define requirements in order to configure the Diagnostic Event Manager Service.

An SwcInternalBehavior may provide several DiagnosticEventManagerNeeds elements that define the mappings for the general diagnostic event manager behavior (for the terms related to the AUTOSAR Diagnostic Event Manager see [42]).

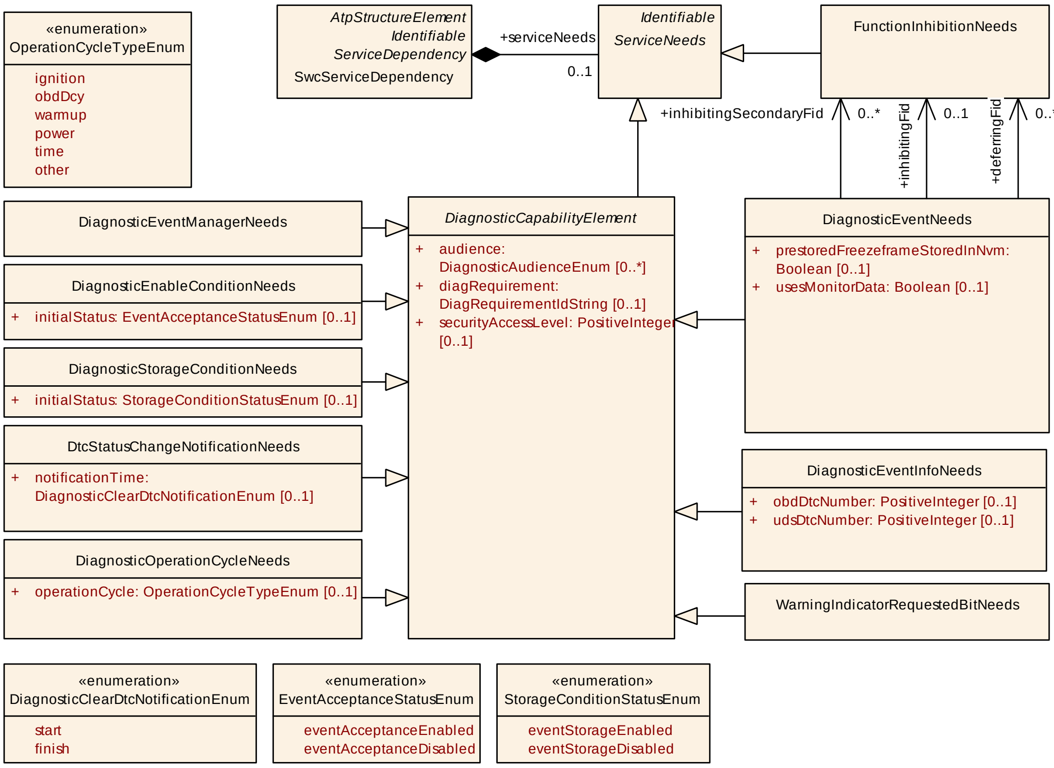


Figure 13.13: General diagnostic event-handling related ServiceNeeds

Class	**DiagnosticEventManagerNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Event Manager (Dem) which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.13: DiagnosticEventManagerNeeds

The meta-class DiagnosticCapabilityElement is used to provide generic information about diagnostic capabilities. Further on, the usage of DiagnosticCapabilityElement indicates that all ServiceNeeds which inherit from DiagnosticCapabilityElement express the following intentions:

- Need to interact with AUTOSAR Service `Dem` or `Dcm`.
- Provide services for the on-board diagnostics.

Class	***DiagnosticCapabilityElement*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class identifies the capability to provide generic information about diagnostic capabilities			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Subclasses	DiagnosticCommunicationManagerNeeds, DiagnosticComponentNeeds, DiagnosticControlNeeds, DiagnosticEnableConditionNeeds, DiagnosticEventInfoNeeds, DiagnosticEventManagerNeeds, DiagnosticEventNeeds, DiagnosticIoControlNeeds, DiagnosticOperationCycleNeeds, DiagnosticRequestFileTransferNeeds, DiagnosticRoutineNeeds, DiagnosticStorageConditionNeeds, DiagnosticUploadDownloadNeeds, DiagnosticValueNeeds, DiagnosticsCommunicationSecurityNeeds, DtcStatusChangeNotificationNeeds, ObdControlServiceNeeds, ObdInfoServiceNeeds, ObdMonitorServiceNeeds, ObdPidServiceNeeds, ObdRatioDenominatorNeeds, ObdRatioServiceNeeds, WarningIndicatorRequestedBitNeeds			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
audience	DiagnosticAudienceEnum	*	attr	This specifies the intended audience for the diagnostic object. Note that this is not only for the documentation but also subsequent audience specific implementation.
diagRequirement	DiagRequirementIdString	0..1	attr	This denotes the requirement identifier to which the object can be linked to. Note that with the implementation of a generic tracing concept in AUTOSAR this attribute might become obsolete.
securityAccessLevel	PositiveInteger	0..1	attr	This attribute denotes the level of security which is touched by the diagnostic object. The higher the level the more relevance for the security exists. This level shall be mapped to the security level in the ECU.

Table 13.14: DiagnosticCapabilityElement

Enumeration	**DiagnosticAudienceEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	The possible values of the intended audience for a diagnostic object.
Aggregated by	*DiagnosticCapabilityElement*.audience
Literal	**Description**
aftermarket	The object is for free aftermarket service organizations. **Tags:**atp.EnumerationLiteralIndex=1
afterSales	The object is relevant for the OEM after-sales organization. **Tags:**atp.EnumerationLiteralIndex=2
development	The object is relevant for engineering only. **Tags:**atp.EnumerationLiteralIndex=3
manufacturing	The object is relevant for manufacturing. **Tags:**atp.EnumerationLiteralIndex=4
supplier	The object is relevant for the ECU-supplier aftermarket organization. **Tags:**atp.EnumerationLiteralIndex=5

Table 13.15: DiagnosticAudienceEnum

The meta-classes `DiagnosticEventNeeds` is used to define requirements to configure the Diagnostic Event Manager Service.

An SwcInternalBehavior may provide several DiagnosticEventNeeds elements where each defines all the requirements related to one diagnostic event (for the terms related to the AUTOSAR Diagnostic Event Manager see [42]).

In addition, ObdPidServiceNeeds and ObdRatioServiceNeeds are required in order to specify the needs for OBD diagnostic service calls.

The diagEventDebounceAlgorithm attribute defines the kind of expected debouncing by the Diagnostic Event Manager or defines that the debouncing is implemented by the software component.

The class DiagEventDebounceAlgorithm inherits from Identifiable in order to allow further documentation of the debouncing algorithm as well as non formalized description or non standardized description by the means of Sdg on expected configuration of the DiagEventDebounceAlgorithm in the Diagnostic Event Manager.

[constr_1138] SwcServiceDependency.assignedPort and DiagEventDebounceMonitorInternal ⌈If a SwcServiceDependency aggregates DiagnosticEventNeeds in the role serviceNeeds, then an assignedPort with attribute role set to the value CallbackGetFaultDetectCounter shall only exist if the monitor implements internal debouncing, i.e concrete subclass DiagEventDebounceMonitorInternal is aggregated in the role DiagnosticEventNeeds.diagEventDebounceAlgorithm.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

Rationale for the existence of [constr_1138]: the *fault detect counter* is typically an internal state of the Dem, i.e. if the debouncing is executed in the Dem internally.

If, on the other hand, the debouncing is implemented in the monitor software-component, it may happen that a diagnostic tester requests the value of the *fault detect counter*.

In this case, the Dem will request the value of the fault detect counter via the PortPrototype to which the assignedPort with attribute role set to the value CallbackGetFaultDetectCounter refers to.

If the monitor does not implement the debouncing, then this PortPrototype shall not exist.

[constr_10104] RoleBasedPortAssignment where attribute role is set to CallbackGetFaultDetectCounter shall refer to a PPortPrototype in the role portPrototype ⌈If a SwcServiceDependency aggregates both

- a DiagnosticEventNeeds that in turn aggregates DiagEventDebounceMonitorInternal in the role diagEventDebounceAlgorithm and
- a RoleBasedPortAssignment in the role assignedPort where attribute role is set to CallbackGetFaultDetectCounter,

then the target of the reference SwcServiceDependency.assignedPort.portPrototype shall be a PPortPrototype.

This rule shall be imposed `at the time when the RTE is generated.`⌋()

Class	DiagnosticEventNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Diagnostic Event Manager for one diagnostic event. Its shortName can be regarded as a symbol identifying the diagnostic event from the viewpoint of the component or module which owns this element. In case the diagnostic event specifies a production error, the shortName shall be the name of the production error.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
deferringFid	FunctionInhibitionNeeds	*	ref	This reference contains the link to a function identifier within the FiM which is used by the monitor before delivering a result.
diagEvent Debounce Algorithm	DiagEventDebounce Algorithm	0..1	aggr	Specifies the abstract need on the Debounce Algorithm applied by the Diagnostic Event Manager.
inhibitingFid	FunctionInhibitionNeeds	0..1	ref	This represents the primary Function Inhibition Identifier used for inhibition of the diagnostic monitor. The FID might either inhibit the monitoring of a symptom or the reporting of detected faults.
inhibiting SecondaryFid	FunctionInhibitionNeeds	*	ref	This represents the secondary Function Inhibition Identifier used for inhibition of the diagnostic monitor. Any of the FID inhibitions leads to an inhibition of the monitoring of a symptom or the reporting of detected faults.
prestored Freezeframe StoredInNvm	Boolean	0..1	attr	If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestored FreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm).
usesMonitor Data	Boolean	0..1	attr	This attribute defines whether additional monitor data shall be added to the reporting of events.

Table 13.16: DiagnosticEventNeeds

Class	*DiagEventDebounceAlgorithm* (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class represents the ability to specify the pre-debounce algorithm which is selected and/or required by the particular monitor. This class inherits from Identifiable in order to allow further documentation of the expected or implemented debouncing and to use the category for the identification of the expected / implemented debouncing.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Subclasses	DiagEventDebounceCounterBased, DiagEventDebounceMonitorInternal, DiagEventDebounceTime Based			
Aggregated by	DiagnosticDebounceAlgorithmProps.debounceAlgorithm, DiagnosticEventNeeds.diagEventDebounce Algorithm			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.17: DiagEventDebounceAlgorithm

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounce CounterBased.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	DiagnosticDebounceAlgorithmProps.debounceAlgorithm, DiagnosticEventNeeds.diagEventDebounce Algorithm			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
counterBased FdcThreshold StorageValue	Integer	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
counter DecrementStep Size	Integer	0..1	attr	This value shall be taken to decrement the internal debounce counter. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterFailed Threshold	Integer	0..1	attr	This value defines the event-specific limit that indicates the "failed" counter status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counter IncrementStep Size	Integer	0..1	attr	This value shall be taken to increment the internal debounce counter. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJump Down	Boolean	0..1	attr	This value activates or deactivates the counter jump-down behavior. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJump DownValue	Integer	0..1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from incrementing to decrementing. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJumpUp	Boolean	0..1	attr	This value activates or deactivates the counter jump-up behavior. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterJumpUp Value	Integer	0..1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
counterPassed Threshold	Integer	0..1	attr	This value defines the event-specific limit that indicates the "passed" counter status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 13.18: DiagEventDebounceCounterBased

Class	DiagEventDebounceTimeBased			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to DemDebounceTime Base.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	DiagnosticDebounceAlgorithmProps.debounceAlgorithm, DiagnosticEventNeeds.diagEventDebounce Algorithm			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
timeBasedFdc Threshold StorageValue	TimeValue	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
timeFailed Threshold	TimeValue	0..1	attr	This value represents the event-specific delay indicating the "failed" status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
timePassed Threshold	TimeValue	0..1	attr	This value represents the event-specific delay indicating the "passed" status. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table 13.19: DiagEventDebounceTimeBased

Class	DiagEventDebounceMonitorInternal			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	"This meta-class represents the ability to indicate that no Dem pre-debounce algorithm shall be used for this diagnostic monitor. The SWC might implement an internal debouncing algorithm and report qualified (debounced) results to the Dem/DM.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	DiagnosticDebounceAlgorithmProps.debounceAlgorithm, DiagnosticEventNeeds.diagEventDebounce Algorithm			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.20: DiagEventDebounceMonitorInternal

The figure 13.14 shows the relationship of the class `DiagnosticEventNeeds`. The given M2 structure support to express following properties of a diagnostic monitor in addition to the basic set of attributes provided by `DiagnosticCapabilityElement`:

The used `PortPrototype` which has to be connected to the Function Inhibition Managers is determined by the `RoleBasedPortAssignment` of the related `FunctionInhibitionNeeds` instance on M1.

The reference from a M1 instance of an `ObdRatioServiceNeeds` to an M1 instance of a `DiagnosticEventNeeds` specifies that the related Diagnostic Monitor supports Rate Based Monitoring. For further details see section 13.8.5.

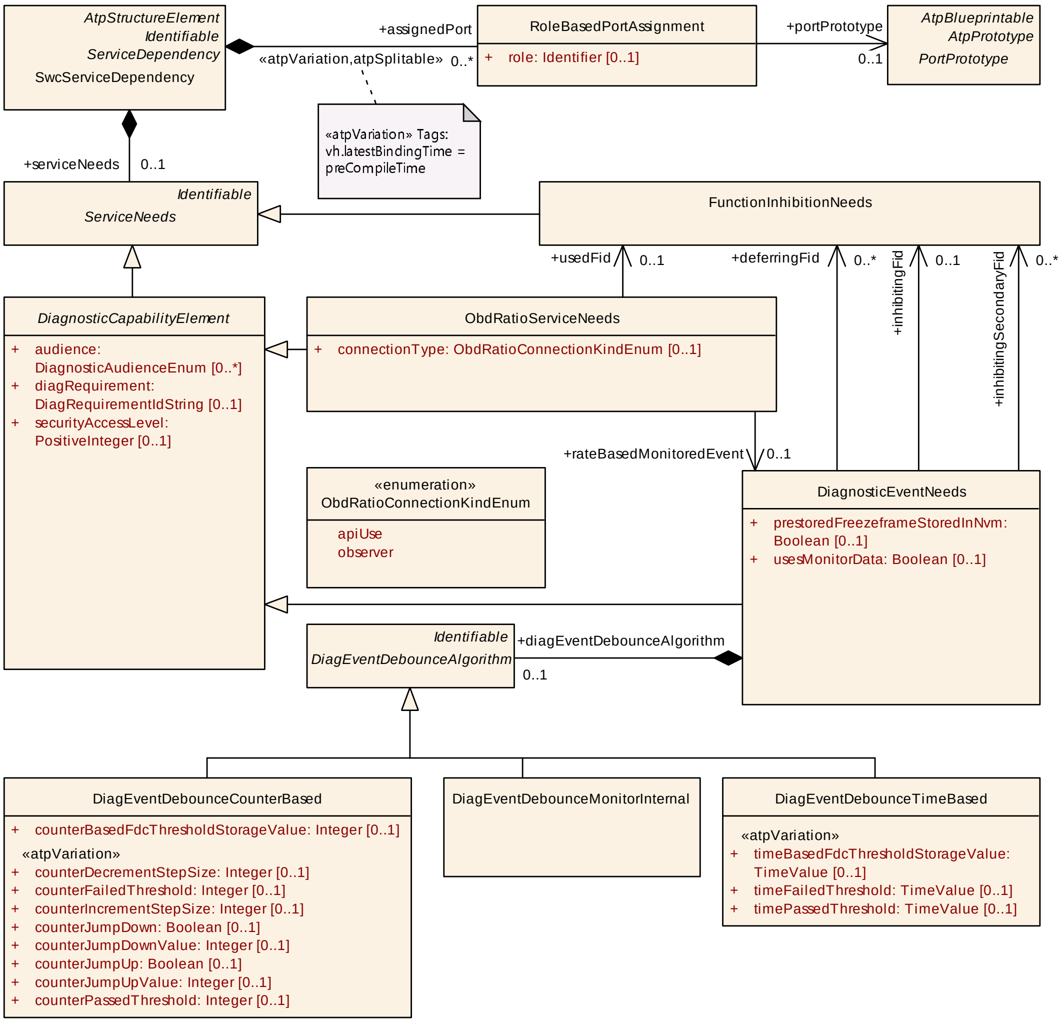


Figure 13.14: Relationship of DiagnosticEventNeeds and FunctionInhibitionNeeds

[TPS_SWCT_01864] Semantics of DiagnosticEventNeeds.inhibitingFid and DiagnosticEventNeeds.inhibitingSecondaryFid ⌈Diagnostic monitor implementations use Function Identifiers (FID) to acquire permission from FiM before executing the fault detection.

Typically, the permission is not granted by FiM if other diagnostic events have already been reported as FAILED, which would lead to a double-detection of the same failure.

These Function Inhibitions are specified by means of the references in the roles DiagnosticEventNeeds.inhibitingFid and DiagnosticEventNeeds.inhibitingSecondaryFid.

Semantically, there is no difference between DiagnosticEventNeeds.inhibitingFid and DiagnosticEventNeeds.inhibitingSecondaryFid, i.e. the collections of relevant references in the "inhibiting" role is created out of the merged content of DiagnosticEventNeeds.inhibitingFid and DiagnosticEventNeeds.inhibitingSecondaryFid.⌋*()*

This relationship describes the implementation of the software-component. Through this formal reference, the implementer of the software-component can make this information available for later process steps.

Example use-case 1: Certification documentation requires a description of the inhibitions.

Example use-case 2: If a Ratio is added to the ECU-configuration at a later step for a DemEvent (not requested through a ServiceDependency with ObdRatioServiceNeeds), the configurator can derive the related FIDs for the ratio from the inhibiting/deferring FIDs of the DemEvent itself.

[TPS_SWCT_01582] Semantics of DiagnosticEventNeeds.deferringFid
⌈Diagnostic monitor implementations use *Function Identifiers* (FID) to acquire permission from FiM before executing the fault detection.

Typically, the permission is not granted by FiM if other *Events* have already been reported as *FAILED*, which would lead to a double-detection of the same failure.

In some cases (see [41]), diagnostic monitor implementations do not only shut down completely in case of “no permission”, but fully compute their result and do just not deliver it to Dem before further conditions are fulfilled.

Typically, such diagnostics can detect a coarse failure quickly. But it avoids reporting *FAIL* early to give other Events a chance to deliver a more precise *FAIL*.

In such cases, the delivery of the result is only allowed when FiM grants a permission, with inhibitions on NOT_TESTED of other Events. These *Function Inhibitions* are specified by means of the attribute DiagnosticEventNeeds.deferringFid.

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

As a corresponding concept to DiagnosticEventNeeds, the DiagnosticEventInfoNeeds represents the needs to a given software-component that is interested to get information about specific DTCs.

Class	**DiagnosticEventInfoNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component interested to get information regarding specific DTCs.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
obdDtcNumber	PositiveInteger	0..1	attr	This represents a reasonable Diagnostic Trouble Code. This allows to predefine the Diagnostic Trouble Code, e.g. if the function developer has received a particular requirement from the OEM or from a standardization body. This attribute applies for the OBD diagnostics use case.

▽

△

Class	DiagnosticEventInfoNeeds			
udsDtcNumber	PositiveInteger	0..1	attr	This represents a reasonable Diagnostic Trouble Code. This allows to predefine the Diagnostic Trouble Code, e.g. if the function developer has received a particular requirement from the OEM or from a standardization body. This attribute applies for the UDS diagnostics use case.

Table 13.21: DiagnosticEventInfoNeeds

Class	DiagnosticOperationCycleNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component to provide information regarding the operation cycle management to the Dem module.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
operationCycle	OperationCycleTypeEnum	0..1	attr	Operation cycles types for the Dem to be supported by cycle-state APIs.

Table 13.22: DiagnosticOperationCycleNeeds

Enumeration	OperationCycleTypeEnum
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	The possible values of the operation cycles types for the Dem.
Aggregated by	DiagnosticOperationCycleNeeds.operationCycle
Literal	**Description**
ignition	Ignition ON / OFF cycle. **Tags:**atp.EnumerationLiteralIndex=0
obdDcy	OBD Driving cycle. **Tags:**atp.EnumerationLiteralIndex=1
other	Further operation cycle. **Tags:**atp.EnumerationLiteralIndex=2
power	Power ON / OFF cycle. **Tags:**atp.EnumerationLiteralIndex=3
time	Time based operation cycle. **Tags:**atp.EnumerationLiteralIndex=4
warmup	OBD Warm up cycle. **Tags:**atp.EnumerationLiteralIndex=5

Table 13.23: OperationCycleTypeEnum

Class	**DiagnosticEnableConditionNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component to provide the capability to set an enable condition.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
initialStatus	EventAcceptanceStatus Enum	0..1	attr	Defines the initial status for enable or disable of acceptance of event reports of a diagnostic event.

Table 13.24: DiagnosticEnableConditionNeeds

Enumeration	**EventAcceptanceStatusEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumerator specifies the initial status for enable or disable of acceptance of event reports of a diagnostic event.
Aggregated by	DiagnosticEnableConditionNeeds.initialStatus
Literal	*Description*
eventAcceptance Disabled	Acceptance of a diagnostic event is disabled. **Tags:**atp.EnumerationLiteralIndex=0
eventAcceptance Enabled	Acceptance of a diagnostic event is enabled. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.25: EventAcceptanceStatusEnum

Class	**DiagnosticStorageConditionNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component to provide the capability to set a storage condition.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
initialStatus	StorageConditionStatus Enum	0..1	attr	Defines the initial status for enable or disable of storage of a diagnostic event.

Table 13.26: DiagnosticStorageConditionNeeds

Enumeration	**StorageConditionStatusEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration specifies the initial status for enable or disable of storage of a diagnostic event.
Aggregated by	DiagnosticStorageConditionNeeds.initialStatus
Literal	*Description*
eventStorage Disabled	Storage of a diagnostic event is disabled. **Tags:**atp.EnumerationLiteralIndex=0
eventStorage Enabled	Storage of a diagnostic event is enabled. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.27: StorageConditionStatusEnum

13.8.3.1 Dem Service Use Case: diagnostic monitor, debouncing by Dem

Scenario: an AtomicSwComponentType implements a Diagnostic Monitor. The debouncing of the failure condition shall be configured and processed by the Dem. In this case the following setup apply:

[TPS_SWCT_01028] AtomicSwComponentType implements a Diagnostic Monitor
⌈

ServiceNeeds kind DiagnosticEventNeeds

RoleBasedPortAssignment valid roles:

- DiagnosticMonitor [1]
- DiagnosticInfo [0..1]
- CallbackInitMonitorForEvent [0..1]
- CallbackEventUdsStatusChanged [0..1]
- CallbackClearEventAllowed [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Please note that for the implementation of this scenario DiagEventDebounceCounterBased or DiagEventDebounceTimeBased algorithm should be used as diagEventDebounceAlgorithm.

13.8.3.2 Dem Service Use Case: diagnostic monitor, debouncing by SWC

Scenario: an AtomicSwComponentType implements a Diagnostic Monitor. The debouncing of the failure condition shall be processed by the software component. In this case the following setup applies:

[TPS_SWCT_01029] AtomicSwComponentType implements a Diagnostic Monitor
⌈

ServiceNeeds kind DiagnosticEventNeeds

RoleBasedPortAssignment valid roles:

- DiagnosticMonitor [1]
- DiagnosticInfo [0..1]
- CallbackInitMonitorForEvent [0..1]

- CallbackEventUdsStatusChanged [0..1]
- CallbackClearEventAllowed [0..1]
- CallbackGetFaultDetectCounter [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Please note that for the implementation of this scenario DiagEventDebounceMonitorInternal algorithm should be used as diagEventDebounceAlgorithm.

13.8.3.3 Dem Service Use Case: software-component provides information about operation cycles

Scenario: an AtomicSwComponentType provides information about operating cycles, e.g. ignition cycle or driving cycle.

[TPS_SWCT_01132] AtomicSwComponentType provides information about operating cycles ⌈

ServiceNeeds kind DiagnosticOperationCycleNeeds

RoleBasedPortAssignment valid roles:

- OperationCycle [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00601].

13.8.3.4 Dem Service Use Case: software-component enables reporting of DTCs in general

Scenario: a AtomicSwComponentType enables the reporting of DTCs in general.

[TPS_SWCT_01134] AtomicSwComponentType enables reporting of DTCs in general ⌈

ServiceNeeds kind DiagnosticEnableConditionNeeds

RoleBasedPortAssignment valid roles:

- EnableCondition [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00604] and [ECUC_Dem_00656].

13.8.3.5 Dem Service Use Case: software-component enables storage of subsequent DTCs

Scenario: an AtomicSwComponentType enables the storage of subsequent DTCs.

[TPS_SWCT_01135] AtomicSwComponentType enables storage of subsequent DTCs ⌈

ServiceNeeds kind DiagnosticStorageConditionNeeds

RoleBasedPortAssignment valid roles:

- StorageCondition [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00605].

The relevant DTCs shall be configured in ECUC because at the time the AtomicSwComponentType is designed the information about which DTCs are relevant is not fully available.

13.8.3.6 Dem Service Use Case: retrieve information of the lamp status

Scenario: an AtomicSwComponentType retrieves information of the lamp status.

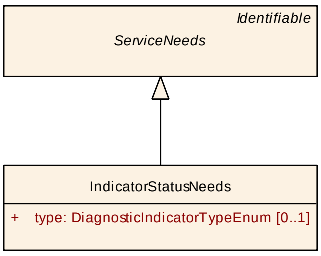


Figure 13.15: Modeling of IndicatorStatusNeeds

[TPS_SWCT_01136] AtomicSwComponentType retrieves information of the lamp status ⌈

ServiceNeedsKind IndicatorStatusNeeds

RoleBasedPortAssignment valid roles:

- IndicatorStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00606].

Class	**IndicatorStatusNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class shall be taken to signal a service use case that affects the indicator status.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	DiagnosticIndicatorTypeEnum	0..1	attr	Defines the type of the indicator.

Table 13.28: IndicatorStatusNeeds

Enumeration	**DiagnosticIndicatorTypeEnum**
Package	M2::AUTOSARTemplates::DiagnosticExtract::Dem::DiagnosticIndicator
Note	Type of an indicator.
Aggregated by	DiagnosticIndicator.type, IndicatorStatusNeeds.type
Literal	***Description***
amberWarning	Amber Warning Lamp **Tags:**atp.EnumerationLiteralIndex=0
malfunction	Malfunction Indicator Lamp **Tags:**atp.EnumerationLiteralIndex=1

▽

△

Enumeration	**DiagnosticIndicatorTypeEnum**
protectLamp	Protect Lamp **Tags:**atp.EnumerationLiteralIndex=2
redStopLamp	Red Stop Lamp **Tags:**atp.EnumerationLiteralIndex=3
warning	Warning **Tags:**atp.EnumerationLiteralIndex=4

Table 13.29: DiagnosticIndicatorTypeEnum

13.8.3.7 Dem Service Use Case: DEM provides information that the fault storage overflows

Please note that for this specific use case the application of a concrete ServiceNeeds *is not yet clarified.*

Scenario: the Dem provides information that the fault storage overflows.

[TPS_SWCT_01137] Dem provides information that the fault storage overflows ⌈

RoleBasedPortAssignment valid roles:

- EvMemOverflowIndication [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00607].

13.8.3.8 Dem Service Use Case: software-component suppresses the storage of DTCs

Scenario: an AtomicSwComponentType suppresses the storage of DTCs within the Dem.

[TPS_SWCT_01138] AtomicSwComponentType suppresses the storage of DTCs within the Dem ⌈

ServiceNeeds kind DiagnosticEventManagerNeeds

RoleBasedPortAssignment valid roles:

- DTCSuppression [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00608].

13.8.3.9 Dem Service Use Case: software-component informs that the PTO is active

Scenario: an AtomicSwComponentType informs the Dem that the PTO is active.

[TPS_SWCT_01139] AtomicSwComponentType informs the Dem that the PTO is active ⌈

ServiceNeeds kind DiagnosticEventManagerNeeds

RoleBasedPortAssignment
The following roles are applicable:

- PowerTakeOff [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00612].

13.8.3.10 Dem Service Use Case: software-component needs information about any DTC status change

Scenario: an AtomicSwComponentType needs information about any DTC status change. There is no limitation on the number of software-components requesting the information.

[TPS_SWCT_01140] AtomicSwComponentType needs information about specific DTC without being a diagnostic monitor ⌈

ServiceNeeds kind DtcStatusChangeNotificationNeeds

RoleBasedPortAssignment valid roles:

- CallbackDTCStatusChange [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00617].

In the case the software-component needs notifications about different kinds of the DTC status change, it is advised to create a SwcServiceDependency for each kind of status change.

13.8.3.11 Dem Service Use Case: call operation if the data of a given diagnostic event changes (I)

Scenario: an AtomicSwComponentType provides a PPortPrototype typed by the ClientServerInterface CallbackEventDataChanged. The service component calls the ClientServerOperation EventDataChanged if the corresponding diagnostic event changes in terms of the underlying data.

For each diagnostic events to which the AtomicSwComponentType is conceptually connected it needs to provide one PPortPrototype towards the service component.

[TPS_SWCT_01425] AtomicSwComponentType provides one callback per event if diagnostic event data change ⌈

ServiceNeeds kind DiagnosticEventInfoNeeds

RoleBasedPortAssignment valid roles:

- CallbackEventDataChanged [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00618].

13.8.3.12 Dem Service Use Case: call operation if the data or status of any diagnostic event changes (II)

Scenario: an AtomicSwComponentType shall react on any diagnostic event status change and/or any diagnostic event data change. For instance this may be used to write a time stamp when any event status changes regardless of the event id.

In contrast to the scenario described in chapter 13.8.3.11 or 13.8.3.10 this case foresees the existence of a single PPortPrototype that covers all relevant diagnostic events.

[TPS_SWCT_01426] AtomicSwComponentType provides callback if any diagnostic event data and/or status changed ⌈

ServiceNeeds kind DiagnosticEventManagerNeeds

RoleBasedPortAssignment valid roles:

- GeneralCallbackEventDataChanged [0..1]
- GeneralCallbackEventUdsStatusChange [0..1]
- GeneralDiagnosticInfo [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

For more information please refer to [SWS_Dem_00616], [SWS_Dem_00619], and [SWS_Dem_00600].

In order to react on diagnostic event status changes the software component shall provide a single PPortPrototype typed as a client server interface compatible to GeneralCallbackEventDataChanged.

In order to react on diagnostic event data changes the software component shall provide a single PPortPrototype typed as a client server interface compatible to GeneralCallbackEventDataChanged.

If the software-component additionally has to read further information of the specific diagnostic event from Dem it shall provide a RPortPrototype typed as a client server interface compatible to GeneralDiagnosticInfo. It shall also specify DiagnosticEventInfoNeeds.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

13.8.3.13 Dem Service Use Case: software-component provides data for diagnostic purposes

Please note that for this specific use case the application of a concrete ServiceNeeds is not yet clarified.

Scenario: an AtomicSwComponentType provides data to be used for diagnostic purposes. The provision of data can be done by means of PPortPrototypes typed by either ClientServerInterfaces or SenderReceiverInterfaces. The usage of the latter, however, is not further detailed in the applicable SWS [42] and therefore no more details are to be provided in this document.

[TPS_SWCT_01427] AtomicSwComponentType provides data for diagnostic purposes via ClientServerInterface ⌈

RoleBasedPortAssignment valid roles:

- DataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01634] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (DataServices_{Data}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00621].

13.8.3.14 Dem Service Use Case: software-component gets information about a specific DTC

Scenario: an AtomicSwComponentType specifies DiagnosticEventInfoNeeds in order to be able to get information about specific DTCs. This use case to some extent is similar to [TPS_SWCT_01426] but does not replace that use case.

[TPS_SWCT_01453] Software-component gets information about a specific DTC ⌈

ServiceNeeds kind DiagnosticEventInfoNeeds

RoleBasedPortAssignment valid roles:

- DiagnosticInfo [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00599].

13.8.3.15 Dem Service Use Case: Software-Component wants to be triggered on Monitor Status Changes

Scenario: a software-component wants to be triggered on monitor status changes if this is supported for the specific monitor status. Events reported from basic-software modules cannot be considered in this service use case.

The Dem will not provide corresponding PortPrototypes for events reported by basic-software modules.

This way, the service use case cannot be used for events reported by the basic-software.

However, for the creator of the service use case there is no way to find out whether the event will be reported by basic software of application software-component.

[TPS_SWCT_01715] Software-Component wants to be triggered on Monitor Status Changes ⌈

ServiceNeeds kind DiagnosticEventManagerNeeds

RoleBasedPortAssignment valid roles:

- CallbackMonitorStatusChange [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

13.8.3.16 Dem Service Use Case: write parameter identifier (PID) by software-component

Scenario: A software-component computes the PIDs, and pushes them to Dem for storage and reporting to Dcm.

[TPS_SWCT_01766] Software-component computes the PIDs, and pushes them to Dem for storage and reporting to Dcm ⌈

ServiceNeeds kind ObdPidServiceNeeds

RoleBasedPortAssignment valid roles:

- SetDataOfPID21 [1]
- SetDataOfPID4D [1]
- SetDataOfPID4E [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00170*, *RS_SWCT_03190*)

13.8.3.17 Dem Service Use Case: read parameter identifier by software-component

Scenario: A software-component located on an `OBD master ECU` reads the PID 21, and sends the value around via "regular" sender-receiver communication to other software-components located on `OBD primary ECUs` with the obligation to push the PID value to their local Dems.

[TPS_SWCT_01767] Software-component located on an `OBD master ECU` reads the PID 21, and sends the value around via "regular" sender-receiver communication ⌈

ServiceNeeds kind `ObdPidServiceNeeds`

RoleBasedPortAssignment valid roles:

- `GetDataOfPID21` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00170*, *RS_SWCT_03190*)

13.8.3.18 Dem Service Use Case: diagnostic monitor provides monitor data, debouncing by Dem

Scenario: an `AtomicSwComponentType` implements a Diagnostic Monitor that is able to provide monitor data. The debouncing of the failure condition shall be configured and processed by the Dem. In this case the following setup applies:

[TPS_SWCT_01789] `AtomicSwComponentType` implements a Diagnostic Monitor that provides monitor data, debouncing by Dem ⌈

ServiceNeeds kind `DiagnosticEventNeeds` (with attribute `usesMonitorData` set to `true`)

RoleBasedPortAssignment valid roles:

- `DiagnosticMonitor` [0..1]

- `DiagnosticMonitor_MonitorData` [1]
- `DiagnosticInfo` [0..1]
- `CallbackInitMonitorForEvent` [0..1]
- `CallbackEventStatusChange` [0..1]
- `CallbackClearEventAllowed` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Please note that for the implementation of this scenario the sub-class `DiagEventDebounceCounterBased` or `DiagEventDebounceTimeBased` algorithm should be used as the value for attribute `DiagnosticEventNeeds.diagEventDebounceAlgorithm`.

13.8.3.19 Dem Service Use Case: diagnostic monitor provides monitor data, debouncing by software-component

Scenario: an `AtomicSwComponentType` implements a Diagnostic Monitor that is able to provide monitor data. The debouncing of the failure condition shall be configured and processed by the software-component that implements the monitor. In this case the following setup applies:

[TPS_SWCT_01790] **`AtomicSwComponentType` implements a Diagnostic Monitor that provides monitor data, debouncing by software-component** ⌈

ServiceNeeds kind `DiagnosticEventNeeds` (with attribute `usesMonitorData` set to `true`)

RoleBasedPortAssignment valid roles:

- `DiagnosticMonitor` [0..1]
- `DiagnosticMonitor_MonitorData` [1]
- `DiagnosticInfo` [0..1]
- `CallbackInitMonitorForEvent` [0..1]
- `CallbackEventStatusChange` [0..1]
- `CallbackClearEventAllowed` [0..1]
- `CallbackGetFaultDetectCounter` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Please note that for the implementation of this scenario the sub-class DiagEvent-DebounceMonitorInternal should be used as the value for attribute Diagnos-ticEventNeeds.diagEventDebounceAlgorithm.

13.8.3.20 Dem Service Use Case: software-component checks whether an event is suppressed

Scenario: a software-component representing a monitor of a given event needs to check for the availability of the event in order to decide whether reporting of that event is cleared by the Dem.

A typical use case for this ability is a scenario where actuators are moved to their limits for testing purposes. During this phase the monitoring shall cease.

For this purpose the software component needs to expose an RPortPrototype towards the Dem.

[TPS_SWCT_01808] Dem Service Use Case: software-component checks whether an event is suppressed ⌈

ServiceNeeds kind DiagnosticEventInfoNeeds

RoleBasedPortAssignment valid roles:

- DiagnosticInfo [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

13.8.3.21 Dem Service Use Case: software-component wants information about DTC clearance

An AtomicSwComponentType wants information about DTC clearance. The execution of the ClearDtcNotification callback can be triggered at the start of the DTC operation (i.e. DtcStatusChangeNotificationNeeds.notificationTime = DiagnosticClearDtcNotificationEnum.start) or when the DTC operation ends (i.e. i.e. DtcStatusChangeNotificationNeeds.notificationTime = DiagnosticClearDtcNotificationEnum.finish).

For this purpose, the AtomicSwComponentType shall expose a PPortPrototype typed by the standardized AUTOSAR ClientServerInterface named ClearDtcNotification (see [SWS_Dem_91003]).

[TPS_SWCT_01862] AtomicSwComponentType wants information about DTC clearance ⌈

ServiceNeeds kind DtcStatusChangeNotificationNeeds

RoleBasedPortAssignment valid roles:

- ClearDtcNotification [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_01240] and [SWS_Dem_01241].

[constr_10123] Existence of attribute DtcStatusChangeNotificationNeeds.notificationTime ⌈Attribute DtcStatusChangeNotificationNeeds.notificationTime shall only exist if the enclosing SwcServiceDependency contains a RoleBasedPortAssignment where attribute role is set to the value ClearDtcNotification **at the time when the RTE is generated**.⌋*()*

Class	**DtcStatusChangeNotificationNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component interested to get information regarding any DTC status change.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
notificationTime	DiagnosticClearDtcNotificationEnum	0..1	attr	This attribute determines the time when the notification about the DTC operation shall be executed. This attribute is only relevant for the configuration of the ClearDtcNotification.

Table 13.30: DtcStatusChangeNotificationNeeds

Enumeration	**DiagnosticClearDtcNotificationEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration supports the specification of the time when the ClearDtcNotification callback is supposed to be executed.

▽

△

Enumeration	**DiagnosticClearDtcNotificationEnum**
Aggregated by	DtcStatusChangeNotificationNeeds.notificationTime
Literal	***Description***
finish	The ClearDtcCallback shall be executed when the DTC operation finishes. **Tags:**atp.EnumerationLiteralIndex=1
start	The ClearDtcCallback shall be executed when the DTC operation starts. **Tags:**atp.EnumerationLiteralIndex=0

Table 13.31: DiagnosticClearDtcNotificationEnum

13.8.3.22 Dem Service Use Case: software-component wants to reset a degradation state

Scenario: an `AtomicSwComponentType` wants to reset a degradation state and therefore deactivate the "limp home" mode.

For this purpose, the software-component uses the `ClientServerOperation` `ResetMonitorStatus` that is defined in `ClientServerInterface` `DiagnosticMonitor`.

In this case, the following setup applies:

[TPS_SWCT_01870] `AtomicSwComponentType` wants to reset a degradation state ⌈

ServiceNeeds kind `DiagnosticEventNeeds`

RoleBasedPortAssignment valid roles:

- `DiagnosticMonitor` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Please note that the software-component might want to choose getting information about the associated DTC first, see [TPS_SWCT_01453].

13.8.4 Diagnostic Communication Needs

The meta-class `DiagnosticCommunicationManagerNeeds` is used to define requirements in order to configure the Diagnostic Communication Manager Service.

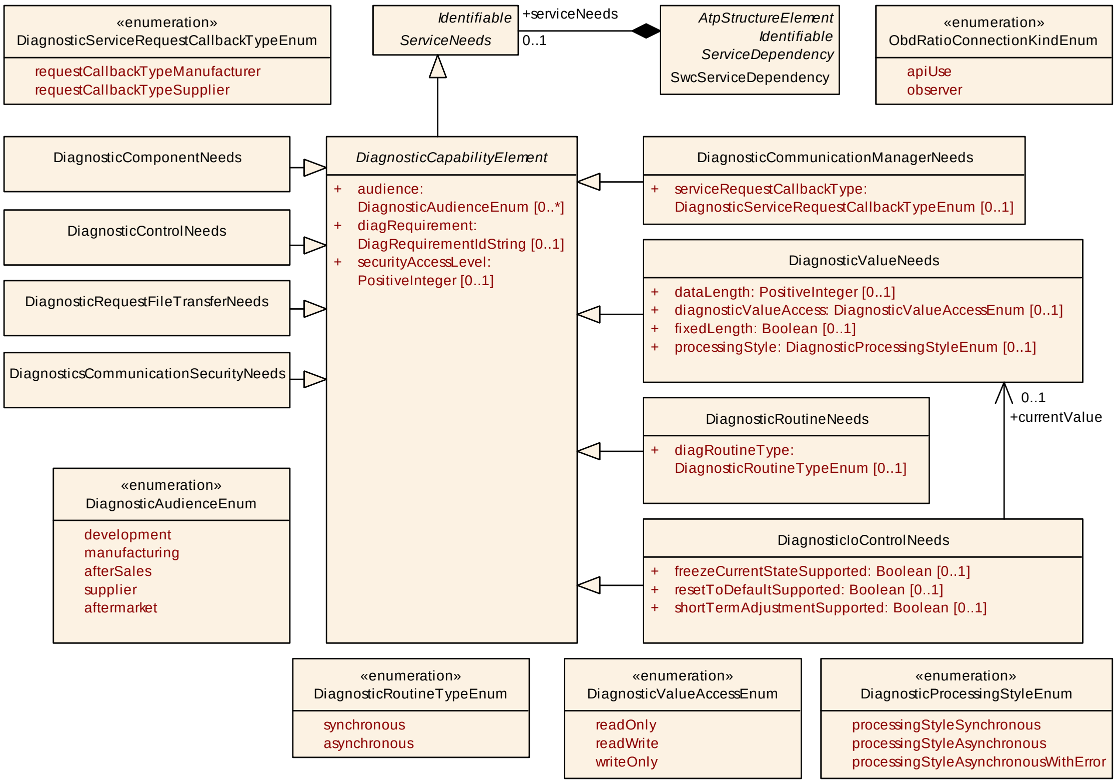


Figure 13.16: General diagnostic service-related ServiceNeeds

An SwcInternalBehavior may provide a DiagnosticCommunicationManagerNeeds element which defines the mappings for the general diagnostic communication (for the terms related to the AUTOSAR Diagnostic Communication Manager see [43]).

The meta-class DiagnosticRoutineNeeds is used to define requirements to configure the Diagnostic Communication Manager Service. A PPortPrototype typed by a ClientServerInterface[1] may provide ClientServerOperations (for example, “start”, “stop”, and “RequestResults”).

The PPortPrototype corresponds to the diagnostic service RoutineControl. Within the SwcInternalBehavior up to three RunnableEntitys are defined for implementing the ClientServerOperations mentioned before.

The enumeration parameter DiagnosticRoutineTypeEnum is used to define whether the diagnostic server or client is responsible for stopping the routine.

Please note that [constr_1340] and [constr_1341] apply for the application of DiagnosticRoutineNeeds. These constraints are part of the specification of the Diagnostic Extract.

[1]where isService shall be set to true

Class	**DiagnosticCommunicationManagerNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (Dcm) which are not related to a particular item (e.g. a PID or DiagnosticRoutineNeeds). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
serviceRequestCallbackType	DiagnosticServiceRequestCallbackTypeEnum	0..1	attr	This represents the ability to define whether the usage of PortInterface ServiceRequestNotification has the characteristics of being initiated by a manufacturer or by a supplier.

Table 13.32: DiagnosticCommunicationManagerNeeds

Enumeration	**DiagnosticServiceRequestCallbackTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This represents the ability to define whether a Service Request Notification was used in the role of a manufacturer or a supplier.
Aggregated by	DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType
Literal	*Description*
requestCallbackTypeManufacturer	This represents the case that the usage of PortInterface ServiceRequestNotification has the characteristics of being used by a manufacturer. **Tags:**atp.EnumerationLiteralIndex=0
requestCallbackTypeSupplier	This represents the case that the usage of PortInterface ServiceRequestNotification has the characteristics of being used by a supplier. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.33: DiagnosticServiceRequestCallbackTypeEnum

Class	**DiagnosticRoutineNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (Dcm) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
diagRoutineType	DiagnosticRoutineTypeEnum	0..1	attr	This denotes the type of diagnostic routine which is implemented by the referenced server port.

Table 13.34: DiagnosticRoutineNeeds

[constr_1986] Existence of the reference `DiagnosticRoutineNeeds.diagRoutineType` ⌈For each `DiagnosticRoutineNeeds`, the reference to `BswInternalBehavior` in the role `diagRoutineType` shall exist **`at the time when the RTE is generated`**.⌋*()*

Enumeration	DiagnosticRoutineTypeEnum
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumerator specifies the different types of diagnostic routines.
Aggregated by	DiagnosticRoutineNeeds.diagRoutineType
Literal	**Description**
asynchronous	This indicates that the diagnostic server is not blocked while the diagnostic routine is running. **Tags:**atp.EnumerationLiteralIndex=0
synchronous	This indicates that the diagnostic routine blocks the diagnostic server in the ECU while the routine is running. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.35: DiagnosticRoutineTypeEnum

The meta-class DiagnosticIoControlNeeds is used to define requirements to configure the Diagnostic Communication Manager Service. The PPortPrototype corresponds to the diagnostic service InputOutputControlByIdentifier. Within the SwcInternalBehavior up to three RunnableEntitys are defined for implementing the ClientServerOperations mentioned before.

Class	DiagnosticIoControlNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
currentValue	DiagnosticValueNeeds	0..1	ref	Reference to the DiagnosticValueNeeds indicating the access to the current value via signalBasedDiagnostics.
freezeCurrent StateSupported	Boolean	0..1	attr	This attribute determines, if the referenced port supports temporary freezing of I/O value.
resetToDefault Supported	Boolean	0..1	attr	This represents a flag for the existence of the ResetTo Default operation in the service interface.
shortTerm Adjustment Supported	Boolean	0..1	attr	This attribute determines, if the referenced port supports temporarily setting of I/O value to a specific value provided by the diagnostic tester.

Table 13.36: DiagnosticIoControlNeeds

The meta-class DiagnosticValueNeeds is used to define requirements in order to configure the Diagnostic Communication Manager Service as well as the Diagnostic Event Manager Service.

The DCM can access either local values via a ClientServerInterface or it may access dataElements in a PPortPrototype typed by a SenderReceiverInterface. For this purpose, the DiagnosticValueNeeds require associations to local values (i.e. inside InternalBehavior) or respectively dataElements.

The attribute DiagnosticValueNeeds.diagnosticValueAccess of type DiagnosticValueAccessEnum allows for distinguishing between current values to read diagnostic information (readOnly) and data elements which are additionally classified as configurable (readWrite).

[constr_1363] Existence of attributes of DiagnosticValueNeeds ⌈if DiagnosticValueNeeds is aggregated by a SwcServiceDependency in the role serviceNeeds then the attributes

- DiagnosticValueNeeds.diagnosticValueAccess
- DiagnosticValueNeeds.dataLength

shall **not** exist **at any time in the workflow**.⌋*()*

[constr_1364] Existence of attributes of DiagnosticIoControlNeeds ⌈if DiagnosticIoControlNeeds is aggregated by a SwcServiceDependency in the role serviceNeeds then the attributes

- DiagnosticIoControlNeeds.freezeCurrentStateSupported
- DiagnosticIoControlNeeds.shortTermAdjustmentSupported

shall **not** exist **at any time in the workflow**.⌋*()*

For all intents and purposes, the statement made by [constr_1363] and [constr_1364] boils down to the fact that these attributes can only be reasonably used in the context of a BswServiceDependency.

Class	**DiagnosticValueNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the DCM which are not related to a particular item. In the case of using a sender receiver communicated value, the related value shall be taken via assigned Data in the role "signalBasedDiagnostics". In case of using a client/server communicated value, the related value shall be communicated via the port referenced by assignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the related software specifications (SWS).			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the DiagnosticValue Needs is aggregated within a BswModuleDependency. This attribute represents the length of data (in bytes) provided for this particular PID signal.
diagnosticValue Access	DiagnosticValueAccess Enum	0..1	attr	This attribute is applicable only if the DiagnosticValue Needs is aggregated within a BswModuleDependency. This attribute controls whether the data can be read and written or whether it is to be handled read-only.
fixedLength	Boolean	0..1	attr	This attribute is applicable only if the DiagnosticValue Needs is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.

▽

△

Class	DiagnosticValueNeeds			
processingStyle	DiagnosticProcessingStyleEnum	0..1	attr	This attribute controls whether interaction requires the software-component to react synchronously on a request or whether it processes the request in background but still the DCM has to issue the call again to eventually obtain the result of the request.

Table 13.37: DiagnosticValueNeeds

Enumeration	DiagnosticValueAccessEnum
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Defines the access of the configured diagnostic current values which will be used by the Dem or Dcm module.
Aggregated by	DiagnosticValueNeeds.diagnosticValueAccess
Literal	Description
readOnly	The access to the data element is limited to read-only. This is typically used to read-out diagnostic information (e.g. current values). **Tags:**atp.EnumerationLiteralIndex=0
readWrite	The value of the diagnostic data element is classified as configurable (read and write access is possible). **Tags:**atp.EnumerationLiteralIndex=1
writeOnly	The access to the data element is limited to write-only. This supports the use case where the Dcm just writes data to the application software without the intention to read it back, **Tags:**atp.EnumerationLiteralIndex=2

Table 13.38: DiagnosticValueAccessEnum

Enumeration	DiagnosticProcessingStyleEnum
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This meta-class represents the ability to define the processing style of diagnostic requests.
Aggregated by	DiagnosticValueNeeds.processingStyle
Literal	Description
processingStyleAsynchronous	The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request. **Tags:**atp.EnumerationLiteralIndex=0
processingStyleAsynchronousWithError	The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request or handle error code. **Tags:**atp.EnumerationLiteralIndex=1
processingStyleSynchronous	The software-component is supposed to react synchronously on the request. **Tags:**atp.EnumerationLiteralIndex=2

Table 13.39: DiagnosticProcessingStyleEnum

Class	DiagnosticsCommunicationSecurityNeeds
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This meta-class represents the needs of a software-component to verify the access to security level via diagnostic services.
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*

▽

△

Class	DiagnosticsCommunicationSecurityNeeds			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.40: DiagnosticsCommunicationSecurityNeeds

13.8.4.1 Dcm Service Use Case: read/write current values by Client Server Interface

Scenario: an AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write current value via diagnostic services (e.g. measurements, variant coding)

[TPS_SWCT_02002] AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write current value via diagnostic services
⌈

ServiceNeeds kind DiagnosticValueNeeds

RoleBasedPortAssignment valid roles:

- DataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01628] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (DataServices_{Data}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00686].

13.8.4.2 Dcm Service Use Case: read/write current values of specific DID by Client Server Interface

Scenario: an AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write current values via diagnostic services (e.g. measurements, variant coding) where the applicable DID is passed as an argument to the access functions. This use case applies mostly if the software-component provides the information related to more than one DID.

[TPS_SWCT_01639] AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write current value via diagnostic services where the applicable DID is passed as an argument to the access functions ⌈

ServiceNeeds kind DiagnosticValueNeeds

RoleBasedPortAssignment valid roles:

- DataServices_DIDRange [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01640] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (DataServices_DIDRange_{Range}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00769].

13.8.4.3 Dcm Service Use Case: read/write current values by Sender Receiver Interface or Nv Data Interface

Scenario: an AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces or NvDataInterfaces to read/write current values via diagnostic services (e.g. measurements, variant coding) that correspond to elements of a DID (as opposed to the entire DID at once, which is described in [TPS_SWCT_01853]).

This is mainly used for data which are available at ports anyhow used for other communication purpose.

Note: this scenario can be implemented as a regular sender/receiver communication without the necessity to use a SwcServiceDependency. The description of a SwcServiceDependency (even if it is technically not required) may help to advertise the special role of the corresponding dataElement with respect to diagnostics.

[TPS_SWCT_02003] AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces or NvDataInterfaces to read/write current values via diagnostic services ⌈

ServiceNeeds kind DiagnosticValueNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- signalBasedDiagnostics [1..2]

RepresentedPortGroups
N/A

To read the signal the AtomicSwComponentType shall offer an AbstractProvidedPortPrototype, to write the signal the AtomicSwComponentType shall offer an AbstractRequiredPortPrototype.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [TPS_SWCT_01579], [TPS_SWCT_01831] and [SWS_Dcm_00687].

The service interface used for this purpose is DataServices_{Data}, see also [SWS_Dcm_00687]. For more information please refer to [TPS_SWCT_01579], [TPS_SWCT_01831].

[constr_1679] Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = signalBasedDiagnostics ⌈If the attribute RoleBasedDataAssignment.role is set to the value signalBasedDiagnostics then the reference RoleBasedDataAssignment.usedDataElement.localVariable shall not exist **at the time when the RTE is generated**.⌋*()*

For explanation of the existence of [constr_1679], it is not intended to provide diagnostic access to local variables inside the SwcInternalBehavior.

13.8.4.4 Dcm Service Use Case: read/write entire DID by Sender Receiver Interface or Nv Data Interface

Scenario: an AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces or NvDataInterfaces to read/write the content of an entire DID via diagnostic services.

[TPS_SWCT_01853] AtomicSwComponentType offers PortPrototype typed by SenderReceiverInterface or NvDataInterfaces to read/write the content of an entire DID via diagnostic services ⌈

ServiceNeeds kind DiagnosticValueNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- signalBasedDiagnostics [1..2]

RepresentedPortGroups
N/A

To read the data, the AtomicSwComponentType shall offer an AbstractProvidedPortPrototype. To write the data, the AtomicSwComponentType shall offer an AbstractRequiredPortPrototype.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

The service interface used for this purpose is DataServices_{Data}, see also [SWS_Dcm_91057]. For more information please refer to [TPS_SWCT_01579], [TPS_SWCT_01831].

[constr_1679] applies.

13.8.4.5 Dcm Service Use Case: start/stop or request routine results

Scenario: an AtomicSwComponentType offers a PortPrototype typed by a ClientServerInterface to start/stop or request routine results of diagnostic routines.

[TPS_SWCT_02004] AtomicSwComponentType offers a PortPrototype typed by a ClientServerInterface to start/stop or request routine results of diagnostic routines ⌈

ServiceNeeds kind DiagnosticRoutineNeeds

RoleBasedPortAssignment valid roles:

- RoutineServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01632] Suffix used for the resulting name of the PortInterface for the Routine Services ⌈The *suffix* used for the resulting name of the PortInterface for the Routine Services (RoutineServices_{RoutineName}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00690].

[constr_1724] Usage of attribute ClientServerOperation.diagArgIntegrity ⌈With the exception of the context of a ServiceSwComponentType, the attribute ClientServerOperation.diagArgIntegrity shall only have the value true if the ClientServerInterface containing the respective ClientServerOperation is used to type a PPortPrototype that is referenced by a RoleBasedPortAssignment aggregated by a SwcServiceDependency that in turn aggregates DiagnosticRoutineNeeds.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

In other words, on the level of application software the attribute diagArgIntegrity shall only exist in the context of a PPortPrototype.

Obviously, the ServiceSwComponentType that is used to represent the Dcm will use an RPortPrototype for the other end of the AssemblySwConnector that connects to said PortPrototype in the application software.

13.8.4.6 Dcm Service Use Case: IO control by Client Server Interface

Scenario: an AtomicSwComponentType offers a PortPrototype typed by a ClientServerInterface to adjust the IO signal via diagnostic services.

[TPS_SWCT_02005] AtomicSwComponentType offers PortPrototypes typed by ClientServerInterfaces to adjust the IO signal via diagnostic services ⌈

ServiceNeeds kind DiagnosticIoControlNeeds

RoleBasedPortAssignment valid roles:

- DataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01629] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (DataServices_{Data}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00686].

13.8.4.7 Dcm Service Use Case: IO control by Sender Receiver Interface

This use case represents an alternative to the use case described in chapter 13.8.4.6, i.e. for the same purpose it is also possible to utilize a SenderReceiverInterface.

The essential idea behind the existence of I/O PortPrototypes typed by SenderReceiverInterface is the possibility to have quick access to the dataElements currently under control.

Especially cases where access to dataElements is required from different partitions (for example in multi core systems) can benefit from this approach.

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a SenderReceiverInterface (in particular: IOControlRequest) to adjust the I/O signal via diagnostic services and offers a PPortPrototype typed by a SenderReceiverInterface (in particular: IOControlResponse) to provide the IO "operation response".

In case of using IOControlRequest (which owns **three** dataElements) and IOControlResponse the whole PortPrototype is related to **exactly one** IO control and needs to be consistent.

Therefore, the usage of RoleBasedPortAssignment (instead of the RoleBasedDataAssignment, which would otherwise typically be used for a sender/receiver-based scenario) is required for avoiding modeling overhead.

[TPS_SWCT_01654] AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces to adjust the IO signal via diagnostic services
⌈

ServiceNeeds kind DiagnosticIoControlNeeds

RoleBasedPortAssignment valid roles:

- IOControlRequest [1]
- IOControlResponse [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

The IOControl service requires in its diagnostic response the current value of the IO-DID, which is identical to the current value represented by DiagnosticValueNeeds of the ReadDataByIdentifer response.

[TPS_SWCT_01655] Reference from DiagnosticIoControlNeeds to DiagnosticValueNeeds ⌈In the scenario described by [TPS_SWCT_01654], the DiagnosticIoControlNeeds shall reference the DiagnosticValueNeeds which relates to the access of the current value via diagnostic services (see [TPS_SWCT_02003]).⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01656] Suffix used for the resulting name of the PortInterface for IOControlRequest and IOControlResponse ⌈The *suffix* used for the resulting name of the PortInterface for the IOControlRequest_{Data} and IOControlResponse_{Data} shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

The service use case is visualized in Figure 13.17. The SwComponentPrototype contains two SwcServiceDependencys, one for the I/O Control, and one for the access of the dataElement with the shortName “IOx” by the Dcm.

Please note that, in this example, the SenderReceiverInterface used on the PPortPrototype of the ApplicationSwComponentType has several dataElements (where the dataElement with the shortName “IOx” is one of them). This is a perfectly valid configuration.

On the other hand, the SenderReceiverInterface used on the RPortPrototype of the ServiceSwComponentType representing the Dcm can only have **one** dataElement. This single dataElement shall (as far as the example is concerned) be given the shortName “IOx”.

Note the reference from the DiagnosticIoControlNeeds to the DiagnosticValueNeeds. this reference explicitly expresses that access to a DID is combined with the usage of I/O control.

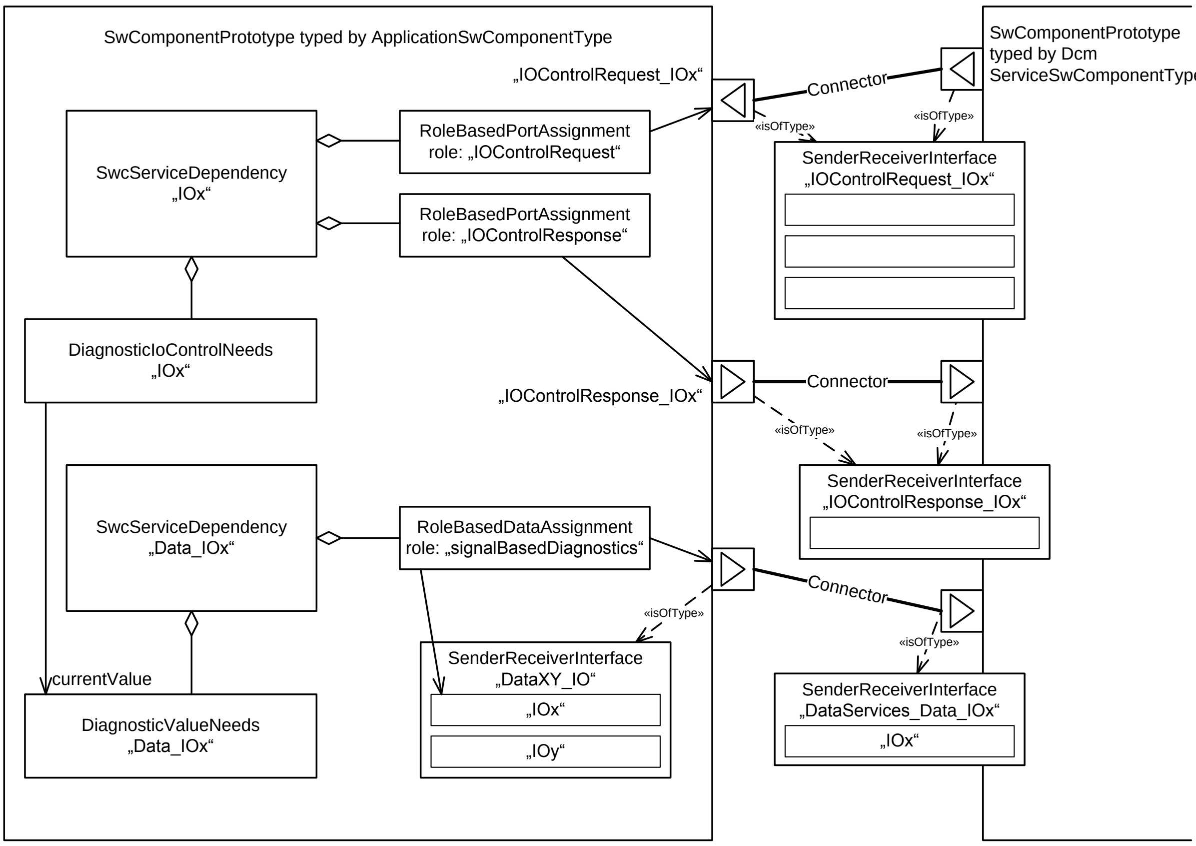


Figure 13.17: Visualization of the service use case

[TPS_SWCT_01657] NamingRule for RPortPrototype referenced by a RoleBasedPortAssignment with attribute role set to “IOControlRequest” ⌈The shortName of a RPortPrototype referenced by a RoleBasedPortAssignment with attribute role set to “IOControlRequest” shall be created by concatenating the prefix “IOControlRequest” and the SwcServiceDependency.shortName, separated by a single underscore character (i.e. “_”).⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_01308] and [SWS_Dcm_01309].

[constr_1741] Restriction to explicit sending semantics for the usage of DataServices in the context of a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds ⌈A dataElement

- that is referenced by a RoleBasedDataAssignment (where the attribute role is set to signalBasedDiagnostics) owned by a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds
- **shall also be referenced** by a VariableAccess aggregated in the role dataSendPoint by a given RunnableEntity that in turn belongs to the enclosing SwcInternalBehavior.
- **shall not be referenced** by a VariableAccess aggregated in the role dataWriteAccess by a given RunnableEntity that in turn belongs to the enclosing SwcInternalBehavior.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

In other words, the data handled by the diagnostics transformer shall only be sent explicitly. The usage of **implicit sending of these data is not supported**.

13.8.4.8 Dcm Service Use Case: Access to protocol, session and security information

Scenario: an AtomicSwComponentType offers a server port to get protocol, session and security information or to request a Reset to Default Session.

[TPS_SWCT_02013] AtomicSwComponentType offers a server port to get protocol, session and security information or to request a Reset to Default Session ⌈

ServiceNeeds kind DiagnosticCommunicationManagerNeeds

RoleBasedPortAssignment valid roles:

- DCMServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00698]

13.8.4.9 Dcm Service Use Case: Verify the access to security level

Scenario: an AtomicSwComponentType provides a server port to verify the access to security level via diagnostic services.

[TPS_SWCT_02015] AtomicSwComponentType verifies the access to security level via diagnostic services ⌈

ServiceNeeds kind DiagnosticsCommunicationSecurityNeeds

RoleBasedPortAssignment valid roles:

- SecurityAccess [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01627] Suffix used for the resulting name of the PortInterface for the Security Access ⌈The *suffix* used for the resulting name of the PortInterface for the Security Access (SecurityAccess_{SecurityLevel}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00685]

13.8.4.10 Dcm Service Use Case: multiple testers access one ECU

Scenario: an AtomicSwComponentType provides a server port to get information on the status of the protocol communication. Further on the AtomicSwComponentType may disallow a protocol.

[TPS_SWCT_02016] AtomicSwComponentType requires information on the status of the protocol communication and may disallow a protocol ⌈

ServiceNeeds kind DiagnosticCommunicationManagerNeeds

RoleBasedPortAssignment valid roles:

- CallbackDCMRequestServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00170*, *RS_SWCT_03190*)

For more information please refer to [SWS_Dcm_00692]

13.8.4.11 Dcm Service Use Case: Service Request Notification

Scenario: an `AtomicSwComponentType` provides a server port to get notified about a Service Request via diagnostic services. This indicates the successful reception of a new request to application.

Within this Service Request Notification this function application can examine the permission of the diagnostic service / environment.

Please note that the Service Request Notification can be used in two characteristics, i.e. as *manufacturer* ([TPS_SWCT_01577] applies) or as a *supplier* ([TPS_SWCT_01578] applies).

[TPS_SWCT_01577] **`AtomicSwComponentType` requires the notification about a Service Request via diagnostic services with *manufacturer* characteristics** ⌈

The attribute `DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType` shall be set to the value `requestCallbackTypeManufacturer`.

ServiceNeeds kind `DiagnosticCommunicationManagerNeeds`

RoleBasedPortAssignment valid roles:

- `ServiceRequestNotification` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_03190*)

[TPS_SWCT_01578] **`AtomicSwComponentType` requires the notification about a Service Request via diagnostic services with *supplier* characteristics** ⌈

The attribute `DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType` shall be set to the value `requestCallbackTypeSupplier`.

ServiceNeeds kind `DiagnosticCommunicationManagerNeeds`

RoleBasedPortAssignment valid roles:

- `ServiceRequestNotification` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00694]

13.8.4.12 Dcm Service Use Case: read/write and IOCtrl current values by Client Server Interface

Scenario: an AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write and IOCtrl current value via diagnostic services (e.g. measurements, variant coding)

[TPS_SWCT_01690] AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write and IOCtrl current value via diagnostic services ⌈

ServiceNeeds kind DiagnosticValueNeeds, DiagnosticIoControlNeeds

RoleBasedPortAssignment valid roles:

- DataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01691] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The suffix used for the resulting name of the PortInterface for the Data Services (DataServices_{Data}) shall be taken from the shortName of the SwcServiceDependency that aggregates the DiagnosticIoControlNeeds.⌋ *(RS_SWCT_00170, RS_SWCT_03190)*

13.8.4.13 Dcm Service Use Case: A software-component acts as a "file server" to a diagnostic tester

Scenario: an AtomicSwComponentType acts as a "file server" to a diagnostic tester.

[TPS_SWCT_01791] AtomicSwComponentType acts as a "file server" to a diagnostic tester ⌈

ServiceNeeds kind DiagnosticRequestFileTransferNeeds

RoleBasedPortAssignment valid roles:

- RequestFileTransfer [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Class	**DiagnosticRequestFileTransferNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class indicates the existence of a service use case that involves UDS service 0x38, Request File Transfer.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.41: DiagnosticRequestFileTransferNeeds

13.8.5 OBD related Needs

The ObdRatioServiceNeeds describes further properties of the implementation of the Rate Based Monitoring (e.g. connectionType) as well as the logical dependencies relevant for the ECU configuration.

Class	**ObdRatioServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular "ratio monitoring" which is supported by this component or module.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
connectionType	ObdRatioConnection KindEnum	0..1	attr	Defines how the DEM is connected to the component or module to perform the IUMPR (In use monitor performance ratio) service.
rateBased MonitoredEvent	DiagnosticEventNeeds	0..1	ref	The rate based monitored Diagnostic Event.
usedFid	FunctionInhibitionNeeds	0..1	ref	This represents the primary Function Inhibition Identifier used for the rate based monitor. This is an optional attribute.

Table 13.42: ObdRatioServiceNeeds

Class	**ObdControlServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Service 08 (request control of on-board system) in relation to a particular test-Identifier (TID) supported by this component or module.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.43: ObdControlServiceNeeds

Enumeration	**ObdRatioConnectionKindEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Defines the way how the IUMPR service connection between the Dem and the client component or module is handled (for details see the DEM Specification).
Aggregated by	ObdRatioServiceNeeds.connectionType
Literal	***Description***
apiUse	The IUMPR service (of the DEM) uses an explicit API to connect to the component or module. **Tags:**atp.EnumerationLiteralIndex=0
observer	The IUMPR service (of the Dem) uses no API but "observes" the associated diagnostic event. **Tags:**atp.EnumerationLiteralIndex=1

Table 13.44: ObdRatioConnectionKindEnum

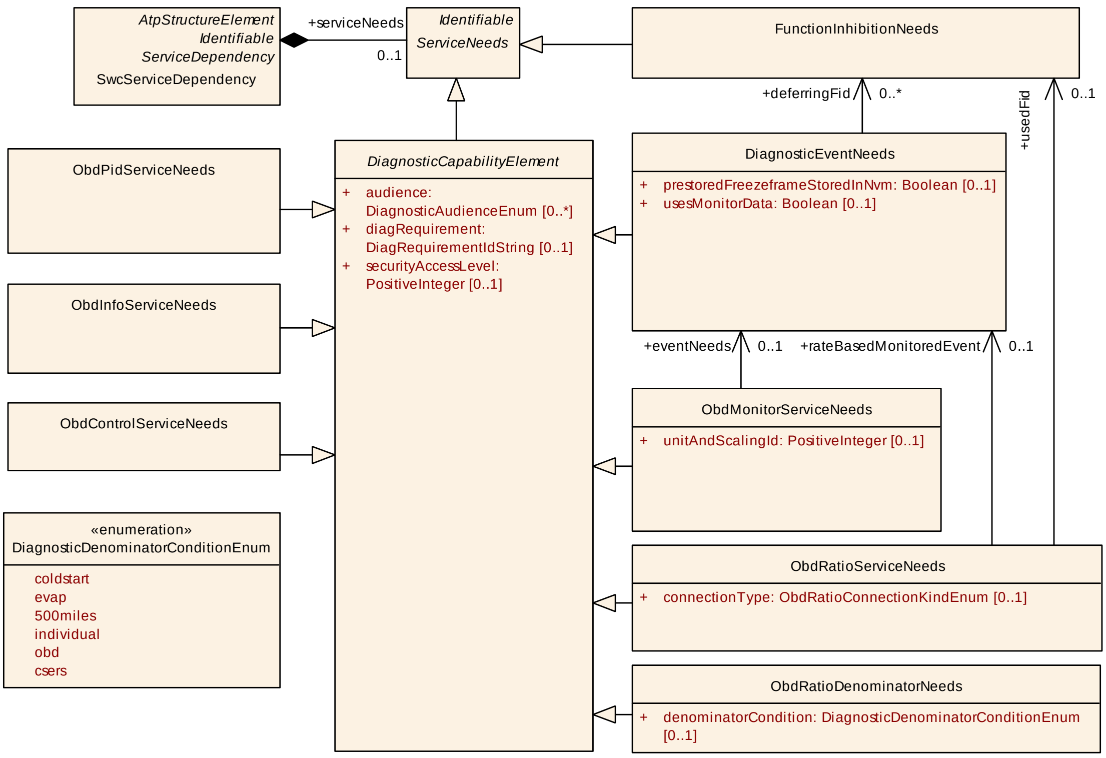


Figure 13.18: ServiceNeeds: Diagnostic-related ServiceNeeds with emphasis on OBD

In addition, ObdPidServiceNeeds, ObdInfoServiceNeeds, ObdMonitorServiceNeeds and ObdControlServiceNeeds are required in order to specify the specific needs for OBD diagnostic service calls. Note that ObdPidServiceNeeds is used for the Diagnostic Event Manager as well.

[constr_1520] Semantics of ObdRatioServiceNeeds.rateBasedMonitoredEvent ⌈In the context of an SwcServiceDependency, each DiagnosticEventNeeds referenced in the role rateBasedMonitoredEvent shall only be referenced by at most a single ObdRatioServiceNeeds **at the time when the RTE is generated**.⌋*()*

Class	**ObdPidServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular PID (parameter identifier) which is supported by this component or module. In case of using a client/server communicated value, the related value shall be communicated via the port referenced by assignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the related software specifications (SWS).			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.45: ObdPidServiceNeeds

Class	**ObdInfoServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a given InfoType (OBD Service 09) which is supported by this component or module.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.46: ObdInfoServiceNeeds

Class	**ObdMonitorServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular on-board monitoring test supported by this component or module. (OBD Service 06).			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
applicationDataType	ApplicationDataType	0..1	ref	reference to an ApplicationDataType that describes the scaling of the data reported by the software-component to the Dem.

▽

△

Class	ObdMonitorServiceNeeds			
eventNeeds	DiagnosticEventNeeds	0..1	ref	This reference identifies the corresponding diagnostic event.
unitAndScalingId	PositiveInteger	0..1	attr	Unit and scaling ID according to ISO 15031-5.

Table 13.47: ObdMonitorServiceNeeds

13.8.5.1 Dem Service Use Case: In-Use-Monitor Performance Ratio calculation

Scenario: an AtomicSwComponentType implements a OBD system monitor with In-Use-Monitor Performance Ratio (IUMPR) and offers client ports to provide the capability to define the number of times a fault could have been found.

[TPS_SWCT_02007] AtomicSwComponentType implements a OBD system monitor with In-Use-Monitor Performance Ratio ⌈

ServiceNeeds kind ObdRatioServiceNeeds

RoleBasedPortAssignment valid roles:

- IUMPRNumerator [0..1]
- IUMPRDenominator [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00610] and [SWS_Dem_00611].

[constr_2053] Consistency between role IUMPRNumerator and ObdRatioServiceNeeds.connectionType ⌈If a SwcServiceDependency with a ObdRatioServiceNeeds is defined and the attribute connectionType of the contained ObdRatioServiceNeeds is set to ObdRatioConnectionKindEnum.apiUse, a RoleBasedPortAssignment with the role value IUMPRNumerator shall be defined.

If the attribute connectionType of the contained ObdRatioServiceNeeds is set to ObdRatioConnectionKindEnum.observer, the role value IUMPRNumerator is not applicable.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

13.8.5.2 Dcm Service Use Case: read parameter identifier via diagnostic services by Client Server Interface

Scenario: an AtomicSwComponentType offers a server port to read/write current value via OBD services.

[TPS_SWCT_02008] AtomicSwComponentType offers a server port to read/write current value via OBD services ⌈

ServiceNeeds kind ObdPidServiceNeeds

RoleBasedPortAssignment
The following roles are applicable:

- DataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01630] Suffix used for the resulting name of the PortInterface for the Data Services ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (DataServices_{Data}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00686].

13.8.5.3 Dcm Service Use Case: read parameter identifier via diagnostic services by Sender Receiver Interface

Scenario: an AtomicSwComponentType offers sender receiver ports to read/write current values via OBD services.

[TPS_SWCT_02009] AtomicSwComponentType offers sender receiver ports to read/write current values via OBD services ⌈

ServiceNeeds kind ObdPidServiceNeeds

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment
The following roles are applicable:

- signalBasedDiagnostics [1..2]

RepresentedPortGroups
N/A

To read the signal the AtomicSwComponentType shall offer an AbstractProvidedPortPrototype, to write the signal the AtomicSwComponentType shall offer an AbstractRequiredPortPrototype.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00687].

Please note that [constr_1679] applies for this service use case.

13.8.5.4 Dcm Service Use Case: Request vehicle information

Scenario: an AtomicSwComponentType offers a server port to read vehicle information values via OBD services.

[TPS_SWCT_02010] AtomicSwComponentType offers a server port to read vehicle information values via OBD services ⌈

ServiceNeeds kind ObdInfoServiceNeeds

RoleBasedPortAssignment valid roles:

- InfotypeServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

[TPS_SWCT_01631] Suffix used for the resulting name of the PortInterface for the Infotype Services ⌈The *suffix* used for the resulting name of the PortInterface for the Infotype Services (InfotypeServices_{VehInfoData}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dcm_00688].

13.8.5.5 Dem Service Use Case: Read DTR data from SW-C for OBD Service $06

Scenario: an AtomicSwComponentType exposes a client port to read DTR value.

[TPS_SWCT_02011] AtomicSwComponentType offers a client port to read DTR value ⌈

ServiceNeeds kind ObdMonitorServiceNeeds

RoleBasedPortAssignment valid roles:

- DTRCentralReport [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00170, RS_SWCT_03190*)

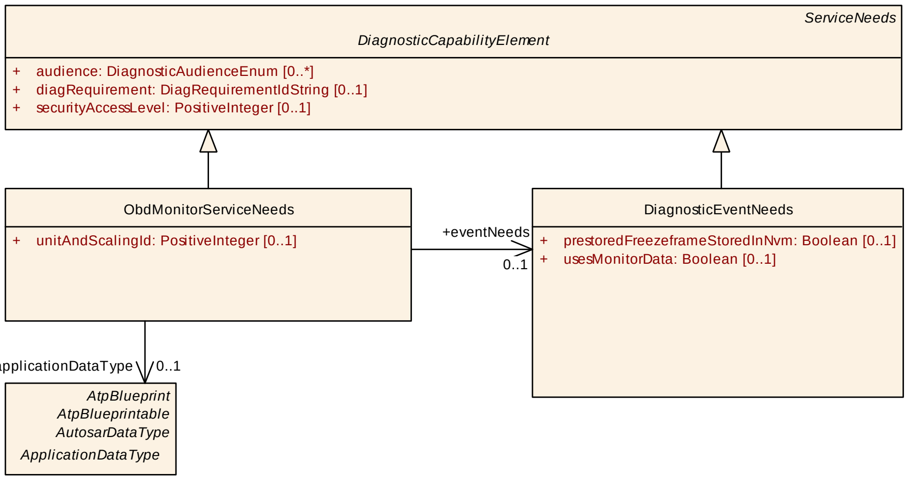


Figure 13.19: Modeling of `ObdMonitorServiceNeeds`

13.8.5.6 Dcm Service Use Case: request control of on-board system, test or component

Scenario: an `AtomicSwComponentType` offers a server port for request control of on-board system, test or component via OBD services.

[TPS_SWCT_02012] `AtomicSwComponentType` offers a server port for request control of on-board system, test or component via OBD services ⌈

ServiceNeeds kind `ObdControlServiceNeeds`

RoleBasedPortAssignment
The following roles are applicable:

- `RequestControlServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00170, RS_SWCT_03190*)

[TPS_SWCT_01633] Suffix used for the resulting name of the `PortInterface` for the Request Control Services ⌈The *suffix* used for the resulting name of the `PortInterface` for the Request Control Services (RequestControlServices_{Tid})

shall be taken from the shortName of the applicable SwcServiceDependency.⌋
(RS_SWCT_00170, RS_SWCT_03190)

For more information please refer to [SWS_Dcm_00691].

13.8.5.7 Dem Service Use Case: In-Use-Monitoring Performance Ratio Denominator interface

Scenario: an AtomicSwComponentType implements a denominator (or accesses a ratio for transmission to other control units).

[TPS_SWCT_01765] Dem Service Use Case: In-Use-Monitoring Performance Ratio Denominator interface ⌈

ServiceNeeds kind ObdRatioDenominatorNeeds

RoleBasedPortAssignment
The following roles are applicable:

- IUMPRDenominatorCondition [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00170, RS_SWCT_03190)*

For more information please refer to [SWS_Dem_00742].

Class	**ObdRatioDenominatorNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class shall be used to indicate that a software-component wants to access the in-use-monitoring performance ration denominator.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
denominator Condition	DiagnosticDenominator ConditionEnum	0..1	attr	This attribute indicates the applicable denominator condition.

Table 13.48: ObdRatioDenominatorNeeds

Enumeration	**DiagnosticDenominatorConditionEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration contains valid denominator types.
Aggregated by	ObdRatioDenominatorNeeds.denominatorCondition
Literal	***Description***

▽

△

Enumeration	**DiagnosticDenominatorConditionEnum**
_500miles	Condition based on definition of 500miles conditions as defined for OBD2. **Tags:**atp.EnumerationLiteralIndex=2
coldstart	Condition based on definition of "cold start" as defined for EU5+ **Tags:**atp.EnumerationLiteralIndex=0
csers	Conditions based on the "Cold start emission reduction strategy" denominator **Tags:**atp.EnumerationLiteralIndex=5
evap	Condition based on definition of "EVAP" conditions as defined for OBD2. **Tags:**atp.EnumerationLiteralIndex=1
individual	condition based on definition of individual requirements. **Tags:**atp.EnumerationLiteralIndex=3
obd	Condition based on definition of OBD requirements. **Tags:**atp.EnumerationLiteralIndex=4

Table 13.49: DiagnosticDenominatorConditionEnum

13.8.6 Diagnostics over IP

This chapter describes the usage of specific meta-classes to support the specification of diagnostics over IP. For more details, please refer to ISO 13400 [44].

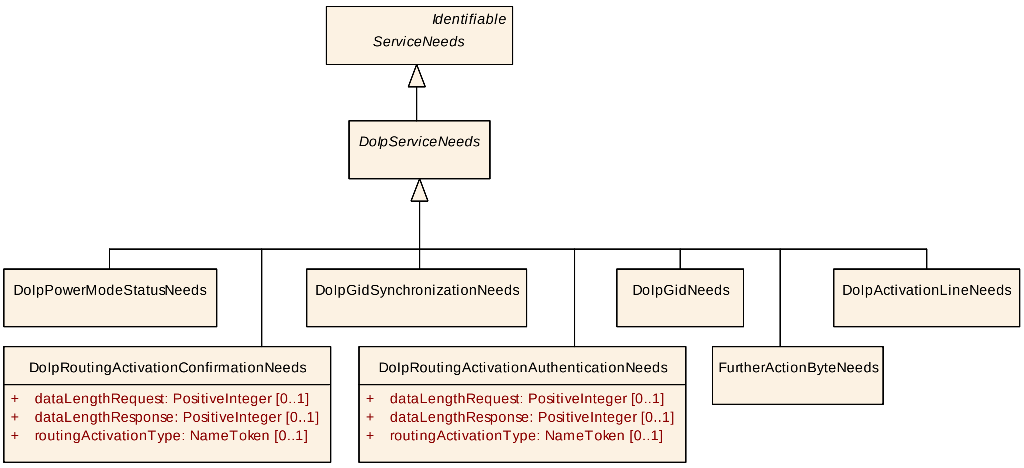


Figure 13.20: Subclasses of `ServiceNeeds` for implementing diagnostics over IP

Class	***DoIpServiceNeeds*** (abstract)
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This represents an abstract base class for ServiceNeeds related to DoIP.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*
Subclasses	DoIpActivationLineNeeds, DoIpGidNeeds, DoIpGidSynchronizationNeeds, DoIpPowerModeStatusNeeds, DoIpRoutingActivationAuthenticationNeeds, DoIpRoutingActivationConfirmationNeeds, FurtherActionByteNeeds

▽

△

Class	***DoIpServiceNeeds*** (abstract)			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.50: DoIpServiceNeeds

Class	DoIpGidNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The DoIpGidNeeds indicates that the software-component owning this ServiceNeeds is providing the GID number either after a GID Synchronisation or by other means like e.g. flashed EEPROM parameter. This need can be used independent from DoIpGidSynchronizationNeeds and is necessary if the GID can not be provided out of the DoIP configuration options.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.51: DoIpGidNeeds

Class	DoIpGidSynchronizationNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The DoIpGidSynchronizationNeeds indicates that the software-component owning this ServiceNeeds is triggered by the DoIP entity to start a synchronization of the GID (Group Identification) on the DoIP service 0x0001, 0x0002, 0x0003 or before announcement via service 0x0004 according to ISO 13400-2:2012 if necessary. Note that this need is only relevant for DoIP synchronization masters.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.52: DoIpGidSynchronizationNeeds

Class	DoIpPowerModeStatusNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The DoIpPowerModeStatusNeeds indicates that the software-component owning this ServiceNeeds is providing the PowerModeStatus for the DoIP service 0x4003 according to ISO 13400-2:2012.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.53: DoIpPowerModeStatusNeeds

Class	DoIpRoutingActivationAuthenticationNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	DoIPRoutingActivationAuthenticationNeeds indicates that the software-component owning this Service Needs will have an authentication required for a DoIP routing activation service (0x0005) according to ISO 13400-2:2012.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataLength Request	PositiveInteger	0..1	attr	Describes the length in byte of the additional information for RA authentication that is needed by the software entity. If the software entity is a software-component the attribute does not need to exist as the information is available via the length of the uint8 Array type. Otherwise (i.e the software entity is a Complex Driver) this attribute needs to be filled out if additional information is needed.
dataLength Response	PositiveInteger	0..1	attr	Describes the length in byte of the additional information for RA authentication that is provided by the software entity. If the software entity is a software-component the attribute does not need to exist as the information is available via the length of the uint8 Array type. Otherwise (i.e the software entity is a Complex Driver) this attribute needs to be filled in if additional information is provided.
routing ActivationType	NameToken	0..1	attr	Describes the ISO 13400-2:2012 "routing activation request activation type" which is received via DoIP service 0x0005. 0x00 is DEFAULT, 0x01 is WWH-OBD. If neither of the specified values (0x00 or 0x01) is needed the token shall contain RA_ + hex value representation of the integer value shall be used (i.e: RA_0xE1).

Table 13.54: DoIpRoutingActivationAuthenticationNeeds

Class	DoIpRoutingActivationConfirmationNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	DoIpRoutingActivationConfirmationNeeds indicates that the software-component that owns this Service Needs will have a confirmation required for a DoIP routing activation service (0x0005) according to ISO 13400-2:2012.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataLength Request	PositiveInteger	0..1	attr	Describes the length in byte of the additional information for RA confirmation that is needed by the software entity. If the software entity is a software-component the attribute does not need to exist as the information is available via the length of the uint8 Array type. Otherwise (i.e the software entity is a Complex Driver) this attribute needs to be filled out if additional information is needed.
dataLength Response	PositiveInteger	0..1	attr	Describes the length in byte of the additional information for RA confirmation that is provided by the software entity. If the software entity is a software-component the attribute does not need to exist as the information is available via the length of the uint8 Array type. Otherwise (i.e the software entity is a Complex Driver) this attribute needs to be filled out if additional information is provided.

▽

△

Class	DoIpRoutingActivationConfirmationNeeds			
routing ActivationType	NameToken	0..1	attr	Describes the ISO 13400-2:2012 "routing activation request activation type" which is received via DoIP service 0x0005. 0x00 is DEFAULT, 0x01 is WWH-OBD. If neither of the specified values (0x00 or 0x01) is needed the token shall contain RA_ + hex value representation of the integer value shall be used (i.e: RA_0xE1).

Table 13.55: DoIpRoutingActivationConfirmationNeeds

Class	DoIpActivationLineNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	A DoIP entity needs to be informed when an external tester is attached or activated. The DoIpActivationServiceNeeds specifies the trigger for such an event. Examples would be a Pdu via a regular communication bus, a PWM signal, or an I/O. For details please refer to the ISO 13400.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.56: DoIpActivationLineNeeds

13.8.6.1 DoIP Service Use Case: GID synchronization can be necessary if the ECU is DoIP Gid synchronization master

Scenario: on the event of connecting a tester to an ECU a GID synchronization can be necessary if the ECU is DoIP Gid synchronization master. In this case, it is necessary to define a `DoIpGidSynchronizationNeeds`.

[TPS_SWCT_01537] GID synchronization can be necessary if the ECU is DoIP Gid synchronization master ⌈

ServiceNeeds kind `DoIpGidSynchronizationNeeds`

RoleBasedPortAssignment valid roles:

- `CallbackTriggerGIDSynchronization` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03310, RS_SWCT_03190)*

13.8.6.2 DoIP Service Use Case: Vehicle information is broadcast or can be requested by the tester

Scenario: vehicle information is broadcast or can be requested by the tester. In this case, it is necessary to define a DoIpGidNeeds.

[TPS_SWCT_01538] Vehicle information is broadcast or can be requested by the tester ⌈

ServiceNeeds kind DoIpGidNeeds

RoleBasedPortAssignment valid roles:

- CallbackGetGID [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03310, RS_SWCT_03190)*

13.8.6.3 DoIP Service Use Case: Tester could also request the power status with respect to diagnostics

Scenario: before starting the diagnostics processing for the DoIP entity or sub-networks connected via DoIP, the tester could also request the power status with respect to diagnostics. To support this option it will be necessary to define a DoIpPowerModeStatusNeeds.

[TPS_SWCT_01539] Tester can also request before starting diagnostic processing for the DoIP entity or sub-networks connected via DoIP the power status with respect to diagnostics ⌈

ServiceNeeds kind DoIpPowerModeStatusNeeds

RoleBasedPortAssignment valid roles:

- CallbackGetPowerMode [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03310, RS_SWCT_03190)*

13.8.6.4 DoIP Service Use Case: Routing activation mechanism is used which can lead to additional impact regarding authentication or confirmation

Scenario: to enable diagnostics of the tester to a different target address, the routing activation mechanism is used which can lead to additional impact regarding authentication or confirmation. Here, the definition of DoIpRoutingActivationAuthenticationNeeds and/or DoIpRoutingActivationConfirmationNeeds would be applicable.

[TPS_SWCT_01544] *prefix* used for the actual name of the used PortInterface for the routing activation ⌈The *prefix* used for the actual name of the used PortInterface for the routing activation shall be taken from the shortName of the enclosing SwcServiceDependency.⌋*(RS_SWCT_03310, RS_SWCT_03190)*

[TPS_SWCT_01540] Routing activation mechanism is used which can lead to additional impact regarding authentication or confirmation ⌈

ServiceNeeds kind

- DoIpRoutingActivationAuthenticationNeeds [0..1]
- DoIpRoutingActivationConfirmationNeeds [0..1]

RoleBasedPortAssignment valid roles:

- RoutingActivation [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03190)*

13.8.6.5 DoIP Service Use Case: a DoIP entity needs to be informed when an external tester is attached or activated.

Scenario: to enable diagnostics by connecting a tester to an ECU it is necessary that the application software becomes aware of the tester's presence.

For this purpose, the applicable ServiceSwComponentType is supposed to provide a PPortPrototype typed by the ModeSwitchInterface named DoIPActivationLineStatus towards the application.

To trigger the existence of the PPortPrototype, DoIpActivationLineNeeds shall be defined.

[TPS_SWCT_01546] Notification when an external tester is attached or activated ⌈

ServiceNeeds kind DoIpActivationLineNeeds

RoleBasedPortAssignment valid roles:

- DoIPActivationLineStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_03310, RS_SWCT_03190*)

13.8.6.6 Service Use Case: Set and reset Warning Indicator Request bit

Scenario: In some cases (e.g. controlling a failsafe reaction in application) the “Warning Indicator Request”-bit of a corresponding event in Dem shall be set/reset by a special “failsafe software-component”.

The failsafe software-component has to ensure a proper status of the “Warning Indicator Request”-bit (e.g. regarding ISO14229-1 or manufacture specific requirements).

Therefore, the failsafe SW-C can use existing Dem mechanism to get the information about status changes of events in Dem (e.g. Callback EventStatusChanged).

For this purpose, the applicable ServiceSwComponentType is supposed to provide a PPortPrototype typed by the ClientServerInterface named EventStatus towards the application.

To trigger the existence of the PPortPrototype, WarningIndicatorRequestedBitNeeds shall be defined.

[TPS_SWCT_01547] Ability to set and reset the Warning Indicator Request bit ⌈

ServiceNeeds kind WarningIndicatorRequestedBitNeeds

RoleBasedPortAssignment valid roles:

- EventStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_03310, RS_SWCT_03190*)

Class	WarningIndicatorRequestedBitNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to explicitly request the existence of the WarningIndicator RequestedBit.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.57: WarningIndicatorRequestedBitNeeds

13.8.6.7 DoIP Service Use Case: Atomic Software-Component provides the further action byte to the DoIP Service Component

Scenario: An `AtomicSwComponentType` provides the "further action byte" used in vehicle identification/announcement message.

[TPS_SWCT_01746] Atomic Software-Component provides the further action byte to the DoIP Service Component ⌈

ServiceNeeds kind `FurtherActionByteNeeds`

RoleBasedPortAssignment valid roles:

- `CallbackGetFurtherActionByte` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03310, RS_SWCT_03190)*

Class	FurtherActionByteNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The FurtherActionByteNeeds indicates that the software-component is able to provide the "further action byte" to the DoIp Service Component.			
Base	*ARObject*, *DoIpServiceNeeds*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.58: FurtherActionByteNeeds

13.8.7 Miscellaneous Diagnostic Service Use-Cases

13.8.7.1 Dcm Service Use Case: DiagnosticSessionControl

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the diagnostic session.

[TPS_SWCT_01706] AtomicSwComponentType supports *DiagnosticSessionControl* to get informed about the diagnostic session ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_DiagnosticSessionControlModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_DiagnosticSessionControlModeSwitchInterface is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

Class	**DiagnosticControlNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class indicates a service use-case for reporting the controlled status by diagnostic services.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.59: DiagnosticControlNeeds

13.8.7.2 Dcm Service Use Case: EcuReset

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the current status of *EcuReset* service.

[TPS_SWCT_01707] AtomicSwComponentType supports *EcuReset* service via diagnostic services ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_EcuResetModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_EcuResetModeSwitchInterface is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

13.8.7.3 Dcm Service Use Case: EcuReset ModeRapidPowerShutDown

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the current status of the *EcuReset ModeRapidPowerShutDown* service.

[TPS_SWCT_01708] AtomicSwComponentType supports *EcuReset ModeRapidPowerShutDown* service via diagnostic services ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_ModeRapidPowerShutDownModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_ModeRapidPowerShutDownModeSwitchInterface is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋*(RS_SWCT_00170, RS_SWCT_03190)*

13.8.7.4 Dcm Service Use Case: CommunicationControl

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the current status of the *CommunicationControl* service per ComM Channel.

[TPS_SWCT_01709] AtomicSwComponentType supports *CommunicationControl* service via diagnostic services ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_CommunicationControlModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_CommunicationControl is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋(*RS_SWCT_00170*, *RS_SWCT_03190*)

13.8.7.5 Dcm Service Use Case: ControlDTCSetting

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the current status of the *ControlDTCSetting* service.

[TPS_SWCT_01711] AtomicSwComponentType supports *ControlDTCSetting* service via diagnostic services ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_ControlDTCSettingModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_ControlDTCSettingModeSwitchInterface is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋(*RS_SWCT_00170*, *RS_-SWCT_03190*)

13.8.7.6 Dcm Service Use Case: SecurityAccess

Scenario: an AtomicSwComponentType offers an RPortPrototype typed by a ModeSwitchInterface to get informed about the current diagnostic security level.

[TPS_SWCT_01712] AtomicSwComponentType supports SecurityAccess to get informed about the security level ⌈

ServiceNeeds kind DiagnosticControlNeeds

RoleBasedPortAssignment valid roles:

- Dcm_SecurityAccessModeSwitchInterface [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role Dcm_SecurityAccessModeSwitchInterface is applicable for an RPortPrototype typed by a ModeSwitchInterface.⌋*(RS_SWCT_00170, RS_-SWCT_03190)*

13.8.7.7 Service Use Case: Atomic Software-Component implements a Hardware Shutdown

Scenario: if a hardware component is detected as being defective, the Dem shall inform the SwComponentPrototype typed by an AtomicSwComponentType which is responsible for executing a hardware-shutdown.

[TPS_SWCT_01680] Dem Use Case: Atomic Software-Component implements a Hardware Shutdown ⌈

ServiceNeeds kind DiagnosticComponentNeeds

RoleBasedPortAssignment valid roles:

- CallbackComponentStatusChanged [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03190)*

Class	**DiagnosticComponentNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to specify the service needs for the configuration of component events.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.60: DiagnosticComponentNeeds

13.8.7.8 Service Use Case: Upload and download of data

Scenario: a software-component implements the ability to accept data for upload and/or provide data for download. For this purpose the software-component provides a PPortPrototype that is supposed to be connected to the Dcm service component.

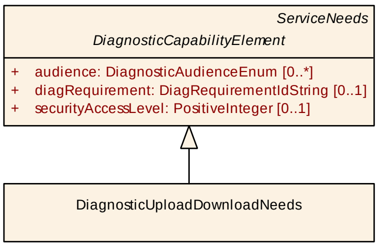


Figure 13.21: Specification of DiagnosticUploadDownloadNeeds

[TPS_SWCT_01769] Dcm Use Case: Upload and download of data ⌈

ServiceNeeds kind `DiagnosticUploadDownloadNeeds`

RoleBasedPortAssignment valid roles:

- `UploadDownloadServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_03190)*

Class	**DiagnosticUploadDownloadNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to specify needs regarding upload and download by means of diagnostic services.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.61: DiagnosticUploadDownloadNeeds

13.9 Diagnostic Log and Trace Dependency

The meta-class `DltUserNeeds` is used together with the `SwcServiceDependency` to define requirements in order to configure the Diagnostic Log and Trace module (for the terms related to the AUTOSAR Specification of Module DLT see [45]).

Class	DltUserNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class specifies the needs on the configuration of the Diagnostic Log and Trace module for one SessionId. This class currently contains no attributes. An instance of this class is used to find out which PortPrototypes of an AtomicSwComponentType belong to this SessionId in order to group the request and response PortPrototypes of the same SessionId. The actual SessionId value is stored in the PortDefinedArgumentValue of the respective PortPrototype specification.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.62: DltUserNeeds

Please note that for the described use case of the Dlt Service the following rule applies: For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`.

Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`.

The possible role attribute values and the multiplicity of the related `PortPrototype`s are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

13.9.1 Dlt use Case:Application software component transmits debug information

Scenario: `AtomicSwComponentType` sends log messages. In this case the following setup applies:

[TPS_SWCT_02506] Setup for Dlt use Case: Application software component accesses the Dlt module ⌈

ServiceNeeds kind : `DltUserNeeds`

RoleBasedPortAssignment valid roles:

- `DltControlService` [1]
- `LogTraceSessionControl` [1]
- `InjectionCallback` [0..1]
- `DltSwcMessageService` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(RS_SWCT_00030)

For more information please refer to [SWS_Dlt_00495], [SWS_Dlt_00496], and [SWS_Dlt_00498].

In this case the software-component has to provide one Client Port (DLTService) in order to register and de-register the context and to send log or trace messages.

13.10 Synchronized Time-Base Manager Dependency

The meta-class SyncTimeBaseMgrUserNeeds is used together with the SwcServiceDependency to define requirements in order to configure the Synchronized Time-Base Manager module (for the terms related to the AUTOSAR Specification of Module StbM see [46]).

Class	**SyncTimeBaseMgrUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the needs on the configuration of the Synchronized Time-base Manager for one time-base. This class currently contains no attributes. An instance of this class is used to find out which ports of a software-component belong to this time-base in order to group the request and response ports of the same time-base. The actual time-base value is stored in the PortDefinedArgumentValue of the respective port specification.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 13.63: SyncTimeBaseMgrUserNeeds

Please note that for the described use cases of the StbM Service following rule applies:
For every used ClientServerInterface it is necessary to create a RoleBasedPortAssignment.

Thereby the value of the attribute role of the RoleBasedPortAssignment has to be set to the name of the used standardized ClientServerInterface.

The possible role attribute values and the multiplicity of the related PortPrototypes are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

The general idea behind the time synchronization concept is that the role of global time master and global time slave are partly implemented in the application software.

For this purpose, the application software provides PortPrototypes typed by the standardized PortInterfaces GlobalTime_Master and GlobalTime_Slave.

In many cases both PortInterfaces GlobalTime_Master and GlobalTime_Slave will be used by the application software of one ECU. This means that the ECU is a global time slave on one domain and a global time master on another domain.

In terms of modeling, a given global time domain is represented by a SwcServiceDependency.

If one software-component has to deal with different global time domains (e.g. because it represents a slave in one domain and a master in another) then the corresponding SwcInternalBehavior needs to define one SwcServiceDependency per global time domain.

13.10.1 StbM Use Case: start timer and potentially get notified about its expiration

Scenario: a software-component wants to wind up a timer in the StbM with a given expiration time. The software-component may want to receive a notification when the timer expires. In this case the following setup applies:

[TPS_SWCT_01678] StbM use Case: start timer and potentially get notified about its expiration ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- StartTimer [1]
- TimeNotification [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software-component needs to have an RPortPrototype typed by the ClientServerInterface StartTimer and (if applicable) a PPortPrototype typed by the ClientServerInterface TimeNotification.

13.10.2 StbM Use Case: Software-Components wants to get notifications of status changes

Scenario: a software-component wants to receive events whenever the status of the StbM changes. For this purpose, the software-component sports a sender/receiver RPortPrototype. In this case the following setup applies:

[TPS_SWCT_01679] StbM use Case: Software-Components wants to get notifications of status changes ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- StatusNotification [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

The events received from the StbM have a fixed structure. For more details, please refer to [SWS_StbM_00284].

13.10.3 StbM Use Case: Process time snapshot obtained from global time slave for diagnostics purposes

Scenario: a software-component provides a PPortPrototype onto which the global time slave pushes snapshots of time synchronization records. This data is typically used for diagnostic purposes.

[TPS_SWCT_01740] StbM use Case: Process time snapshot obtained from global time slave for diagnostics purposes ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- MeasurementNotification [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

Note that in this case the software-components acts as a server, the StbM implements a client role!

[TPS_SWCT_01741] Suffix used for the resulting name of the PortInterface for measurement notification ⌈The suffix used for the resulting name of the PortInterface for the measurement notification MeasurementNotification_{TB_Name} shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00030)*

13.10.4 StbM Use Case: Software-component represents a global time master

Scenario: a software-component implements the application-software part of the global time master role. For this purpose the software-component exposes an RPortPrototype that is supposed to be connected to the StbM service component.

[TPS_SWCT_01742] StbM use Case: Software-component represents a global time master ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- GlobalTime_Master [1]
- StatusNotification [0..1]
- MeasurementNotification [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

[TPS_SWCT_01743] Suffix used for the resulting name of the PortInterface for the global time master role ⌈The suffix used for the resulting name of the PortInterface for the global time master role GlobalTime_Master_{Name} shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00030)*

13.10.5 StbM Use Case: Software-component represents a global time slave

Scenario: a software-component implements the application-software part of the global time slave role. For this purpose the software-component exposes an RPortPrototype that is supposed to be connected to the StbM service component.

[TPS_SWCT_01744] StbM use Case: Software-component represents a global time slave ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- GlobalTime_Slave [1]
- StatusNotification [0..1]
- MeasurementNotification [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*()*

[TPS_SWCT_01745] Suffix used for the resulting name of the PortInterface for the global time slave role ⌈The suffix used for the resulting name of the PortInterface for the global time slave role GlobalTime_Slave_{Name} shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00030)*

13.10.6 StbM Use Case: Software-component analyzes predictions about the time synchronization process

Scenario: A Time Slave collects information on the time synchronization process, to predict e.g. the sync ingress based on its local instance of the global time and check whether master and slave agree upon the current time. The software-component analyzes the predictions to detect any impairments.

[TPS_SWCT_01810] Software-component analyzes predictions about the time synchronization process ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- TimeBaseProviderNotification [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

For this purpose, the analyzer software-component exposes a PPortPrototype typed by the ClientServerInterface TimeBaseProviderNotification_{bus}_{TimeBase} to the StbM service component.

13.10.7 StbM Use Case: Software-Components wants to initiate the cloning of a time base

Scenario: a software-component wants to initiate the cloning of an existing time base by means of using the standardized Service Interface GlobalTime_Master, specifically the ClientServerOperation named Clone. In this case the following setup applies:

[TPS_SWCT_01854] Software-Components wants to initiate the cloning of a time base ⌈

ServiceNeeds Kind : SyncTimeBaseMgrUserNeeds

RoleBasedPortAssignment valid roles:

- GlobalTime_Master [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

The cloning of a time base is subject to certain pre-conditions. For more information, please refer to [SWS_StbM_00530]. Please note that regarding the determination of the “{Name}” suffix [TPS_SWCT_01743] applies.

13.11 Secure On-Board Communication

The meta-class SecureOnBoardCommunicationNeeds is used together with the SwcServiceDependency to define requirements in order to configure the Secure On-Board Communication module (for the terms related to the AUTOSAR Specification of Module **SecOc**, see [47]).

Class	**SecureOnBoardCommunicationNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the need for the existence of the SecOc module on the respective ECU. This class currently contains no attributes. An instance of this class is used to find out which ports of a software-component deal with the administration of secure communication in order to group the request and response ports.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
verification StatusIndication Mode	VerificationStatus IndicationModeEnum	0..1	attr	This attribute provides the ability to control the mode in which the application software is notified about the result of authentication attempts.

Table 13.64: SecureOnBoardCommunicationNeeds

Enumeration	**VerificationStatusIndicationModeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration provides options for setting the mode of a verification status indication.
Aggregated by	SecureOnBoardCommunicationNeeds.verificationStatusIndicationMode
Literal	*Description*
failureAndSuccess	Verification attempts that came out "false" or "true" shall be forwarded to the application software. **Tags:**atp.EnumerationLiteralIndex=1
failureOnly	Only verification attempts that came out "false" shall be forwarded to the application software. **Tags:**atp.EnumerationLiteralIndex=0

Table 13.65: VerificationStatusIndicationModeEnum

13.11.1 SecOc Use Case: obtain the verification status of secure communication

In this scenario, the `ApplicationSwComponentType` wants to obtain the status of secure communication.

It is not interested in the details (and would not be able to help anyway even if the details were available).

The `SwcServiceDependency` shall aggregate a `SecureOnBoardCommunicationNeeds`.

[TPS_SWCT_01668] SecOc Use Case: obtain the verification status of secure communication ⌈

RoleBasedPortAssignment valid roles:
n/a

RoleBasedDataAssignment valid roles:

- `VerificationStatus` [1]

RepresentedPortGroups
n/a

⌋(*RS_SWCT_00030*)

[constr_1681] Existence of attribute `RoleBasedDataAssignment.usedDataElement.localVariable` for `RoleBasedDataAssignment.role` = `VerificationStatus` ⌈If the attribute `RoleBasedDataAssignment.role` is set to the value `VerificationStatus` then the reference `RoleBasedDataAssignment.usedDataElement.localVariable` shall not exist **`at the time when the RTE is generated`**.⌋*()*

For explanation of the existence of [constr_1681], it is not intended to provide access to local variables inside the `SwcInternalBehavior`.

13.11.2 SecOc Use Case: software component retires from secure communication for a given period

In this scenario, the `ApplicationSwComponentType` undergoes a reconfiguration period in which it is not able to process any security-related data. During this period, the verification status shall always be set to "failed".

The `SwcServiceDependency` shall aggregate a `SecureOnBoardCommunicationNeeds`.

[TPS_SWCT_01672] `SecOc` Use Case: software component retires from secure communication for a given period ⌈

RoleBasedPortAssignment valid roles:

- `VerifyStatusConfiguration` [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(*RS_SWCT_00030*)

13.11.3 SecOc Use Case: deliver freshness to SecOC I

Scenario: a dedicated software-component computes and delivers the freshness to SecOc. The freshness can optionally be truncated by the software-component.

For this purpose, the software-component exposes a `PPortPrototype` to SecOc. This is used for sending a secured message by using the `ClientServerOperation` `GetTxFreshness` or `GetTxFreshnessTruncData`.

[TPS_SWCT_01716] `SecOc` Use Case: deliver freshness to SecOC I ⌈

ServiceNeeds kind `SecureOnBoardCommunicationNeeds`

RoleBasedPortAssignment valid roles:

- FreshnessManagement [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(RS_SWCT_00030)

13.11.4 SecOc Use Case: deliver freshness to SecOC II

Scenario: SecOc invokes transmit notification (SPduTxConfirmation) at freshness manager. This information can be vital for the computation of the freshness.

[TPS_SWCT_01717] SecOc Use Case: deliver freshness to SecOC II ⌈

ServiceNeeds kind SecureOnBoardCommunicationNeeds

RoleBasedPortAssignment valid roles:

- FreshnessManagement [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(RS_SWCT_00030)

13.11.5 SecOc Use Case: deliver freshness to SecOC III

Scenario: caused by out-of-sync freshness, SecOC cannot verify a MAC, and contacts the freshness manager (software-component) again to obtain a recalculated freshness value.

Each recalculation of the freshness inside the freshness manager is counted. After a given threshold of retries, SecOC has to drop the received message. For this purpose, the ClientServerOperation GetRxFreshness or GetRxFreshnessAuthData is used.

[TPS_SWCT_01718] SecOc Use Case: deliver freshness to SecOC III ⌈

ServiceNeeds kind SecureOnBoardCommunicationNeeds

RoleBasedPortAssignment valid roles:

- FreshnessManagement [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(*RS_SWCT_00030*)

13.11.6 SecOc Use Case: enable the sending of Pdus even if computation of the MAC is not possible

Scenario: there are cases where the ability to send authenticated messages associated with a given freshness id using a default-MAC is required, e.g. if an ECU has been replaced but was not yet provided with cryptographic keys.

Receivers can distinguish this case from the regular authenticated data exchange by looking at the MAC, i.e. a (configurable) default MAC is used in the described case.

[TPS_SWCT_01784] `SecOc` Use Case: enable the sending of Pdus even if computation of the MAC is not possible ⌈

ServiceNeeds kind `SecureOnBoardCommunicationNeeds`

RoleBasedPortAssignment valid roles:

- `SendDefaultAuthenticationInformation` [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(*RS_SWCT_00030*)

13.11.7 SecOc Use Case: Receive notification about an authentication attempt

Scenario: a software-component wants to receive a notification about the status of an authentication attempt.

Such a notification can be used to continuously monitor the number of failed verification attempts with the goal to set up a security management system/intrusion detection system that is able to detect an attack that consists of a flood of authentication requests and react with adequate dynamic counter-measures.

For this purpose, the software-component exposes a `PPortPrototype` that implements the notification handler. The handler may be interested in only *failed* attempts **or** *failed and passed* attempts.

This aspect can be configured by means of attribute SecureOnBoardCommunicationNeeds.verificationStatusIndicationMode.

[TPS_SWCT_01832] SecOc Use Case: Receive notification about an authentication attempt ⌈

ServiceNeeds kind SecureOnBoardCommunicationNeeds

RoleBasedPortAssignment valid roles:

- VerificationStatusIndication [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_00030)*

13.12 J1939 Communication

The J1939-specific meta-classes J1939RmOutgoingRequestServiceNeeds and J1939RmIncomingRequestServiceNeeds are used together with the SwcServiceDependency to define requirements in order to configure the J1939 request manager (for the terms related to the AUTOSAR Specification of Module **J1939RM**, see [48]).

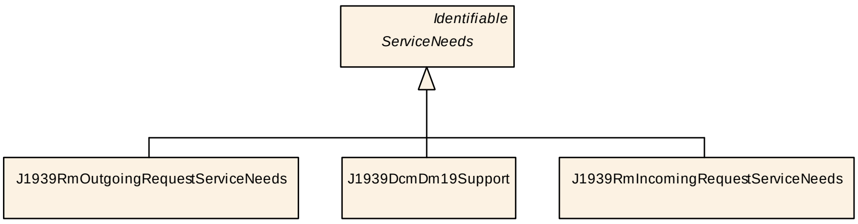


Figure 13.22: ServiceNeeds: J1939-related ServiceNeeds

Class	**J1939RmOutgoingRequestServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class shall be used to specify needs with respect to the configuration of the J1939Rm, in particular for the case where an ApplicationSwComponentType needs to send a request to another J1939 node.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.66: J1939RmOutgoingRequestServiceNeeds

Class	J1939RmIncomingRequestServiceNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	"This meta-class shall be used to specify needs with respect to the configuration of the J1939Rm, in particular for the case where an ApplicationSwComponentType needs to accept a request from another J1939 node.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.67: J1939RmIncomingRequestServiceNeeds

13.12.1 J1939RM Use Case: AtomicSwComponentType sends requests to the bus

Scenario: An `AtomicSwComponentType` sends requests to the bus. In this case the following setup applies:

To indicate the scenario described in this use case the `SwcServiceDependency` shall aggregate a `J1939RmOutgoingRequestServiceNeeds`.

[TPS_SWCT_01673] Application Software Component sends requests using the J1939Rm ⌈

ServiceNeeds kind `J1939RmOutgoingRequestServiceNeeds`

RoleBasedPortAssignment valid roles:

- `AppSendRequest` [1]
- `AppAckIndication` [1]
- `AppRequestTimeoutIndication` [0..1]
- `AppCancelRequestTimeout` [0..1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_03180)*

For more information please refer to the following specification items: [SWS_-J1939Rm_00104], [SWS_J1939Rm_00106], [SWS_J1939Rm_00108], and [SWS_-J1939Rm_00105].

13.12.2 J1939RM Use Case: AtomicSwComponentType accepts requests from the bus

Scenario: An AtomicSwComponentType accepts requests from the bus. In this case the following setup applies:

To indicate the scenario described in this use case the SwcServiceDependency shall aggregate a J1939RmIncomingRequestServiceNeeds.

[TPS_SWCT_01674] Application Software Component accepts requests using the J1939Rm ⌈

ServiceNeeds kind J1939RmIncomingRequestServiceNeeds

RoleBasedPortAssignment valid roles:

- AppRequestIndication [1]
- AppSendAck [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_03180)*

For more information please refer to the following specification items: [SWS_-J1939Rm_00103] and [SWS_J1939Rm_00107].

13.12.3 J1939Dcm wants to retrieve calibration verification numbers from an application software-component

Scenario: J1939Dcm wants to retrieve calibration verification numbers from an application software-component. For this purpose, the software-component exposes a PPortPrototype to the J1939Dcm

[TPS_SWCT_01809] J1939Dcm wants to retrieve calibration verification numbers from an application software-component ⌈

ServiceNeeds kind J1939DcmDm19Support

RoleBasedPortAssignment valid roles:

- J1939Dcm_CalibrationInformation [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋(RS_SWCT_03180)

Class	J1939DcmDm19Support			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The software-component provides information about calibration verification numbers for inclusion in DM19			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 13.68: J1939DcmDm19Support

13.13 Error Tracer

The meta-class ErrorTracerNeeds is used to define requirements in order to configure the Default Error Tracer.

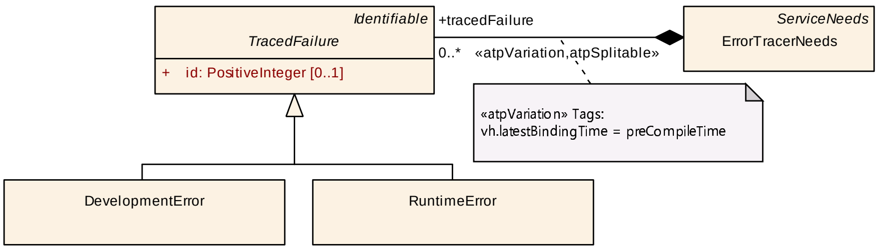


Figure 13.23: Modeling of ErrorTracerNeeds

In particular, ErrorTracerNeeds provides the exhaustive list of all tracedFailure implemented in the enclosing software-component and reported via the PortPrototype referenced via RoleBasedPortAssignment.

Each tracedFailure relates to one ID, represented by attribute TracedFailure.id.

For more explanation, please consult with the specification of Default Error Tracer [49].

Class	ErrorTracerNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the need to report failures to the error tracer.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	Type	Mult.	Kind	Note

▽

△

Class	ErrorTracerNeeds			
tracedFailure	TracedFailure	*	aggr	list of traced failures **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=tracedFailure.shortName, traced Failure.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 13.69: ErrorTracerNeeds

Class	***TracedFailure*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the ability to report a specific failure to the error tracer. The short name specifies the literal applicable for the Default Error Tracer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DevelopmentError, RuntimeError, TransientFault			
Aggregated by	ErrorTracerNeeds.tracedFailure			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
id	PositiveInteger	0..1	attr	ID of detected failure used in reporting API as error or fault id.

Table 13.70: TracedFailure

Class	**DevelopmentError**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The reported failure is classified as development error.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TracedFailure*			
Aggregated by	ErrorTracerNeeds.tracedFailure			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.71: DevelopmentError

Class	**RuntimeError**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The reported failure is classified as runtime error.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TracedFailure*			
Aggregated by	ErrorTracerNeeds.tracedFailure			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.72: RuntimeError

13.13.1 Error Tracer Use Case: Default Error Tracer Service use Case: report failure

[TPS_SWCT_01694] Setup for Default Error Tracer Service use Case: development errors or runtime error ⌈

Scenario: a software-component reports development errors or runtime error to the Default Error Tracer. In this case the following setup applies

ServiceNeeds kind ErrorTracerNeeds

RoleBasedPortAssignment valid roles:

- DETService[1]

RoleBasedDataAssignment valid roles:
n/a

RoleBasedDataTypeAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_00030)*

[constr_1433] Transient faults are not applicable to software-components ⌈An ErrorTracerNeeds aggregated in the context of a SwcInternalBehavior is - **at the time when the RTE is generated** - not allowed to own a TransientFault in the role ErrorTracerNeeds.tracedFailure.⌋*()*

13.14 Vehicle-2-X Facilities

The meta-class V2xFacUserNeeds is used together with the SwcServiceDependency to define requirements in order to configure the V2xFac module (for the terms related to the AUTOSAR Specification of Module V2xFac see [50]).

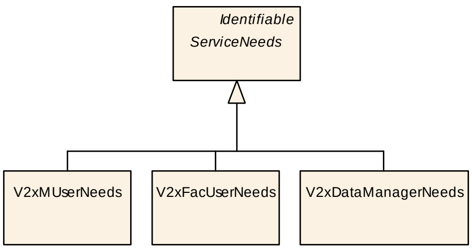


Figure 13.24: Modeling of V2xFacUserNeeds

Class	**V2xFacUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to define service needs for V2x facilities.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.73: V2xFacUserNeeds

Please note that for the described use cases of the V2xFac Service following rule applies:
For every used SenderReceiverInterface it is necessary to create a RoleBasedDataAssignment.

13.14.1 V2xFac Use Case: Application software component provides Vehicle specific data to the V2X-Stack for CAM transmission

Scenario: an AtomicSwComponentType autonomously calls the V2xFac, providing vehicle data collected via the in-vehicle networks in the module. In this case the following setup applies:

[TPS_SWCT_01728] V2xFac Use Case: Application software component provides Vehicle specific data to the V2X-Stack for CAM transmission ⌈

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- V2xFacVdp [1]

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one Sender Port (V2xFacVdp) to

- provide the Vehicle Data

[constr_1682] Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xFacVdp ⌈If the attribute RoleBasedDataAssignment.role is set to the value V2xFacVdp then the reference RoleBasedDataAssignment.usedDataElement.localVariable shall not exist **at the time when the RTE is generated**.⌋*()*

For explanation of the existence of [constr_1682], it is not intended to provide access to local variables inside the SwcInternalBehavior.

13.14.2 V2xFac Use Case: Application software component triggers transmission of DENM message

Scenario: an AtomicSwComponentType shall be able to trigger the transmission of different DENM types. In this case the following setup applies:

[TPS_SWCT_01730] V2xFac Use Case: Application software component triggers transmission of DENM message ⌈

RoleBasedPortAssignment valid roles:

- V2xFacDenBs [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xFacDenBs) to

- trigger new DENM message
- trigger updated DENM message
- trigger a cancellation of a DENM message

13.15 Vehicle-2-X Management

The meta-class V2xMUserNeeds is used together with the SwcServiceDependency to define requirements in order to configure the V2X Manager module (for the terms related to the AUTOSAR Specification of Module V2xM see [51]).

Class	**V2xMUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to express service needs for the V2x management.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.74: V2xMUserNeeds

Please note that for the described use cases of the V2xFac Service following rule applies:
For every used ClientServerInterface it is necessary to create a RoleBasedPortAssignment.

Thereby the value of the attribute role of the RoleBasedPortAssignment has to be set to the name of the used standardized ClientServerInterface.

The possible role attribute values and the multiplicity of the related PortPrototypes are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

13.15.1 V2xM Use Case: Application software component provides Vehicle specific data to the V2X-Stack for Position and Time information

Scenario: an AtomicSwComponentType autonomously calls the V2X Manager, providing vehicle data collected via the in-vehicle networks in the module. In this case the following setup applies:

[TPS_SWCT_01731] V2xM Use Case: Application software component provides Vehicle specific data to the V2X-Stack for Position and Time information ⌈

RoleBasedPortAssignment valid roles:

- V2xM_Vdp [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xM_Vdp) to

- set the current time and position

13.15.2 V2xM Use Case: Application software component needs V2X specific data from the V2X Manager

Scenario: an AtomicSwComponentType autonomously calls the V2X Manager, getting information of V2X specific data in the module. In this case the following setup applies:

[TPS_SWCT_01732] V2xM Use Case: Application software component needs V2X specific data from the V2X Manager ⌈

RoleBasedPortAssignment valid roles:

- V2xM_Vdp [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xM_Vdp) to

- access the current time of the V2X-Stack, based on the system clock
- access the earliest date of expiration of a Long Term Certificate
- access the earliest date of expiration of a Pseudonym Certificate

13.15.3 V2xM Use Case: Application software component has soft-control over Pseudonym-Change within V2X Manager

Scenario: an AtomicSwComponentType autonomously calls the V2X Manager, setting the locked or unlocked state for pseudonym change. In this case the following setup applies:

[TPS_SWCT_01733] V2xM Use Case: Application software component has soft-control over Pseudonym-Change within V2X Manager ⌈

RoleBasedPortAssignment valid roles:

- V2xM_PseudonymChange [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xM_PseudonymChange) to

- set the state of the pseudonym change to locked
- set the state of the pseudonym change to unlocked

13.15.4 V2xM Use Case: Application software component has the ability to do Verification-on-Demand

Scenario: an AtomicSwComponentType autonomously calls the V2X Manager, authenticating a previously received message. In this case the following setup applies:

[TPS_SWCT_01734] V2xM Use Case: Application software component has the ability to do Verification-on-Demand ⌈

RoleBasedPortAssignment valid roles:

- V2xM_Sec [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xM_Sec) to

- verify a previously received message

13.15.5 V2xM Use Case: Application software component do location based calculations

Scenario: an AtomicSwComponentType autonomously calls the V2X Manager, getting results for geographical calculations. In this case the following setup applies:

[TPS_SWCT_01735] V2xM Use Case: Application software component do location based calculations ⌈

RoleBasedPortAssignment valid roles:

- V2xM_GeoMath [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one Client PortPrototype (typed by the standardized ClientServerInterface V2xM_GeoMath) to

- calculate the distance between two location tuples (latitude, longitude)
- calculate an allowed tolerance value between two heading values

13.16 Vehicle-2-X DataManager

The V2X DataManager receives V2X messages and has the ability to pick dedicated portions out of the incoming messages and forward them over a PPortPrototype owned by the respective SwComponentPrototype typed by the V2X DM's ServiceSwComponentType to other communication participants.

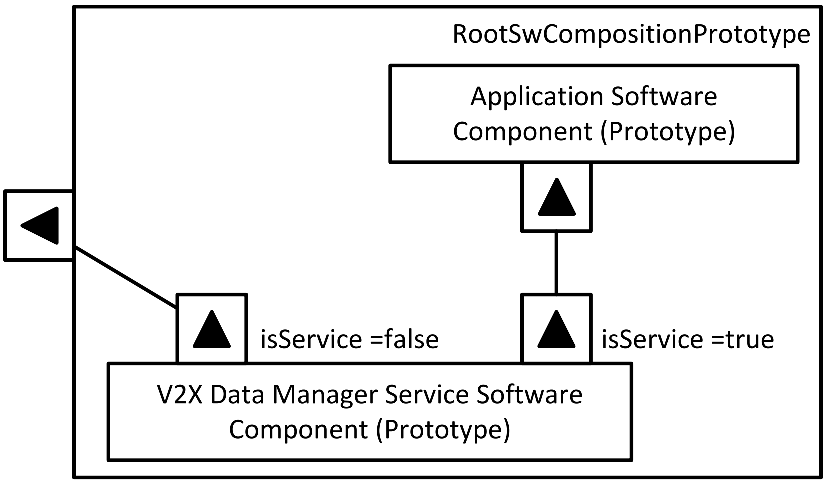


Figure 13.25: V2X DM communication use cases

[TPS_SWCT_01867] Communication Use case of the V2X Data Manager where isService = true ⌈In this communication scenario, the V2X Data Manager sends the data to a SwComponentPrototype on the same EcuInstance. In this case it is foreseen to use a standardized SenderReceiverInterface that set the attribute isService to true and define a SwcServiceDependency that aggregates a V2xDataManagerNeeds.

In this case, the connection between PPortPrototype owned by the SwComponentType representing the V2X DM's service component and the AbstractRequiredPortPrototype owned by the SwComponentType typed by an ApplicationSwComponentType is established by an AssemblySwConnector.⌋*()*

[TPS_SWCT_01868] Communication Use case of the V2X Data Manager where isService = false ⌈In this communication scenario, the V2X DataManager sends the data to another PPortPrototype that is typed by a compatible SenderReceiverInterface and is owned by the RootSwCompositionPrototype.

Thus, the connection between the PPortPrototype owned by the SwComponentType representing the V2X DM's service component and the PPortPrototype owned by the RootSwCompositionPrototype is established by means of an DelegationSwConnector.⌋()

The use cases mentioned in [TPS_SWCT_01867] (send data to a local software-component) and [TPS_SWCT_01868] (send data to a different ECU) are illustrated in Figure 13.25.

Class	V2xDataManagerNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to define service needs for V2x Data Manager.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.75: V2xDataManagerNeeds

13.16.1 V2x DM Use Case: Application software component receives V2X data

Scenario: an application software component receives portions of an ASN.1 message via catalog items defined and provided by the service-component of the V2X Data Manager.

[TPS_SWCT_01869] V2x DM Use Case: Application software component receives portions of an ASN.1 message ⌈

ServiceNeeds kind `V2xDataManagerNeeds`

RoleBasedPortAssignment valid roles:
N/A

RoleBasedDataAssignment valid roles:

- `dataManagerServiceCommunication` [1]

RoleBasedDataTypeAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_00030)*

In this case the software component has to provide one receiver RPortPrototype (typed by the standardized AUTOSAR SenderReceiverInterface used by the V2X DM) to access the transmitted portions of the ASN.1 message.

13.17 Hardware Test Manager

The service use cases for the *Hardware Test Manager* are indicated by the usage of meta-class HardwareTestNeeds in the role SwcServiceDependency.serviceNeeds.

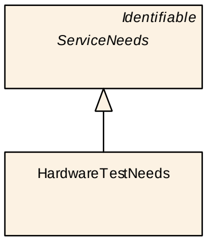


Figure 13.26: Modeling of HardwareTestNeeds

Class	**HardwareTestNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that a software-component is interested in the results of the hardware test and will establish a PortPrototype to query the hardware test manager.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table 13.76: HardwareTestNeeds

13.17.1 HtssM Service Use Case: Query results of hardware tests

Scenario: A software-component wants to query the results of hardware tests conducted by the HtssM. For this purpose, the software-component exposes an RPortPrototype that shall be connected to the HtssM.

[TPS_SWCT_01763] HtssM Service Use Case: Query results of hardware tests ⌈

ServiceNeeds kind : HardwareTestNeeds

RoleBasedPortAssignment valid roles:

- GetTestStatus [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(*RS_SWCT_00030*)

13.18 Intrusion Detection System Manager

The interaction with the Intrusion Detection System Manager consists of the ability to report security events.

For this purpose, two different kinds of ServiceNeeds are provided: IdsMgrNeeds for the actual reporting of security events, and IdsMgrCustomTimestampNeeds for the retrieval of a timestamp by the IdsM.

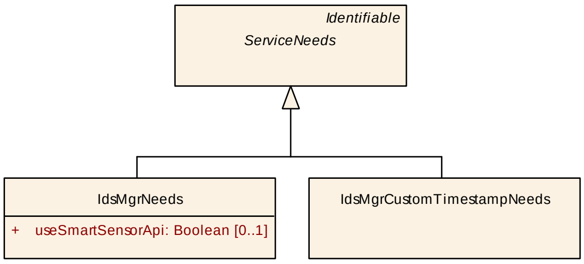


Figure 13.27: Modeling of ServiceNeeds for intrusion detection system management

Class	**IdsMgrNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class is used to indicate that the enclosing SwcServiceDependency represents a service use case for the Intrusion Detection System Manager. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
useSmart SensorApi	Boolean	0..1	attr	This attribute controls whether the reporting of the security event shall be done by means of the smart sensor API.

Table 13.77: IdsMgrNeeds

Class	**IdsMgrCustomTimestampNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class is used to indicate that the enclosing SwcServiceDependency represents a service use case for the retrieval of a custom timestamp by the Intrusion Detection System Manager. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Aggregated by	BswServiceDependency.serviceNeeds, SwcServiceDependency.serviceNeeds			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table 13.78: IdsMgrCustomTimestampNeeds

13.18.1 IdsM Service Use Case: AtomicSwComponentType reports security event

Scenario: An AtomicSwComponentType reports a security event. It does not use the so-called Smart Sensor API.

To indicate the scenario described in this use case the SwcServiceDependency shall aggregate a IdsMgrNeeds.

[TPS_SWCT_01826]{DRAFT} **Application Software Component reports security event** ⌈

ServiceNeeds kind IdsMgrNeeds

RoleBasedPortAssignment valid roles:

- IdsMService [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_00030)*

[TPS_SWCT_01827]{DRAFT} **Suffix used for the resulting name of the PortInterface for the IdsM Services** ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (IdsMService_{EventName}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋*(RS_SWCT_00030)*

13.18.2 IdsM Service Use Case: AtomicSwComponentType reports security event using Smart Sensor API

Scenario: An AtomicSwComponentType reports a security event. For this purpose, the so-called Smart Sensor API is used.

To indicate the scenario described in this use case the SwcServiceDependency shall aggregate a IdsMgrNeeds.

[TPS_SWCT_01828]{DRAFT} **Application Software Component reports security event using Smart Sensor API** ⌈

ServiceNeeds kind IdsMgrNeeds

RoleBasedPortAssignment valid roles:

- IdsMSmartSensorService [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*()*

[TPS_SWCT_01829]{DRAFT} **Suffix used for the resulting name of the PortInterface for the IdsM Services** ⌈The *suffix* used for the resulting name of the PortInterface for the Data Services (IdsMSmartSensorService_{EventName}) shall be taken from the shortName of the applicable SwcServiceDependency.⌋ *(RS_SWCT_00030)*

13.18.3 IdsM Service Use Case: AtomicSwComponentType provides time stamp to IdsM

Scenario: An AtomicSwComponentType exposes a PPortPrototype for providing a time stamp to the IdsM.

To indicate the scenario described in this use case the SwcServiceDependency shall aggregate a IdsMgrCustomTimestampNeeds.

[TPS_SWCT_01830]{DRAFT} **Application Software Component provides time stamp to IdsM** ⌈

ServiceNeeds kind IdsMgrCustomTimestampNeeds

RoleBasedPortAssignment valid roles:

- IdsMCustomTimestamp [1]

RoleBasedDataAssignment valid roles:
n/a

RepresentedPortGroups
n/a

⌋*(RS_SWCT_00030)*

14 Rapid Prototyping Scenarios

14.1 Definition of Rapid Prototyping Scenario

A Rapid Prototyping Scenario consist out of two main aspects: The description of the byPassPoints (see Figure 14.1) and the relation to a rptHook (see Figure 14.2).

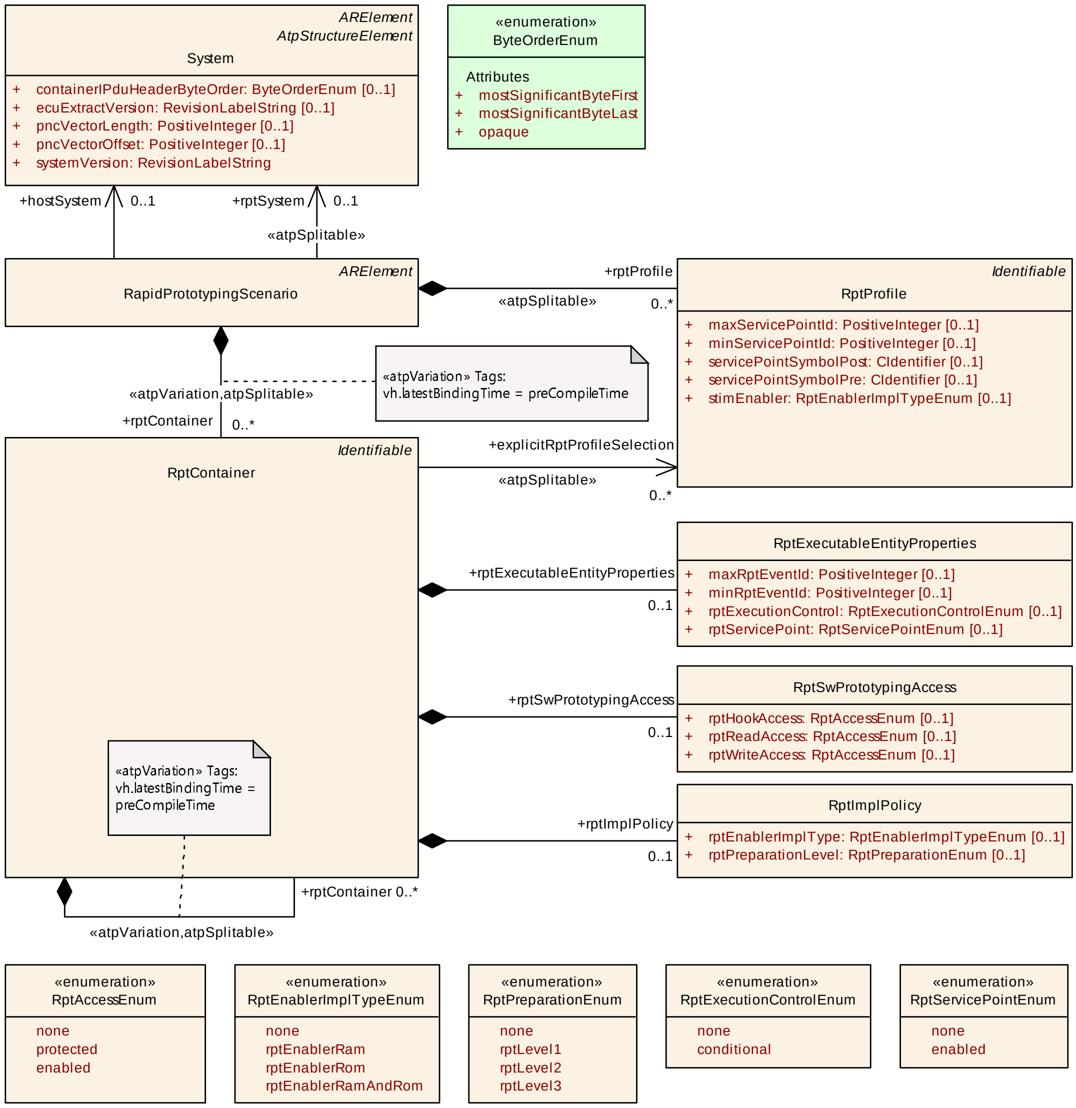


Figure 14.1: Rapid Prototyping Scenario

A Rapid Prototyping Scenario is structured by means of RptContainers. The correct usage of RptContainer structure is described in 14.2.

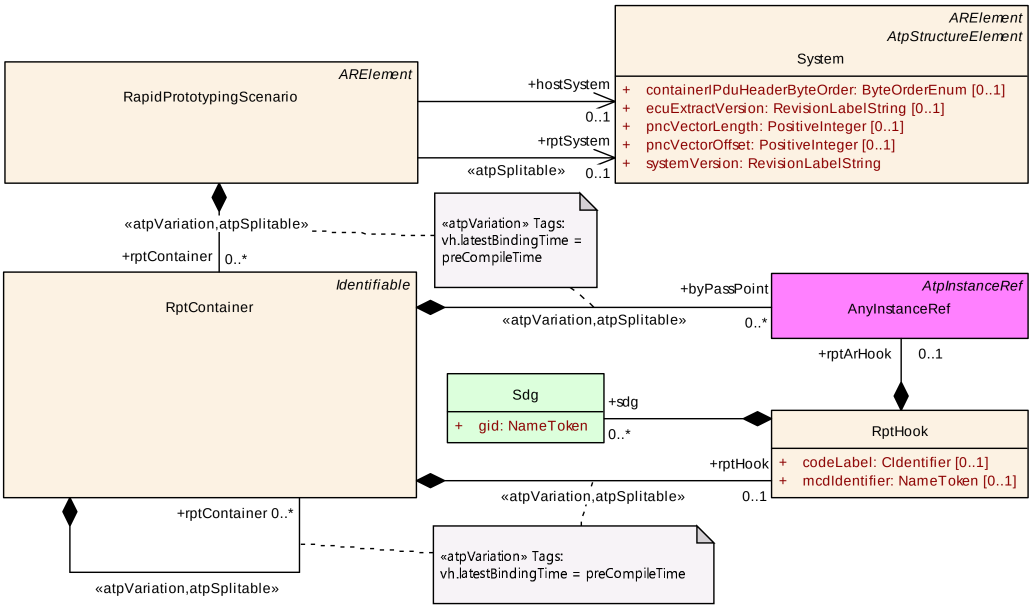


Figure 14.2: Rapid Prototyping Hooks

Class	**RapidPrototypingScenario**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class provides the ability to describe a Rapid Prototyping Scenario. Such a Rapid Prototyping Scenario consist out of two main aspects, the description of the byPassPoints and the relation to an rpt Hook. **Tags:**atp.recommendedPackage=RapidPrototypingScenarios			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
hostSystem	System	0..1	ref	System which describes the software components of the host ECU.
rptContainer	RptContainer	*	aggr	Top-level rptContainer definitions of this specific rapid prototyping scenario. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rptContainer.shortName, rpt Container.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
rptProfile	RptProfile	*	aggr	Defiens the applicable Rapid Prototyping profils which are especially defining the smbol of the service functions and the valid id range. The order of the RptProfiles determines the order of the service function invocation by RTE. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=rptProfile.shortName
rptSystem	System	0..1	ref	System which describes the rapid prototyping algorithm in the format of AUTOSAR Software Components. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=rptSystem

Table 14.1: RapidPrototypingScenario

[constr_1987] Existence of instance reference `RapidPrototypingScenario.hostSystem` ⌈For each `RapidPrototypingScenario`, the instance reference to

ModeDeclaration in the role hostSystem shall exist **at the time when the RTE is generated**.⌋()

Class	**RptContainer**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class defines a byPassPoint and the relation to a rptHook. Additionally it may contain further rptContainers if the byPassPoint is not atomic. For example a byPass Point referencing to a RunnableEntity may contain rptContainers referring to the data access points of the RunnableEntity. The RptContainer structure on M1 shall follow the M1 structure of the Software Component Descriptions. The category attribute denotes which level of the Software Component Description is annotated.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	RapidPrototypingScenario.rptContainer, RptContainer.rptContainer			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
byPassPoint	AtpFeature	*	iref	byPassPoint describes the required preparation of the host ECU. At a byPassPoint the host ECU shall be capable to communicate with a RPT System in order to support the execution of the rapid prototyping algorithms with the original data calculated by the host system and to replace dedicated results of the host system by the results of the rapid prototyping algorithm. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=byPassPoint.contextElement, byPass Point.target, byPassPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime **InstanceRef implemented by:**AnyInstanceRef
explicitRpt ProfileSelection	RptProfile	*	ref	This attribute defines the applicable RptProfiles for the specific RptContainer. If not any references to a specific RptProfile is defined, all RptProfiles defined in the Rapid PrototypingScenario are applicable. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=explicitRptProfileSelection
rptContainer	RptContainer	*	aggr	Sub-level rptContainer definitions of this specific rapid prototyping scenario. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rptContainer.shortName, rpt Container.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
rptExecutable EntityProperties	RptExecutableEntity Properties	0..1	aggr	Describes the required code preparation for rapid prototyping at ExecutableEntity invocation.
rptHook	RptHook	0..1	aggr	The rptHook describes the link between a byPassPoint and the rapid prototyping algorithm. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rptHook, rptHook.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
rptImplPolicy	RptImplPolicy	0..1	aggr	Describes the required code preparation for rapid prototyping at data accesses.
rptSw Prototyping Access	RptSwPrototyping Access	0..1	aggr	Describes the required accessibility of data and modes by the rapid prototyping tooling.

Table 14.2: RptContainer

Class	RptHook			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class provide the ability to describe a rapid prototyping hook. This can either be described by an other AUTOSAR system with the category RPT_SYSTEM or as a non AUTOSAR software.			
Base	*ARObject*			
Aggregated by	RptContainer.rptHook			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
codeLabel	CIdentifier	0..1	attr	This attribute provides a code label which is used in the implementation of the hook. For example this can be an C function name or the name of data definition.
mcdIdentifier	NameToken	0..1	attr	This attribute provides an identifier which shall be used in a MCD System to display the Rpt Hook.
rptArHook	AtpFeature	0..1	iref	This describes the hook with the means of another AUTOSAR system. **InstanceRef implemented by:**AnyInstanceRef
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data.

Table 14.3: RptHook

[TPS_SWCT_02046] `byPassPoint` specifies the rapid prototyping capability ⌈The `byPassPoint`s are used to describe the preparation of the host ECU. At the `byPassPoint`s the host ECU shall be capable to communicate with an RPT System in order to support the execution of the rapid prototyping algorithms with the original data calculated by the host system and to replace dedicated results of the host system by the results of the rapid prototyping algorithm.⌋*(RS_SWCT_03280, RS_SWCT_03282)*

[TPS_SWCT_02047] `RptHook` specifies the link to rapid prototyping algorithm ⌈The `rptHook` describes the link between the `byPassPoint` and the rapid prototyping algorithm. If the rapid prototyping algorithm is described as an AUTOSAR Software Component the `rptArHook` reference is applicable. Otherwise, the definition of a `codeLabel` and optionally `mcdIdentifier` shall be used.⌋*(RS_SWCT_03280, RS_SWCT_03281, RS_SWCT_03282)*

In order to describe an RPT system as AUTOSAR software component a `System` with the `category` `RPT_SYSTEM` shall be defined.

[constr_2054] Valid targets of `rptSystem` ⌈The `System` referenced in the role `rptSystem` shall be of `category` `RPT_SYSTEM`.

This rule shall be imposed **`at the time when the RTE is generated`**.⌋*()*

14.2 Usage of RptContainers on M1

The `RptContainer` structure on M1 shall follow the M1 structure of the Software Component Descriptions. The `category` attribute denotes which level of the Software Component Description is annotated.

The following values of the attribute category are predefined by the AUTOSAR standard:

[constr_2055] Valid targets of byPassPoint and rptHook reference, depending on the value of attribute category ⌈

Category	*Meaning*	*Specific properties*
SW_COMPONENT _PROTOTYPE	Adds one SwComponentPrototype to an Rapid Prototyping Scenario.	The byPassPoint and rptArHook shall reference a SwComponentPrototypes.
DATA_PROTOTYPE	Adds one instance of a DataPrototype to an Rapid Prototyping Scenario.	The byPassPoint and rptArHook shall reference a DataPrototype instances in PortPrototypes.
RUNNABLE_ENTITY	Adds one RunnableEntity to an Rapid Prototyping Scenario.	The byPassPoint and rptArHook shall reference a RunnableEntity instances.
ACCESS_POINTS	Adds one VariableAccess, ParameterAccess, ServerCallPoint, AsynchronousServerCallResultPoint, InternalTriggeringPoint, ModeSwitchPoint, ModeAccessPoint or ExternalTriggeringPoint to a Rapid Prototyping Scenario.	The byPassPoint and rptArHook shall reference a VariableAccess, ParameterAccess, ServerCallPoint, AsynchronousServerCallResultPoint, InternalTriggeringPoint, ModeSwitchPoint, ModeAccessPoint or ExternalTriggeringPoint instances.

This rule shall be imposed **at the time when the RTE is generated**.

⌋*()*

Hereby, the following semantic applies:

[TPS_SWCT_02048] Implicit SwComponentPrototype selection for Rapid Prototyping Scenario ⌈If a SwComponentPrototype is referenced in the role byPassPoint by a RptContainer without further “Sub” rptContainer, all RTE Interfaces of the AtomicSwComponentType shall be able to support a connection to a rptHook.⌋*(RS_SWCT_03280)*

[TPS_SWCT_02049] Implicit RunnableEntity selection for Rapid Prototyping Scenario ⌈If a RunnableEntity is referenced in the role byPassPoint by a RptContainer without further “Sub” rptContainer, all RTE Interfaces of the RunnableEntity shall be able to support a connection to a rptHook.⌋*(RS_SWCT_03280)*

[TPS_SWCT_02050] Explicit selection of access points for Rapid Prototyping Scenario ⌈If a VariableAccess, ParameterAccess, ServerCallPoint, AsynchronousServerCallResultPoint, InternalTriggeringPoint, ModeSwitchPoint, ModeAccessPoint or ExternalTriggeringPoint is referenced in the role byPassPoint by a RptContainer, only RTE Interfaces related to the specific access point are required be able to support a connection to a rptHook.⌋ *(RS_SWCT_03280)*

[TPS_SWCT_02051] Explicit DataPrototype selection for Rapid Prototyping Scenario ⌈If a DataPrototype instances in a PortPrototypes is referenced in the role byPassPoint by a RptContainer, only RTE Interfaces related to the specific DataPrototype are required be able to support a connection to a rptHook.⌋ *(RS_SWCT_03280)*

[constr_2056] Consistency of RapidPrototypingScenario with respect to rptSystem and rptArHook references ⌈Within one RapidPrototypingScenario all rptSystem references shall point to instances in one and only one System, and if existent, all rptArHook shall point to instances in one other and only one other System.

This rule shall be imposed **at the time when the RTE is generated**.⌋*()*

14.3 Usage of atpSplitable for RptContainers on M1

In order to support the later definition of the RptHooks, which may require as well the detailed specification byPassPoints, the aggregation of RptContainer and RptHook is ≪atpSplitable≫.

[TPS_SWCT_02052] Definition of Rapid Prototyping Scenario is splittable ⌈The aggregations in the roles rptContainer, byPassPoint, explicitRptProfileSelection, and rptHook at RptContainer are decorated by stereotype ≪atpSplitable≫. By this means, it is possible to specify aspects of RptContainers in a later process step.⌋*(RS_SWCT_03280)*

Please note that the later specification of RptHooks may require additional byPassPoints as well to show their relationship to lower level elements in a component description, such as VariableAccess, where in contrast the byPassPoints may only be specified on higher level elements such as SwComponentPrototypes in a first step.

14.4 Modifications of the Meta-Model for supporting the RPT scenario

The implementation of the rapid-prototyping scenario implies the definition of *access points* (see [constr_2055]). To be able to fulfill this role, the *access points* shall be represented by meta-classes derived from Referrable.

Most candidates for becoming *access points* are already inheriting from Referrable and therefore do not require further treatment (see Figure 14.3). Two meta-classes in this collection, however, are not derived from Referrable:

- ExternalTriggeringPoint
- ModeAccessPoint

It is not feasible to fix this issue by simply letting the two meta-classes inherit from Referrable because this would break the backwards compatibility of the AUTOSAR XML Schema[1].

[1]Because in this case the shortName becomes mandatory.

Therefore, a different approach (as sketched in Figure 14.3) has been implemented.

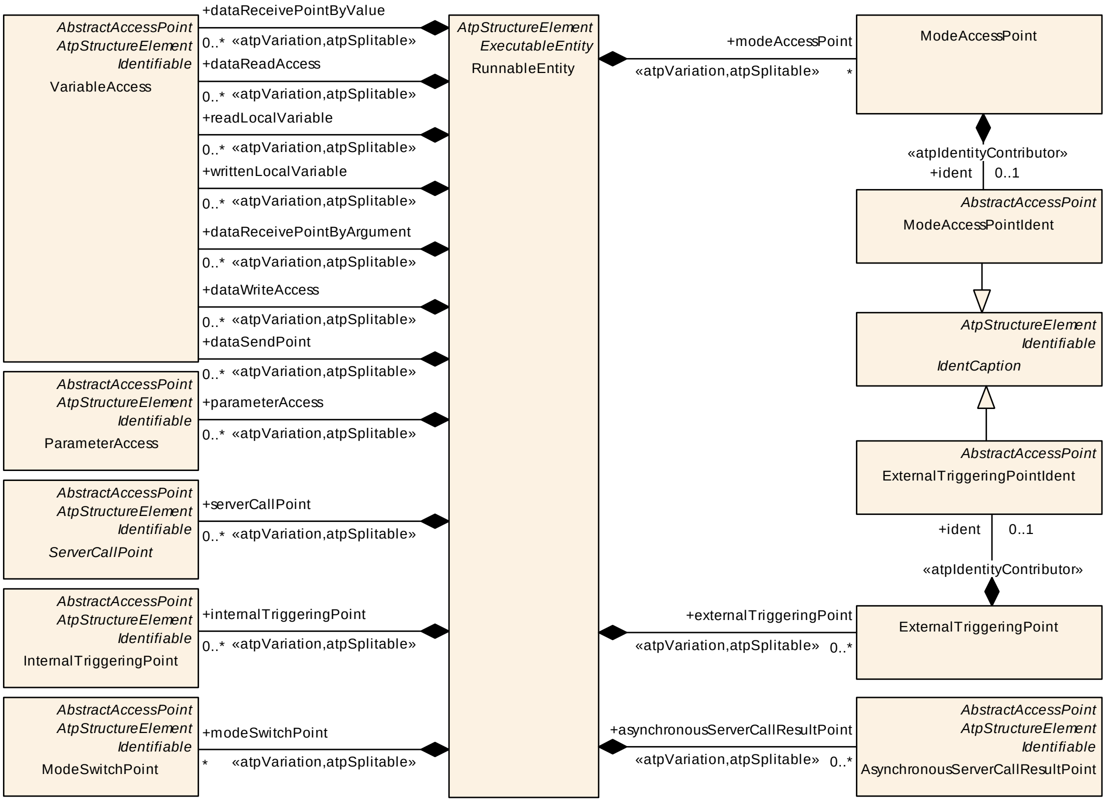


Figure 14.3: Access Points used in the context of the Rapid Prototyping Scenario

A new meta-class `IdentCaption` is created that introduces the capabilities of the meta-class `Identifiable` (that, in turn, inherits from `Referrable`) to its subclasses, `ModeAccessPointIdent` and `ExternalTriggeringPointIdent`.

These, in turn, are **optionally**[2] aggregated in the role `ident` by `ModeAccessPoint`, or in the role `ident` by meta-class `ExternalTriggeringPoint`.

Class	***IdentCaption*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class represents the caption. This allows having some meta-classes optionally identifiable.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswServiceDependencyIdent, DiagnosticParameterIdent, ExternalTriggeringPointIdent, ModeAccessPointIdent			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table 14.4: IdentCaption

[2] Again, the optional aggregation is a necessary prerequisite to not break the backwards compatibility of the AUTOSAR XML schema.

Class	ModeAccessPointIdent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class has been created to introduce the ability to become referenced into the meta-class Mode AccessPoint without breaking backwards compatibility.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *IdentCaption*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeAccessPoint.ident			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 14.5: ModeAccessPointIdent

Class	ExternalTriggeringPointIdent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta-class has been created to introduce the ability to become referenced into the meta-class ExternalTriggeringPoint without breaking backwards compatibility.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *IdentCaption*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ExternalTriggeringPoint.ident			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table 14.6: ExternalTriggeringPointIdent

The following (simplified) listing 14.1 sketches the usage of the meta-class `IdentCaption` for the purpose of effectively allowing references to a `ModeAccessPoint`.

```
<AR-PACKAGE>
  <SHORT-NAME>IC_Example</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>ASCT</SHORT-NAME>
      <INTERNAL-BEHAVIORS>
        <SWC-INTERNAL-BEHAVIOR>
          <SHORT-NAME>IB</SHORT-NAME>
          <RUNNABLES>
            <RUNNABLE-ENTITY>
              <SHORT-NAME>RE</SHORT-NAME>
              <MODE-ACCESS-POINTS>
                <MODE-ACCESS-POINT>
                  <IDENT>
                    <SHORT-NAME>ident</SHORT-NAME>
                  </IDENT>
                </MODE-ACCESS-POINT>
              </MODE-ACCESS-POINTS>
            </RUNNABLE-ENTITY>
          </RUNNABLES>
        </SWC-INTERNAL-BEHAVIOR>
      </INTERNAL-BEHAVIORS>
    </APPLICATION-SW-COMPONENT-TYPE>
    <COMPOSITION-SW-COMPONENT-TYPE>
      <SHORT-NAME>CSCT</SHORT-NAME>
      <COMPONENTS>
          <SW-COMPONENT-PROTOTYPE>
```

```
            <SHORT-NAME>SCP</SHORT-NAME>
            <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-TYPE">/IC_Example/
               ASCT</TYPE-TREF>
          </SW-COMPONENT-PROTOTYPE>
        </COMPONENTS>
      </COMPOSITION-SW-COMPONENT-TYPE>
      <RAPID-PROTOTYPING-SCENARIO>
        <SHORT-NAME>rptScenario</SHORT-NAME>
        <RPT-CONTAINERS>
          <RPT-CONTAINER>
            <SHORT-NAME>rptContainer</SHORT-NAME>
            <BY-PASS-POINT-IREFS>
              <BY-PASS-POINT-IREF>
                <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/
                   IC_Example/CSCT/SCP</CONTEXT-ELEMENT-REF>
                <TARGET-REF DEST="MODE-ACCESS-POINT-IDENT">/IC_Example/ASCT
                   /IB/RE/ident</TARGET-REF>
              </BY-PASS-POINT-IREF>
            </BY-PASS-POINT-IREFS>
          </RPT-CONTAINER>
        </RPT-CONTAINERS>
      </RAPID-PROTOTYPING-SCENARIO>
    </ELEMENTS>
</AR-PACKAGE>
```

Listing 14.1: Example for the definition of an RPT scenario

14.5 Extended Buffer Access Method

The Extended Buffer Access method enhances the support for rapid prototyping (RP) to support the bypass use case where the RTE cannot be regenerated by the bypass user.

The goal is to ensure that all VariableDataPrototypes that are communicated via RTE APIs are written to and read back from an RP global buffer that can be modified by rapid prototyping tools (RPT).

The method applies to all RTE APIs and not just those for implicit access and hence is termed the *extended* buffer access method.

Within the Extended buffer access method, a VariableDataPrototype can be flagged for rapid prototyping at one of three levels depending on whether post-build hooking is used. “Level 1” is intended for use by post-build hooking tools and “Level 2” and “Level 3” by non post-build hooking.

Additional RP buffers and RP flags are created when using “Level 2” and “Level 3” and the Extended Buffer Access method includes mechanisms for describing their creation and use to RP tooling.

- RP global buffer – A buffer read/written by RP. The RP global buffer is conceptually separated from the RTE managed buffer holding the variable data prototype value.
- RP global measurement buffer – A buffer used by RP to store the original variable data prototype value for subsequent measurement purposes before replacement by the RP generated value.
- RP enabler flag – A Boolean flag to permit run-time enabling/disabling bypass.

14.5.1 RP Preparation

The Extended Buffer Access method of Rapid Prototyping requires the definition of *preparation level* (see [constr_2055]) to enable RPT-related code generation.

The RptProfile of category EXTENDED_BUFFER_ACCESS provides the common attributes to implement the RPT support in the code.

An ECU may have to support multiple RptProfiles in parallel – for example, to support in one ECU the RPT tools of different suppliers.

Nevertheless, not all components might need to support all possible methods, for instance an XCP based RPT method might not be applicable for hard real-time critical functions, and therefore the RptProfile can be selected.

[TPS_SWCT_01719] Selection of applicable RptProfiles ⌈The reference RptContainer.explicitRptProfileSelection provides a list of RptProfiles which needs to be applied when the RPT support is implemented.

If the explicitRptProfileSelection is not defined, all RptProfiles defined in the owing RapidPrototypingScenario are applicable.⌋*(RS_SWCT_03280)*

Class	**RptProfile**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	The RptProfile describes the common properties of a Rapid Prototyping method.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	RapidPrototypingScenario.rptProfile			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
maxServicePointId	PositiveInteger	0..1	attr	Highest service point id useable for RTE generated service points.
minServicePointId	PositiveInteger	0..1	attr	Lowest service point id useable for RTE generated service points.
servicePointSymbolPost	CIdentifier	0..1	attr	Complete symbol of the function implementing the post service point. This symbol is used for post-build hooking purposes.
servicePointSymbolPre	CIdentifier	0..1	attr	Complete symbol of the function implementing the pre service point. This symbol is used for post-build hooking purposes.

▽

△

Class	RptProfile			
stimEnabler	RptEnablerImplType Enum	0..1	attr	Defines if the service points support the stimulation enabler. If RptProfile.stimEnabler is "none" then no stimulation enabler is passed to the service function. Otherwise the stimulation enabler will be passed as a parameter.

Table 14.7: RptProfile

[constr_1988] Existence of attribute RptProfile.maxServicePointId ⌈For each RptProfile, attribute maxServicePointId shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1989] Existence of attribute RptProfile.minServicePointId ⌈For each RptProfile, attribute minServicePointId shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1990] Existence of attribute RptProfile.servicePointSymbolPost ⌈For each RptProfile, attribute servicePointSymbolPost shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1991] Existence of attribute RptProfile.servicePointSymbolPre ⌈For each RptProfile, attribute servicePointSymbolPre shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1992] Existence of attribute RptProfile.stimEnabler ⌈For each RptProfile, attribute stimEnabler shall exist **at the time when the RTE is generated**.⌋*()*

Class	RptImplPolicy			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	Describes the code preparation for rapid prototyping at data accesses.			
Base	*ARObject*			
Aggregated by	McDataInstance.rptImplPolicy, RptComponent.rpImplPolicy, RptContainer.rptImplPolicy, RptExecutableEntityEvent.rptImplPolicy			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
rptEnablerImplType	RptEnablerImplType Enum	0..1	attr	For Level 2 or Level3 this property determines how the RTE implements the additional "RP enabler" flag.
rptPreparationLevel	RptPreparationEnum	0..1	attr	Mandates RP preparation level for access to VariableDataPrototype within generated RTE implementation.

Table 14.8: RptImplPolicy

[constr_1993] Existence of attribute RptImplPolicy.rptEnablerImplType ⌈For each RptImplPolicy, attribute rptEnablerImplType shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1994] Existence of attribute RptImplPolicy.rptPreparationLevel ⌈For each RptImplPolicy, attribute rptPreparationLevel shall exist **at the time when the RTE is generated**.⌋*()*

Enumeration	**RptEnablerImplTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Describes the required / implemented usage of enabler flags for data access in the code.
Aggregated by	RptImplPolicy.rptEnablerImplType, RptProfile.stimEnabler
Literal	*Description*
none	No "RP enabler" is implemented. **Tags:**atp.EnumerationLiteralIndex=0
rptEnablerRam	"RP enabler" is implemented as a RAM variable **Tags:**atp.EnumerationLiteralIndex=1
rptEnablerRamAnd Rom	The RTE generator implements both the RAM and ROM "RP enabler". **Tags:**atp.EnumerationLiteralIndex=3
rptEnablerRom	"RP enabler" is implemented as a calibrateable ROM variable. **Tags:**atp.EnumerationLiteralIndex=2

Table 14.9: RptEnablerImplTypeEnum

Enumeration	**RptPreparationEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines the RP preparation level for access to VariableDataPrototypes within the generated RTE implementation.
Aggregated by	RptImplPolicy.rptPreparationLevel
Literal	*Description*
none	No RP preparation for VariableDataPrototype. **Tags:**atp.EnumerationLiteralIndex=0
rptLevel1	The RTE implementation uses an "RP global buffer" for measurement and post-build hooking purposes. **Tags:**atp.EnumerationLiteralIndex=1
rptLevel2	As rpLevel1 but the RTE implementation also uses both "RP enabler flag" to permit RP overwrite at run-time. **Tags:**atp.EnumerationLiteralIndex=2
rptLevel3	As rpLevel2 but the RTE implementation also uses "RP global measurement buffer" to record the original ECU-generated value in addition to the RP value. **Tags:**atp.EnumerationLiteralIndex=3

Table 14.10: RptPreparationEnum

[TPS_SWCT_01720] Preparation Levels ⌈`RptImplPolicy.rptPreparationLevel` supports three preparation levels:

- **Level 1** – If `RptImplPolicy.rptPreparationLevel` is set to `rptLevel1` then the generated RTE uses a specific memory access pattern (a write-read cycle within accessing code created by the RTE generator) suitable for access by post-build hooking tools patch writes to buffers.
- **Level 2** – If `RptImplPolicy.rptPreparationLevel` is set to `rptLevel2` then in addition to the use of an RP global buffer (as for `rptLevel1`) the generated code also includes an RP enabler flag that is used to make update of the RP global buffer conditional.

The RP enabler flag can be in either (calibratable) ROM or RAM based on RptContainer.rptEnablerImplType.

- **Level 3** – If RptImplPolicy.rptPreparationLevel is set to rptLevel3 then in addition to the requirements of rptLevel2, the generated code also records the original ECU-generated value as well as the RP replacement value.

⌋*(RS_SWCT_03280)*

[TPS_SWCT_01721] References from RptContainer ⌈If rptImplPolicy of a RptContainer is used the RptContainer can reference:

- VariableDataPrototype – the preparation level applies to a single data item.
- ArgumentDataPrototype – the preparation level applies to a single operation argument.
- ModeDeclarationGroupPrototype – the preparation level applies to a single mode.
- operation – the preparation level applies to all operation ArgumentDataPrototype.
- RunnableEntity – the preparation level applies to all data items / arguments accessed by the RunnableEntity.
- SwComponentPrototype – the preparation level applies to all RunnableEntitys (and hence all accessed data items and arguments) in the software component.

⌋*(RS_SWCT_03280)*

The generated RTE includes appropriate descriptions to enable RP tools to access the generated RP buffers and RP enabler flags.

Class	**RptSwPrototypingAccess**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Describes the accessibility of data and modes by the rapid prototyping tooling.			
Base	*ARObject*			
Aggregated by	McDataInstance.resultingRptSwPrototypingAccess, RptContainer.rptSwPrototypingAccess			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
rptHookAccess	RptAccessEnum	0..1	attr	The related data element can be modified using a post-build hooking tool. An ENABLED VariableData Prototype is implicitly READABLE/WRITABLE.
rptReadAccess	RptAccessEnum	0..1	attr	The related data element can be used as input for bypass functionality by RP tool. If rptImplPolicy is not specified then RTE generation shall ensure at least suitable MC read points are created.
rptWriteAccess	RptAccessEnum	0..1	attr	The related data element can be used as output for bypass functionality by RP tool. The data element shall be prepared to rptLevel2 and related write service points are present.

Table 14.11: RptSwPrototypingAccess

[constr_1995] Existence of attribute RptSwPrototypingAccess.rptHookAccess ⌈For each RptSwPrototypingAccess, attribute rptHookAccess shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1996] Existence of attribute RptSwPrototypingAccess.rptReadAccess ⌈For each RptSwPrototypingAccess, attribute rptReadAccess shall exist **at the time when the RTE is generated**.⌋*()*

[constr_1997] Existence of attribute RptSwPrototypingAccess.rptWriteAccess ⌈For each RptSwPrototypingAccess, attribute rptWriteAccess shall exist **at the time when the RTE is generated**.⌋*()*

Enumeration	**RptAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines the access rights to a data object with respect to rapid prototyping.
Aggregated by	RptSwPrototypingAccess.rptHookAccess, RptSwPrototypingAccess.rptReadAccess, RptSwPrototypingAccess.rptWriteAccess
Literal	***Description***
enabled	The related data element is accessible by RP tool. **Tags:**atp.EnumerationLiteralIndex=0
none	The related data element is not accessible by RP tool. **Tags:**atp.EnumerationLiteralIndex=1
protected	The data element is known to the RP tool however its usage for RP can be restricted. Use case: limitation based on access rights **Tags:**atp.EnumerationLiteralIndex=2

Table 14.12: RptAccessEnum

14.5.2 Service Points

Access to the RP global buffers and RP global measurement buffers can be implemented by using a service based ECU interface in which an additional RP Service Component, such as an “XCP on CAN” or “XCP on Ethernet” service, is added to the ECU application.

An RP Service Component is an AUTOSAR or vendor specific BSW module providing an RP service, e.g. “XCP on CAN” or “XCP on Ethernet”.

It provides one or more RP Service Function where data is sampled and/or stimulated at an RP Service Point. Each RP Service Function call is passed an RP Service Point ID to identify the service point and enable different invocations to be distinguished.

The integration of the service can be performed pre-build by means of source code based integration, for example, by adding an XCP or custom BSW component, or post-build by patching the binary code of an already compiled ECU image.

In a service based scenario data is sampled and/or stimulated at RP Service Points. During either sampling or stimulation the data is read and/or written from the memory associated with the VariableDataPrototype to/from a local buffer during the execution of the RP Service Point and hence transferred to/from the RP tool.

Within the context of the RTE the data stimulated by the RP Service Points are the RP global buffers and RP global measurement buffers however any data that is measurable is potentially subject to reading.

[TPS_SWCT_01722] Semantics of RP Service Point ⌈An RP Service Point is simply a call of an RP Service Function that is provided by the RP Service Component.⌋*(RS_SWCT_03280)*

[TPS_SWCT_01723] Semantics of RP Service Function ⌈The RP Service Function is responsible for sampling (reading) and stimulating (writing) the bypass data. The action of sampling may then trigger the RP system to perform the bypass (this may involve the communication of the sampled data to an external system for computation) ready for reading when the stimulation occurs.⌋*(RS_SWCT_03280)*

Service points can be either “SWC Internal” (i.e. inserted by the SWC developer) or “RTE assigned”. SWC Internal service points are included in the SWC description (using RapidPrototypingScenario, see below) whereas RTE assigned are created by the RTE generator based on the specification of the SWC.

14.5.2.1 Service Functions

The RP Service Function is responsible for sampling the required data. The sampled data is not passed as parameters – the invocation of the RP Service Function passes an RP Service Point ID as the first parameter of the RP Service Point which is used by the RP Service Component to identify the service point.

[TPS_SWCT_01724] Semantics of RapidPrototypingScenario ⌈A RapidPrototypingScenario aggregates one or more RptContainers and one or more RptProfiles:

- Each RptContainer instance specifies, by reference, one or more element(s) applicable to the service-based access.
- Each RptProfile instance specifies both the name of the RP Service Function (attributes servicePointSymbolPre and servicePointSymbolPost) and the applicable RP Service Point IDs (attributes minServicePointId and maxServicePointId).

⌋*(RS_SWCT_03280)*

The cross-product of the information from the RptContainer and RptProfile within a rapid prototyping scenario is used to construct the service function invocations.

Example: An RapidPrototypingScenario contains a single RptContainer that references an RTEEvent instance, and a single RptProfile with service point symbol attribute MyServiceFunction. As a result, the RTE generator then wraps invocations of the RunnableEntity started by the event with calls to service function MyServiceFunction.

Class	RptExecutableEntityProperties			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	Describes the code preparation for rapid prototyping at ExecutableEntity invocation.			
Base	*ARObject*			
Aggregated by	RptContainer.rptExecutableEntityProperties, RptExecutableEntityEvent.rptExecutableEntityProperties			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
maxRptEventId	PositiveInteger	0..1	attr	Highest RPT event id usable for RTE generated service points. This attribute is relevant, if dedicated id range shall be applied to the ExecutableEntitys of a software component or specific ExecutableEntitys.
minRptEventId	PositiveInteger	0..1	attr	Lowest RPT event id usable for RTE generated service points. This attribute is relevant, if dedicated id range shall be applied to the ExecutableEntitys of a software component or specific ExecutableEntitys.
rptExecutionControl	RptExecutionControlEnum	0..1	attr	This attribute specifies the rapid prototyping control of the executable
rptServicePoint	RptServicePointEnum	0..1	attr	Enables generation of service points by the RTE generator.

Table 14.13: RptExecutableEntityProperties

[constr_1998] Existence of attribute RptExecutableEntityProperties.maxRptEventId ⌈For each RptExecutableEntityProperties, attribute maxRptEventId shall exist **at the time when the RTE is generated**.⌋()

[constr_1999] Existence of attribute RptExecutableEntityProperties.minRptEventId ⌈For each RptExecutableEntityProperties, attribute minRptEventId shall exist **at the time when the RTE is generated**.⌋()

[constr_10000] Existence of attribute RptExecutableEntityProperties.rptExecutionControl ⌈For each RptExecutableEntityProperties, attribute rptExecutionControl shall exist **at the time when the RTE is generated**.⌋()

[constr_10001] Existence of attribute RptExecutableEntityProperties.rptServicePoint ⌈For each RptExecutableEntityProperties, attribute rptServicePoint shall exist **at the time when the RTE is generated**.⌋()

Enumeration	RptExecutionControlEnum
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines rapid prototyping preparation of an ExecutableEntity.
Aggregated by	RptExecutableEntityProperties.rptExecutionControl
Literal	**Description**

▽

△

Enumeration	**RptExecutionControlEnum**
conditional	The ExecutableEntity is only executed when the rapid prototyping disable flag is NOT set. **Tags:**atp.EnumerationLiteralIndex=0
none	The ExecutableEntity is executed without specific rapid prototyping condition. **Tags:**atp.EnumerationLiteralIndex=1

Table 14.14: RptExecutionControlEnum

Enumeration	**RptServicePointEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario
Note	Specifies whether the invocation of ExecutableEntitys due to activation of specific RteEvents/Bsw Events requires the insertion of Service Points.
Aggregated by	RptExecutableEntityProperties.rptServicePoint
Literal	*Description*
enabled	Enables generation of service points by the RTE generator. **Tags:**atp.EnumerationLiteralIndex=0
none	No Service Points are requested. **Tags:**atp.EnumerationLiteralIndex=1

Table 14.15: RptServicePointEnum

A Glossary

Artifact This is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts ([52]).

At a high level, an artifact is represented as a single conceptual file.

AUTOSAR Tool This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an authoring tool, a converter tool, a processor tool or as a combination of those (see separate definitions).

AUTOSAR Authoring Tool An AUTOSAR Tool used to create and modify AUTOSAR XML Descriptions. Example: System Description Editor.

AUTOSAR Converter Tool An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files. Example: ECU Flattener

AUTOSAR Definition This is the definition of parameters which can have values. One could say that the parameter values are Instances of the definitions. But in the meta model hierarchy of AUTOSAR, definitions are also instances of the meta model and therefore considered as a description. Examples for AUTOSAR definitions are: `EcucParameterDef`, `PostBuildVariantCriterion`, `SwSystemconst`.

AUTOSAR XML Description In AUTOSAR this means "filled Template". In fact an AUTOSAR XML description is the XML representation of an AUTOSAR model.

The AUTOSAR XML description can consist of several files. Each individual file represents an AUTOSAR partial model and shall validate successfully against the AUTOSAR XML schema.

AUTOSAR Meta-Model This is an UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is an UML representation of the AUTOSAR templates. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes, UML tags and OCL expressions (object constraint language) are used for defining specific semantics and constraints.

AUTOSAR Meta-Model Tool The AUTOSAR Meta-Model Tool is the tool that generates different views (class tables, list of constraints, diagrams, XML Schema etc.) on the AUTOSAR meta-model.

AUTOSAR Model This is a representation of an AUTOSAR product. The AUTOSAR model represents aspects suitable to the intended use according to the AUTOSAR methodology.

Strictly speaking, this is an instance of the AUTOSAR meta-model. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR meta-model.

AUTOSAR Partial Model In AUTOSAR, the possible partitioning of models is marked in the meta-model by ≪atpSplitable≫. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.

AUTOSAR Processor Tool An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files. Example: RTE Generator

AUTOSAR Specification Element An AUTOSAR Specification Element is a named element that is part of an AUTOSAR specification. Examples: requirement, constraint, specification item, class or attribute in the meta model, methodology, deliverable, methodology activity, model element, bsw module etc.

AUTOSAR Template The term "Template" is used in AUTOSAR to describe the format different kinds of descriptions. The term template comes from the idea, that AUTOSAR defines a kind of form which shall be filled out in order to describe a model. The filled form is then called the description.

In fact the AUTOSAR templates are now defined as a meta-model.

AUTOSAR Validation Tool A specialized `AUTOSAR Tool` which is able to check an AUTOSAR model against the rules defined by a profile.

AUTOSAR XML Schema This is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta-model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.

Blueprint This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta model resp. types, this process is *not* an instantiation.

Instance Generally this is a particular exemplar of a model or of a type.

Life Cycle Life Cycle is the course of development/evolutionary stages of a model element during its life time.

Meta-Model This defines the building blocks of a model. In that sense, a Meta-Model represents the language for building models.

Meta-Data This includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc.

Model A Model is an simplified representation of reality. The model represents the aspects suitable for an intended purpose.

Partial Model This is a part of a model which is intended to be persisted in one particular artifact.

Pattern in GST This is an approach to simplify the definition of the meta model by applying a model transformation. This transformation creates an enhanced model out of an annotated model.

Profile Authoring Support Data Data that is used for efficient authoring of a profile. E.g. list of referable constraints, meta-classes, meta-attributes or other reusable model assets (blueprints)

Profile Authoring Tool A specialized `AUTOSAR Tool` which focuses on the authoring of profiles for data exchange points. It e.g. provides support for the creation of profiles from scratch, modification of existing profiles or composition of existing profiles.

Profile Compatibility Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the compatibility of profiles for data exchange. Note that this compatibility check includes manual compatibility checks by engineers and automated assistance using more formal algorithms.

Profile Consistency Checker Tool A specialized `AUTOSAR Tool` which focuses on checking the consistency of profiles.

Property A property is a structural feature of an object. As an example a “connector” has the properties “receive port” and “send port”

Properties are made variant by the ≪`atpVariation`≫.

Prototype This is the implementation of a role of a type within the definition of another type. In other words a type may contain Prototypes that in turn are typed by "Types". Each one of these prototypes becomes an instance when this type is instantiated.

Type A type provides features that can appear in various roles of this type.

Value This is a particular value assigned to a “Definition”.

Variability Variability of a system is its quality to describe a set of variants. These variants are characterized by variant specific property settings and / or selections. As an example, such a system property selection manifests itself in a particular “receive port” for a connection.

This is implemented using the ≪`atpVariation`≫.

Variant A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no variability anymore with respect to the binding time.

This is implemented using `EvaluatedVariantSet`.

Variation Binding A variant is the result of a variation binding process that resolves the variability of the system by assigning particular values/selections to all the system’s properties.

This is implemented by `VariationPoint`.

Variation Binding Time The variation binding time determines the step in the methodology at which the variability given by a set of variable properties is resolved.

This is implemented by `vh.LatestBindingtime` at the related properties.

Variation Definition Time The variation definition time determines the step in the methodology at which the variation points are defined.

Variation Point A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete variant.

This is implemented by `VariationPoint`.

B Supported Special Use Cases

B.1 Asymmetric Data Transformation between a Software-Component and a Complex Driver

B.1.1 Overview

In this scenario, a SwComponentPrototype typed by an ApplicationSwComponentType needs to communicate with a SwComponentPrototype typed by a ComplexDeviceDriverSwComponentType.

The communication itself is special insofar as it ends in an arbitrary structured data type on the side of the ApplicationSwComponentType and in a flat byte array on the side of the ComplexDeviceDriverSwComponentType.

The communication itself has a client-server nature, where the ApplicationSwComponentType mostly acts as the client and the ComplexDeviceDriverSwComponentType acts as the server.

As a consequence of this set-up, the structured data type somehow has to be serialized into a flat byte array for the call and from flat byte array to structure data type for the returning of the *out* and *inout* arguments as well as the return value.

For a justification of the use case, let's assume that the ComplexDeviceDriverSwComponentType implements an endpoint of a custom communication protocol that needs to interact with the "AUTOSAR world".

This is also the reason why the server implements ClientServerOperations with a flat byte array as the argument. The server (in this case, the ComplexDeviceDriverSwComponentType) itself is completely agnostic of the data, it just represents the endpoint that has to be able to deal with any given data structure.

This means that the ComplexDeviceDriverSwComponentType does not have to be updated and redeployed if the data structures on the side of the ApplicationSwComponentType undergo any changes.

For the ApplicationSwComponentType, on the other hand, there is little motivation to also model the respective data structures as a flat byte array. Usually, the respective data comes from a mixture of internal processing and communication with other software-components.

In other words, using a flat byte array on the side of the ApplicationSwComponentType would mean that the serialization has to be done anyway, potentially inside the implementation of the ApplicationSwComponentType itself.

Specifically, the conversion from the (structured) data type to a flat byte array and back needs to be implemented by a piece of software that is typically generated according to the structure of the data type. This software, however, is **very specific** and only fits to the corresponding data types.

One approach for data serialization in AUTOSAR is the generic concept of data transformation, in the specific case of this scenario a data transformer is needed that does a depth-first serialization over a complex data structure.

This approach is already supported in AUTOSAR by means of the so-called SOME/IP Transformer [53].

This existing concept can be taken over for the implementation of the scenario described in this chapter, albeit with some customizations. For example, the SOME/IP-specific protocol header generated by the SOME/IP Transformer is obviously not relevant for the scenario.

B.1.2 Modeling Aspects

The modeling of this use case shall be explained along a simple example sketched in Figure B.1:

- It is necessary to define two individual ClientServerOperationMappings.
  - One of these (in this example: CSOM1) shall reference a DataTransformation where attribute dataTransformationKind is set to the value DataTransformationKindEnum.asymmetricToByteArray.
  - The other (in this example: CSOM2) shall reference a DataTransformation where attribute dataTransformationKind is set to the value DataTransformationKindEnum.asymmetricFromByteArray.
- CSOM1 shall reference the ClientServerOperation Op1 (on the end of the ApplicationSwComponentType) in the role firstOperation.
- CSOM1 shall reference the ClientServerOperation Op2 (on the end of the ComplexDeviceDriverSwComponentType) in the role secondOperation.
- CSOM2 shall reference the ClientServerOperation Op1 (on the end of the ApplicationSwComponentType) in the role secondOperation.
- CSOM2 shall reference the ClientServerOperation Op2 (on the end of the ComplexDeviceDriverSwComponentType) in the role firstOperation.
- CSOM1 shall aggregate two DataPrototypeMappings in the role argumentMapping
  - The first DataPrototypeMapping shall reference (in the role firstDataPrototype) the ArgumentDataPrototype named In1 of ClientServerOperation OP1 and (in the role secondDataPrototype) the ArgumentDataPrototype named In of ClientServerOperation OP2.
  - The second DataPrototypeMapping shall reference (in the role firstDataPrototype) the ArgumentDataPrototype named In2 of

ClientServerOperation OP1 and (in the role secondDataPrototype) the ArgumentDataPrototype named In of ClientServerOperation OP2.

- CSOM2 shall aggregate two DataPrototypeMappings in the role argumentMapping
  - The first DataPrototypeMapping shall reference (in the role firstDataPrototype) the ArgumentDataPrototype named Out1 of ClientServerOperation OP1 and (in the role secondDataPrototype) the ArgumentDataPrototype named Out of ClientServerOperation OP2.
  - The second DataPrototypeMapping shall reference (in the role firstDataPrototype) the ArgumentDataPrototype named Out2 of ClientServerOperation OP1 and (in the role secondDataPrototype) the ArgumentDataPrototype named Out of ClientServerOperation OP2.

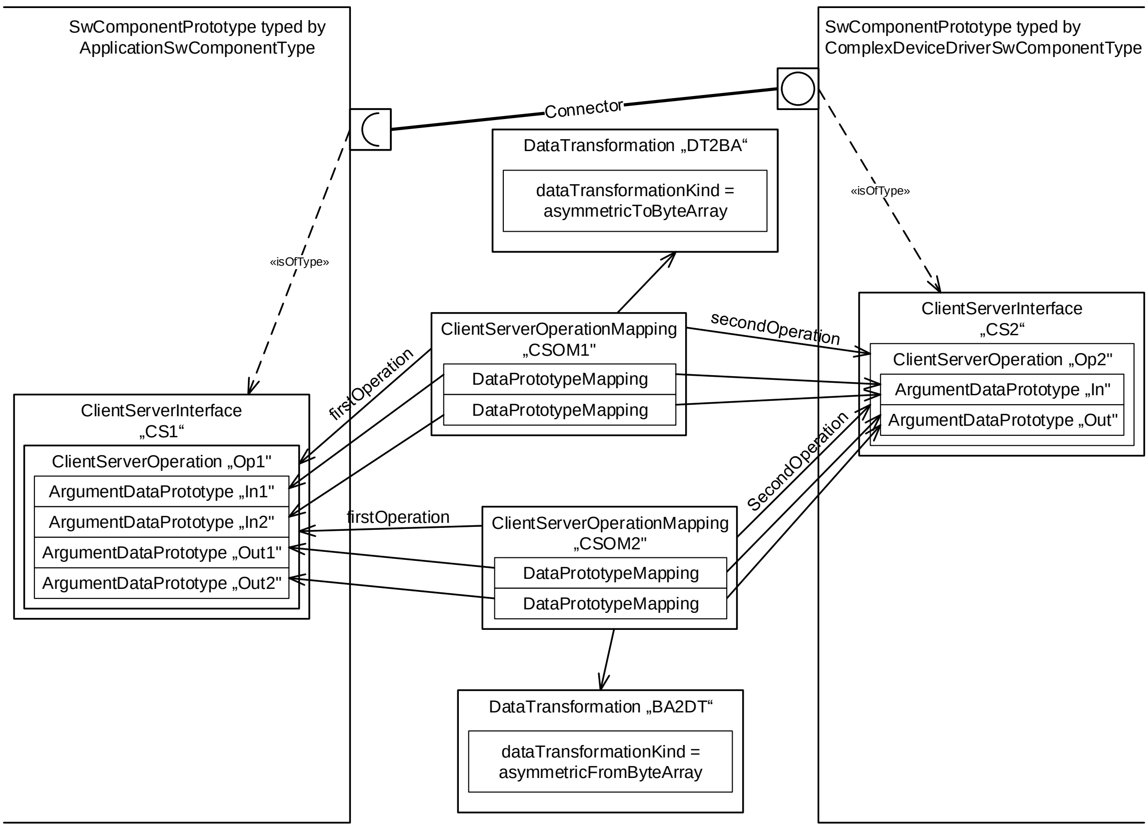


Figure B.1: Modeling of the data transformation use case

Please note that in Figure B.1 the role names of references from DataPrototypeMapping have been left out of the picture for reasons of simplicity.

C History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

C.1 Constraint History of this Document according to AUTOSAR R4.0.1

C.1.1 Changed Constraints in R4.0.1

N/A

C.1.2 Added Constraints in R4.0.1

Number	Heading
[constr_1000]	End-to-end protection is limited to sender/receive communication
[constr_1001]	Value of dataId shall be unique
[constr_1002]	End-to-end protection does not support n:1 communication
[constr_1004]	Mapping of ApplicationDataTypes
[constr_1005]	Compatibility of ImplementationDataTypes mapped to the same ApplicationDataType
[constr_1006]	applicable data categorys
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1008]	Applicability of categorys STRUCTURE and ARRAY
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1010]	If nativeDeclaration does not exist
[constr_1011]	category of SwBaseType
[constr_1012]	Value of category is FIXED_LENGTH
[constr_1013]	Value of category is VARIABLE_LENGTH
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDataDefProps
[constr_1016]	invalidValue is restricted
[constr_1017]	Supported combinations of swImplPolicy and swCalibrationAccess
[constr_1018]	measurementPoint shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess
[constr_1019]	Compatibility of input value and axis
[constr_1020]	ParameterDataPrototype needs to be of compatible data type as referenced in sharedAxisType

▽

△

Number	Heading
[constr_1021]	A CompuMethod shall specify instructions for both directions
[constr_1022]	Limits shall be defined for each direction of CompuMethod
[constr_1023]	Specification of Units in CompuMethods
[constr_1024]	Stepwise definition of CompuMethods
[constr_1025]	Avoid division by zero in rational formula
[constr_1026]	Compatibility of Units
[constr_1027]	Types for record layouts
[constr_1029]	ConstantSpecificationMapping and ConstantSpecification
[constr_1030]	ParameterSwComponentType references ConstantSpecificationMappingSet
[constr_1031]	NvBlockSwComponentType references ConstantSpecificationMappingSet
[constr_1032]	DelegationSwConnector can only connect PortPrototypes of the same kind
[constr_1033]	Communication scenarios for sender/receiver communication
[constr_1035]	Recursive definition of CompositionSwComponentType
[constr_1036]	Connect kinds of PortInterfaces
[constr_1037]	Client may not connect to multiple servers
[constr_1038]	Reference to ApplicationError
[constr_1039]	Relevance of swImplPolicy
[constr_1040]	Conversion of SenderReceiverInterfaces
[constr_1041]	Conversion of ClientServerInterfaces
[constr_1042]	Definition of a linear data scaling
[constr_1043]	PortInterface vs. ComSpec
[constr_1044]	Applicability of DataFilter
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1046]	Applicability of [constr_1045]
[constr_1047]	Compatibility of ApplicationPrimitiveDataTypes
[constr_1048]	Compatibility of ApplicationRecordDataTypes
[constr_1049]	Compatibility of ApplicationArrayDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1051]	Compatibility of SwDataDefProps
[constr_1052]	Compatibility of Units
[constr_1053]	Compatibility of PhysicalDimensions
[constr_1054]	No DataConstr available at the provider
[constr_1055]	ImplementationDataType has category VALUE
[constr_1056]	ImplementationDataType has category TYPE_REFERENCE
[constr_1057]	ImplementationDataType has category DATA_REFERENCE
[constr_1058]	ImplementationDataType has category FUNCTION_REFERENCE
[constr_1059]	Compatibility of data types with category VALUE
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK, or STRING

▽

△

Number	Heading
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_1062]	Compatibility of data types with category BIT
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1064]	Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE or MAP
[constr_1066]	ApplicationDataType is or is not compatible to specific ImplementationDataType
[constr_1067]	ApplicationDataType is or is not compatible to specific ImplementationDataType
[constr_1068]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types
[constr_1069]	Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors
[constr_1070]	Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors
[constr_1071]	compatibility of compatibility of ParameterDataPrototype and VariableDataPrototype
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1075]	Compatibility of ModeDeclarationGroups
[constr_1076]	Compatibility of ArgumentDataPrototypes
[constr_1077]	Compatibility of ApplicationErrors
[constr_1078]	Compatibility of ClientServerOperations
[constr_1079]	Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector
[constr_1080]	Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector
[constr_1081]	Compatibility of TriggerInterfaces in the context of an AssemblySwConnector
[constr_1082]	Compatibility of TriggerInterfaces in the context of an DelegationSwConnector
[constr_1083]	Compatibility of Triggers
[constr_1084]	delegation of an provided outer PortPrototype
[constr_1085]	Compatibility in the case of a flat ECU extract
[constr_1086]	SwConnector between two specific PortPrototypes
[constr_1087]	AssemblySwConnector inside CompositionSwComponentType
[constr_1088]	DelegationSwConnector inside CompositionSwComponentType
[constr_1090]	WaitPoint and RunnableEntity
[constr_1091]	RTEEvents that can unblock a WaitPoint
[constr_1092]	ParameterSwComponentType
[constr_1093]	Definition of textual strings

▽

△

Number	Heading
[constr_1094]	Usage of symbol of RunnableEntity
[constr_1095]	Values of nDataSets vs. reliability
[constr_1096]	SwcModeSwitchEvent and WaitPoint
[constr_1097]	RunnableEntity that has a WaitPoint
[constr_1098]	Mode switch and mode disabling
[constr_1099]	Data type of inter-runnable variables
[constr_1100]	Unconnected RPortPrototype typed by a DataInterface
[constr_1101]	Mode-related communication
[constr_1102]	ApplicationError in the scope of one SwComponentType
[constr_1103]	NonqueuedReceiverComSpec and enableUpdate
[constr_1104]	Trigger sink and trigger source
[constr_1105]	Value of arraySize
[constr_1106]	Structure shall have at least one element
[constr_1107]	Union shall have at least one element
[constr_1108]	Value of ApplicationError.errorCode
[constr_1109]	Mapping of SwComponentPrototypes typed by a SensorActuatorSwComponentType
[constr_1110]	Value of category in EndToEndDescription
[constr_1111]	Constraints of dataId in PROFILE_01
[constr_1112]	Constraints of dataIdMode in PROFILE_01
[constr_1113]	Existence of attributes in PROFILE_01
[constr_1114]	Constraints of crcOffset in PROFILE_01
[constr_1115]	Constraints of counterOffset in PROFILE_01
[constr_1116]	Constraints of dataLength in PROFILE_01
[constr_1117]	Constraints of maxDeltaCounterInit in PROFILE_01
[constr_1118]	Existence of attributes in PROFILE_02
[constr_1119]	Constraints of dataLength in PROFILE_02
[constr_1120]	Constraints of dataId in PROFILE_02
[constr_1121]	Constraints of maxDeltaCounterInit in PROFILE_02
[constr_1122]	Existence of attributes in PROFILE_03
[constr_1123]	Constraints of dataLength in PROFILE_03
[constr_1124]	Constraints of dataId in PROFILE_03
[constr_1125]	Constraints of maxDeltaCounterInit in PROFILE_03
[constr_1126]	Compatibility of DataConstrs
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2001]	Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable
[constr_2002]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReadAccess
[constr_2003]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataWriteAccess

▽

△

Number	Heading
[constr_2004]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataSendPoint
[constr_2005]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument
[constr_2006]	Number of AsynchronousServerCallResultPoint referencing to one AsynchronousServerCallPoint
[constr_2007]	Consistency of typeDefinition attribute
[constr_2009]	Supported kinds of ports of a NvBlockSwComponentType
[constr_2010]	Connections between SwComponentPrototypes of type NvBlockSwComponentType
[constr_2011]	Connections between SwComponentPrototypes typed by NvBlockSwComponentType and SwComponentPrototypes typed by other AtomicSwComponentTypes
[constr_2012]	Compatibility of ImplementationDataTypes used for ramBlock and romBlock
[constr_2013]	Compatibility of ImplementationDataTypes for NvBlockDataMapping
[constr_2014]	Limitation of RoleBasedPortAssignment.role in NvBlockDescriptors
[constr_2015]	Limitation of SwcInternalBehavior of a NvBlockSwComponentType
[constr_2016]	Connections between SwComponentPrototypes of type ServiceProxySwComponentType
[constr_2017]	Ports of ServiceProxySwComponentTypes
[constr_2018]	Supported remote communication of a ServiceProxySwComponentType
[constr_2019]	ServiceSwComponentType shall have service ports only
[constr_2020]	dataReadAccess can not be used for queued communication
[constr_2021]	WaitPoint referencing a DataReceivedEvent can not be used for non-queued communication
[constr_2022]	Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints
[constr_2023]	Consistency of timeout values
[constr_2024]	enableTakeAddress is restricted to single instantiation
[constr_2025]	Uniqueness of symbol attributes
[constr_2026]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role writtenLocalVariable and readLocalVariable
[constr_2027]	SwcServiceDependency shall be defined for service ports only
[constr_2028]	staticMemory is restricted to single instantiation
[constr_2029]	shortName of constantMemory and staticMemory
[constr_2030]	AsynchronousServerCallResultPoint combined with WaitPoint shall belong to the same RunnableEntity
[constr_2031]	Period of TimingEvent shall be greater than 0
[constr_2032]	transmissionAcknowledge requires a DataSendCompletedEvent
[constr_2033]	Timeout of DataSendCompletedEvent
[constr_2500]	PortInterfaces shall be of same kind

▽

△

Number	Heading
[constr_2526]	PortInterfaces need to be compatible to the blueprints
[constr_2527]	Blueprints shall live in package of a proper category
[constr_2528]	PortPrototypes shall not refer to blueprints of PortInterfaces
[constr_2529]	Blueprints of ports and interfaces shall be compatible
[constr_2533]	Iteration along output axis is only supported for VALUE and VAL_BLK
[constr_4000]	Local communication of mode switches
[constr_4001]	Content of ModeRequestTypeMap
[constr_4002]	Unambiguous mapping of modes to data types
[constr_4003]	Semantics of SwcModeSwitchEvent
[constr_4004]	Context of SenderReceiverAnnotation
[constr_4005]	Context of ClientServerAnnotation
[constr_4006]	Context of ParameterPortAnnotation
[constr_4007]	Context of ModePortAnnotation
[constr_4008]	Context of TriggerPortAnnotation
[constr_4009]	Context of NvDataPortAnnotation
[constr_4010]	Context of DelegatedPortAnnotation
[constr_4011]	ComSpec and ModeSwitchedAckEvent
[constr_4012]	Timeout of ModeSwitchedAckEvent
[constr_4035]	ValueSpecification shall fit into data type

Table C.1: Added Constraints in R4.0.1

C.1.3 Deleted Constraints in R4.0.1

N/A

C.2 Constraint History of this Document according to AUTOSAR R4.0.2

C.2.1 Changed Constraints in R4.0.2

Number	Heading
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_2001]	Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable

Table C.2: Changed Constraints in R4.0.2

C.2.2 Added Constraints in R4.0.2

Number	Heading
[constr_1127]	ServiceSwComponentType shall not have ServiceNeeds
[constr_1128]	Queue length of ClientServerOperations associated with the same RunnableEntity
[constr_1129]	swImplPolicy and NonqueuedReceiverComSpec
[constr_1130]	swImplPolicy and NonqueuedReceiverComSpec
[constr_1131]	swImplPolicy and NonqueuedSenderComSpec
[constr_1132]	swImplPolicy and NonqueuedSenderComSpec
[constr_1133]	Identical CompuScale Symbolic Names shall have the same range
[constr_1134]	Allowed structure of TEXTTABLE
[constr_1135]	Limit of vt in BITFIELD_TEXTTABLE
[constr_1136]	Compatibility of introduction of blueprint and blueprinted element
[constr_1137]	Applicability of ParameterInterface
[constr_1138]	assignedPort and DiagEventDebounceMonitorInternal
[constr_1139]	assignedPort of DiagEventDebounceMonitorInternal shall refer to an RPortPrototype
[constr_2034]	SwAddrMethod referenced by RunnableEntitys or BswSchedulableEntitys
[constr_2035]	swImplPolicy for VariableDataPrototype in SenderReceiverInterface
[constr_2036]	swImplPolicy for VariableDataPrototype in NvDataInterface
[constr_2037]	swImplPolicy for VariableDataPrototype in the role ramBlock
[constr_2038]	swImplPolicy for VariableDataPrototype in the role implicitInter-RunnableVariable
[constr_2039]	swImplPolicy for VariableDataPrototype in the role explicitInter-RunnableVariable
[constr_2040]	swImplPolicy for VariableDataPrototype in the role arTypedPerInstance-Memory
[constr_2041]	swImplPolicy for VariableDataPrototype in the role staticMemory
[constr_2042]	swImplPolicy for ParameterDataPrototype in ParameterInterface
[constr_2043]	swImplPolicy for ParameterDataPrototype in the role staticMemory
[constr_2044]	swImplPolicy for ParameterDataPrototype in the role sharedParameter
[constr_2045]	swImplPolicy for ParameterDataPrototype in the role perInstanceParam-eter
[constr_2046]	swImplPolicy for ParameterDataPrototype in the role constantMemory
[constr_2047]	swImplPolicy for ArgumentDataPrototype
[constr_2048]	swImplPolicy for SwServiceArg
[constr_2535]	Target of an autosarParameter in AutosarParameterRef shall refer to a param-eter
[constr_2536]	Target of an autosarVariable in AutosarVariableRef shall refer to a variable

Table C.3: Added Constraints in R4.0.2

C.2.3 Deleted Constraints in R4.0.2

Number	Heading
[constr_1099]	Data type of inter-runnable variables

Table C.4: Deleted Constraints in R4.0.2

C.3 Constraint History of this Document according to AUTOSAR R4.0.3

C.3.1 Changed Constraints in R4.0.3

Number	Heading
[constr_1006]	applicable data categorys
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDataDefProps
[constr_1043]	PortInterface vs. ComSpec
[constr_1051]	Compatibility of SwDataDefProps
[constr_1053]	Compatibility of PhysicalDimensions
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1110]	Value of category in EndToEndDescription
[constr_1113]	Existence of attributes in PROFILE_01
[constr_1118]	Existence of attributes in PROFILE_02
[constr_1134]	Allowed structure of TEXTTABLE
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2027]	SwcServiceDependency shall be defined for service ports only

Table C.5: Changed Constraints in R4.0.3

C.3.2 Added Constraints in R4.0.3

Number	Heading
[constr_1140]	Combination of invalidValue with the attribute handleInvalid
[constr_1141]	Applicability of the scope attribute
[constr_1142]	category of CompuMethod shall not be extended
[constr_1143]	category of AutosarDataType shall not be extended

▽

△

Number	Heading
[constr_1144]	SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType
[constr_1145]	Finding the symbol for the representation of a CompuScale in C code
[constr_1146]	Applicability of a symbol for a CompuScale in C code
[constr_1147]	Standardized values for the attribute category of meta-class PortGroup
[constr_1148]	PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes
[constr_1149]	PortPrototypes used for NV data management
[constr_1150]	Usage of valueType for PortDefinedArgumentValue
[constr_1151]	Applicability of PortInterfaceMapping
[constr_1151]	category of ApplicationArrayElement and AutosarDataType referenced in the role type shall be kept in sync
[constr_1153]	Applicability of compatibility requirements for CompuScales
[constr_1154]	Compatibility of CompuScales for sender-receiver communication and similar use cases
[constr_1155]	Compatibility of CompuScales for client-server communication
[constr_1156]	Relevance of “names” of CompuScales
[constr_1157]	Applicability of constraints of CompuScales
[constr_1158]	Applicable categorys for attribute compuMethod
[constr_1159]	Consistency of VariableAndParameterInterfaceMapping with respect to the referenced DataInterfaces
[constr_1160]	Size of Compound Primitive Data Type is variant
[constr_1161]	Applicability of the index attribute of Ref
[constr_1162]	Compatibility of SwRecordLayouts
[constr_1163]	Compatibility of CompuMethods
[constr_1164]	Number of arguments owned by a RunnableEntity
[constr_1165]	Applicability of RunnableEntityArgument
[constr_1166]	Restrictions of ModeRequestTypeMap
[constr_1167]	ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType
[constr_1168]	Compatibility of ImplementationDataTypes used used in the ModeRequestTypeMap
[constr_1169]	Allowed values for Trigger.swImplPolicy
[constr_1170]	Interpretation of attribute maxDeltaCounterInit owned by EndToEndDescription
[constr_1171]	Interpretation of attribute maxDeltaCounterInit of EndToEndDescription
[constr_1172]	Allowed values of SwCalibrationAccessEnum for ModeDeclarationGroupPrototype
[constr_1173]	Applicability of AutosarParameterRef referencing a VariableDataPrototype
[constr_1174]	PortInterfaces used in the context of CompositionSwComponentTypes cannot refer to AUTOSAR services
[constr_1175]	Depending on its category, CompuMethod shall refer to a unit

▽

△

Number	Heading
[constr_1176]	Compatibility of CompuScales of category LINEAR and RAT_FUNC
[constr_1177]	Allowed category for SwPointerTargetProps
[constr_1178]	Existence of attributes of SwDataDefProps in the context of ImplementationDataType
[constr_1179]	Existence of ModeDeclaration.value within a ModeDeclarationGroup
[constr_1180]	Existence of ModeDeclarationGroup.onTransitionValue
[constr_1181]	Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue
[constr_1182]	Allowed values for InternalTriggeringPoint.swImplPolicy
[constr_1183]	EndToEndProtectionVariablePrototypes aggregated by EndToEndProtection
[constr_1184]	Consistency of rootDataPrototype and base in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1185]	Consistency of data types in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1186]	Consistency of data types in the context of ArVariableInImplementationDataInstanceRef
[constr_1187]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types
[constr_1188]	Existence of ReceiverComSpec.replaceWith
[constr_1189]	Allowed targets of externalReplacement
[constr_1190]	Only one mapping for composite to primitive use case
[constr_2049]	Different ModeDeclarationGroups shall have different shortNames.
[constr_2050]	Mandatory information of a SwAxisCont
[constr_2051]	Mandatory information of a SwValueCont
[constr_2052]	Values of swArraysize and the number of values provided by swValuesPhys shall be consistent.
[constr_2053]	Consistency between role IUMPRNumerator and ObdRatioServiceNeeds.connectionType
[constr_2544]	Limits need to be consistent
[constr_2545]	invalidValue shall fit in the specified ranges
[constr_2548]	Data constraint of value axis shall match
[constr_2549]	Units of input axis shall be consistent
[constr_2550]	Units of value axis shall be consistent
[constr_2551]	SwCalprmAxis.baseType shall be ignored
[constr_2561]	Application of DataConstrRule.constrLevel

Table C.6: Added Constraints in R4.0.3

C.3.3 Added Specification Items in R4.0.3

Number	Heading
[TPS_SWCT_01000]	Usage of attribute symbol of the symbolProps
[TPS_SWCT_01001]	Prefix symbols generated for the RunnableEntity
[TPS_SWCT_01002]	SwComponentTypes may only interact by means of their PortPrototypes
[TPS_SWCT_01003]	Inconsistencies regarding the value of serviceKind and the actual implementation of the PortInterface
[TPS_SWCT_01004]	Default value if serviceKind is not defined
[TPS_SWCT_01005]	Usage of SwcServiceDependencys for vendor-specific services
[TPS_SWCT_01006]	arraySize of ImplementationDataType shall be used to define the size of the array
[TPS_SWCT_01007]	Semantics of array index
[TPS_SWCT_01008]	Definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration
[TPS_SWCT_01009]	The numerical values used to define the values of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue can be arbitrarily defined
[TPS_SWCT_01010]	categorys for the definition of a ModeDeclarationGroup
[TPS_SWCT_01011]	Default category of a ModeDeclarationGroup
[TPS_SWCT_01012]	AtomicSwComponentType reads the current ECU mode (fixed variant)
[TPS_SWCT_01013]	AtomicSwComponentType shall keep the ECU alive (fixed variant)
[TPS_SWCT_01014]	AtomicSwComponentType wants to select a shutdown target (fixed variant)
[TPS_SWCT_01015]	AtomicSwComponentType wants to select a boot target (fixed variant)
[TPS_SWCT_01016]	AtomicSwComponentType wants to select a shutdown target (flexible variant)
[TPS_SWCT_01017]	AtomicSwComponentType wants to select a boot target (flexible variant)
[TPS_SWCT_01018]	AtomicSwComponentType wants to use an alarm clock (flexible variant)
[TPS_SWCT_01019]	AtomicSwComponentType reads the current ComM mode
[TPS_SWCT_01020]	AtomicSwComponentType requests a ComM mode. It may also check later whether the requested ComM mode has become effective
[TPS_SWCT_01021]	AtomicSwComponentType acts as a mode manager that influences the ECU state
[TPS_SWCT_01022]	Queued processing of internal trigger
[TPS_SWCT_01023]	Mapping of elements of composite data types
[TPS_SWCT_01024]	Combination of ApplicationCompositeDataType and nested ImplementationDataType
[TPS_SWCT_01025]	The role of PortPrototypes in the AUTOSAR architecture
[TPS_SWCT_01026]	The role of PortInterfaces in the AUTOSAR architecture
[TPS_SWCT_01027]	Different flavors of PortInterfaces
[TPS_SWCT_01028]	AtomicSwComponentType implements a Diagnostic Monitor

▽

△

Number	Heading
[TPS_SWCT_01029]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_01030]	RunnableEntity
[TPS_SWCT_01031]	ExclusiveArea
[TPS_SWCT_01032]	CompositionSwComponentType
[TPS_SWCT_01033]	Nested definition of CompositionSwComponentTypes
[TPS_SWCT_01034]	CompositionSwComponentTypes do not have any binary footprint
[TPS_SWCT_01035]	CompositionSwComponentType aggregates SwComponentPrototypes
[TPS_SWCT_01036]	SwComponentPrototype implements a specific role
[TPS_SWCT_01037]	arbitrary numbers of SwComponentPrototypes can be created
[TPS_SWCT_01038]	Support for Variant Handling in the in Software Component Template
[TPS_SWCT_01039]	Purpose of variant handling
[TPS_SWCT_01040]	SwConnector exists depending on a *PostBuild* condition
[TPS_SWCT_01041]	API functions of not existing SwConnector are still part of the software-component's implementation
[TPS_SWCT_01042]	Four types of locations in the meta-model which may exhibit variability
[TPS_SWCT_01043]	ApplicationSwComponentTypes are independent from actual ECU Hardware
[TPS_SWCT_01044]	ServiceNeeds
[TPS_SWCT_01045]	Actual values of ECU configuration parameters fulfill the requirements given by the ServiceNeeds
[TPS_SWCT_01046]	ServiceNeeds are defined in the scope of the SwcInternalBehavior
[TPS_SWCT_01047]	Reference from the software representation of a sensor/actuator to the actual hardware element
[TPS_SWCT_01048]	SensorActuatorSwComponentType may use the I/O hardware abstraction directly
[TPS_SWCT_01049]	Two ways to use the ExclusiveAreas
[TPS_SWCT_01050]	RunnableEntity always runs inside an ExclusiveArea
[TPS_SWCT_01051]	RunnableEntity explicitly enters and leaves a specific ExclusiveArea
[TPS_SWCT_01052]	Inter-runnable variable
[TPS_SWCT_01053]	Relationship of interchanged data with RunnableEntitys
[TPS_SWCT_01054]	Semantics of the explicitInterRunnableVariable
[TPS_SWCT_01055]	Semantics of implicitInterRunnableVariable
[TPS_SWCT_01056]	Physical dimension
[TPS_SWCT_01057]	Unit references one physical dimension
[TPS_SWCT_01058]	UnitGroup
[TPS_SWCT_01059]	Exponent for each of the seven fundamental dimensions
[TPS_SWCT_01060]	Negative exponents
[TPS_SWCT_01061]	Conversion of units
[TPS_SWCT_01062]	Documentation of software-components

▽

△

Number	Heading
[TPS_SWCT_01063]	PortGroup
[TPS_SWCT_01064]	PortGroups have to be defined on the VFB level
[TPS_SWCT_01065]	PortPrototype may belong to more than one PortGroups
[TPS_SWCT_01066]	PortGroups can be associated with certain ServiceNeeds
[TPS_SWCT_01067]	Initial mode
[TPS_SWCT_01068]	Units can be grouped with the help of UnitGroup
[TPS_SWCT_01069]	DataInterface is defined as abstract base class
[TPS_SWCT_01070]	PortInterface acts as a *type* for a PortPrototype
[TPS_SWCT_01071]	ModeDeclaration
[TPS_SWCT_01072]	ApplicationDataType and ImplementationDataType
[TPS_SWCT_01073]	Composite ApplicationDataType
[TPS_SWCT_01074]	Composite ImplementationDataType
[TPS_SWCT_01075]	SwcInternalBehavior
[TPS_SWCT_01076]	Number of elements of a specific ApplicationArrayDataType might vary at run-time
[TPS_SWCT_01077]	Configure the response to mode changes
[TPS_SWCT_01078]	Configurable array size
[TPS_SWCT_01079]	SwConnector
[TPS_SWCT_01080]	Delegation ports
[TPS_SWCT_01081]	Implications of being a delegation port
[TPS_SWCT_01082]	AssemblySwConnector
[TPS_SWCT_01083]	DelegationSwConnector
[TPS_SWCT_01084]	Outer PortPrototype is referenced by multiple DelegationSwConnectors
[TPS_SWCT_01085]	Variation on the behavior level
[TPS_SWCT_01086]	Request mode change
[TPS_SWCT_01087]	Propagation of mode information
[TPS_SWCT_01088]	ComSpecs defined by CompositionSwComponentTypes
[TPS_SWCT_01089]	end-to-end communication protection
[TPS_SWCT_01090]	EndToEndProtection
[TPS_SWCT_01091]	Two cases for end-to-end protection
[TPS_SWCT_01092]	EndToEndProtectionSet
[TPS_SWCT_01093]	Definition of end-to-end protection is splitable
[TPS_SWCT_01094]	category of EndToEndDescription
[TPS_SWCT_01095]	category set to NONE
[TPS_SWCT_01096]	PortGroup
[TPS_SWCT_01097]	CompositionSwComponentType cannot have RunnableEntitys
[TPS_SWCT_01098]	Only AtomicSwComponentType can have RunnableEntitys
[TPS_SWCT_01099]	PortInterfaceMapping

▽

△

Number	Heading
[TPS_SWCT_01100]	Precedence of PortInterfaceMapping
[TPS_SWCT_01101]	Unmapped elements of PortInterfaces
[TPS_SWCT_01102]	VariableAndParameterInterfaceMapping
[TPS_SWCT_01103]	Mapping between different kinds of PortInterfaces
[TPS_SWCT_01104]	Possible mappings are restricted by the swImplPolicy
[TPS_SWCT_01105]	ClientServerInterfaceMapping
[TPS_SWCT_01106]	ClientServerOperation
[TPS_SWCT_01107]	swMinAxisPoints and swMaxAxisPoints represent variation points
[TPS_SWCT_01108]	Added value of an AtomicSwComponentType
[TPS_SWCT_01109]	Adding the SwcInternalBehavior in a later process step
[TPS_SWCT_01110]	Symbolic name of a software-component
[TPS_SWCT_01111]	PortPrototypes need an additional model artifact, the PortInterface
[TPS_SWCT_01112]	PortPrototypes are either *require*- or *provide*-ports.
[TPS_SWCT_01113]	Connecting two PortPrototypes
[TPS_SWCT_01114]	SenderReceiverInterface
[TPS_SWCT_01115]	invalidationPolicy
[TPS_SWCT_01116]	swImplPolicy
[TPS_SWCT_01117]	Communication patterns for sender-receiver communication
[TPS_SWCT_01118]	ClientServerInterface
[TPS_SWCT_01119]	Direction of ArgumentDataPrototypes
[TPS_SWCT_01120]	Client needs to provide ArgumentDataPrototypes
[TPS_SWCT_01121]	Pass correct data type
[TPS_SWCT_01122]	Synchronous call of ClientServerOperation
[TPS_SWCT_01123]	No default values for ArgumentDataPrototypes
[TPS_SWCT_01124]	Definition of ArgumentDataPrototypes within the context of a ClientServerOperation is ordered
[TPS_SWCT_01125]	serverArgumentImplPolicy
[TPS_SWCT_01126]	Access to partial networking via BswM
[TPS_SWCT_01127]	Byte arrary with variable size
[TPS_SWCT_01128]	SwRecordLayout needed
[TPS_SWCT_01129]	Express diagnostic capabilities
[TPS_SWCT_01130]	Measurement and calibration access to model elements is defined by swCalibrationAccess
[TPS_SWCT_01131]	AtomicSwComponentType accepts a request to restart an entire function
[TPS_SWCT_01132]	AtomicSwComponentType provides information about operating cycles
[TPS_SWCT_01133]	AtomicSwComponentType provides information about aging cycles
[TPS_SWCT_01134]	AtomicSwComponentType enables storage of DTCs in general
[TPS_SWCT_01135]	AtomicSwComponentType enables storage of subsequent DTCs
[TPS_SWCT_01136]	AtomicSwComponentType retrieves information from the fault storage

▽

△

Number	Heading
[TPS_SWCT_01137]	Dem provides information that the fault storage overflows
[TPS_SWCT_01138]	AtomicSwComponentType suppresses the storage of DTCs within the Dem
[TPS_SWCT_01139]	AtomicSwComponentType informs the Dem that the PTO is active
[TPS_SWCT_01140]	AtomicSwComponentType needs information about specific DTC without being a diagnostic monitor
[TPS_SWCT_01141]	AtomicSwComponentType may have RPortPrototypes typed by an NvDataInterface
[TPS_SWCT_01142]	non-volatile data are provided by a specialized AtomicSwComponentType
[TPS_SWCT_01143]	Non-volatile data represented by an *NvBlockComponent* can be read and written
[TPS_SWCT_01144]	NvBlockDescriptor specifies the properties of exactly one *NvBlock*
[TPS_SWCT_01145]	ramBlock and the romBlock are described by a VariableDataPrototype and a ParameterDataPrototype
[TPS_SWCT_01146]	romBlock is optional
[TPS_SWCT_01147]	No romBlock is configured
[TPS_SWCT_01148]	NvBlockDataMapping
[TPS_SWCT_01149]	RoleBasedPortAssignment of NvBlockDescriptor
[TPS_SWCT_01150]	InternalBehavior of a NvBlockSwComponentType
[TPS_SWCT_01151]	RunnableEntitys do not have further attributes
[TPS_SWCT_01152]	InternalBehavior does not have further attributes
[TPS_SWCT_01153]	IncludedModeDeclarationGroupSet
[TPS_SWCT_01154]	Attribute prefix of IncludedModeDeclarationGroupSet
[TPS_SWCT_01155]	IncludedDataTypeSet
[TPS_SWCT_01156]	Required if the AutosarDataType is not used for any DataPrototype
[TPS_SWCT_01157]	Attribute literalPrefix of IncludedDataTypeSet
[TPS_SWCT_01158]	Three cases for PortInterfaceMapping
[TPS_SWCT_01159]	Mapping is described separately from the SwConnector as reusable ARElement
[TPS_SWCT_01160]	ModeInterfaceMapping
[TPS_SWCT_01161]	TriggerInterfaceMapping
[TPS_SWCT_01162]	Conditional existence of TextTableMapping
[TPS_SWCT_01163]	Conversion from firstValue to secondValue
[TPS_SWCT_01164]	Conversion from secondValue to firstValue
[TPS_SWCT_01165]	Invertible mapping
[TPS_SWCT_01166]	Non-invertible mapping
[TPS_SWCT_01167]	Validity of ModeInterfaceMapping
[TPS_SWCT_01168]	Linear conversion factor can be calculated
[TPS_SWCT_01169]	Support for partial networking
[TPS_SWCT_01170]	Purpose of Virtual Function Cluster
[TPS_SWCT_01171]	Purpose of a control port

▽

△

Number	Heading
[TPS_SWCT_01172]	Requesting and releasing partial networks
[TPS_SWCT_01173]	Control port shall **not** become a part of the PortGroup
[TPS_SWCT_01174]	Status port shall **not** become a member of the PortGroup
[TPS_SWCT_01175]	Actively query the status of a partial network
[TPS_SWCT_01176]	last-is-best semantics for sender-receiver communication
[TPS_SWCT_01177]	Assignment of constant values
[TPS_SWCT_01178]	Specialized subclasses of ValueSpecification
[TPS_SWCT_01179]	Compound Primitive Data Type
[TPS_SWCT_01180]	Maximum possible size of Compound Primitive Data Type
[TPS_SWCT_01181]	Bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst
[TPS_SWCT_01182]	Conceptual levels for the definition of initial values
[TPS_SWCT_01183]	Actual value of an initValue shall be interpreted according to the AutosarDataType
[TPS_SWCT_01184]	ApplicationPrimitiveDataTypes with category VALUE
[TPS_SWCT_01185]	initValues for Compound Primitive Data Types
[TPS_SWCT_01186]	ConstantSpecificationMapping
[TPS_SWCT_01187]	ConstantSpecificationMappingSet referenced by the InternalBehavior
[TPS_SWCT_01188]	Definition of calibration data sets through RTE-generator and compiler
[TPS_SWCT_01189]	DataTypeMap
[TPS_SWCT_01190]	ModeRequestTypeMap
[TPS_SWCT_01191]	mapped ApplicationDataType and ImplementationDataType shall be compatible
[TPS_SWCT_01192]	Meta-classes that have an association to a DataTypeMappingSet
[TPS_SWCT_01193]	Mappings between application and implementation types do not necessarily have to form a 1:1 relation
[TPS_SWCT_01194]	Symbolic name of an ImplementationDataType
[TPS_SWCT_01195]	Mapping of composite element to primitive DataPrototype
[TPS_SWCT_01196]	Semantics of an external trigger event communication
[TPS_SWCT_01197]	TriggerInterface
[TPS_SWCT_01198]	Period for periodic triggering
[TPS_SWCT_01199]	Queued processing of Triggers
[TPS_SWCT_01200]	ModeDeclarationGroupPrototype per ModeSwitchInterface
[TPS_SWCT_01201]	CompositionSwComponentType requires and provides the modes that are required or provided by its contained SwComponentPrototypes
[TPS_SWCT_01202]	ApplicationDataType defines a subset of the values used in the ModeDeclarationGroup
[TPS_SWCT_01203]	PortPrototype may own port annotations
[TPS_SWCT_01204]	GeneralAnnotation
[TPS_SWCT_01205]	Typical annotations for sender/receiver communication

▽

△

Number	Heading
[TPS_SWCT_01206]	Min and Max annotations are valid for a certain amount of time
[TPS_SWCT_01207]	VariableDataPrototypes use the same application-level SenderReceiverAnnotation
[TPS_SWCT_01208]	Grouping for SenderReceiverAnnotation
[TPS_SWCT_01209]	ClientServerAnnotation
[TPS_SWCT_01210]	IoHwAbstractionServerAnnotation
[TPS_SWCT_01211]	Assign several annotations to ArgumentDataPrototype
[TPS_SWCT_01212]	ParameterPortAnnotation
[TPS_SWCT_01213]	ModePortAnnotation
[TPS_SWCT_01214]	TriggerPortAnnotation
[TPS_SWCT_01215]	NvDataPortAnnotation
[TPS_SWCT_01216]	DelegatedPortAnnotation
[TPS_SWCT_01217]	Semantics of DelegatedPortAnnotation.signalFan
[TPS_SWCT_01218]	Big picture of ComSpec
[TPS_SWCT_01219]	ComSpec for queued and non-queued sender-receiver communication
[TPS_SWCT_01220]	initValue defines an initial value that shall be taken if the corresponding dataElement has not yet been received
[TPS_SWCT_01221]	DataFilter
[TPS_SWCT_01222]	Applicability of DataFilter
[TPS_SWCT_01223]	networkRepresentation defines how a specific dataElement is represented on a communication bus
[TPS_SWCT_01224]	CompuMethods of dataElement and the networkRepresentation are used for conversion purposes
[TPS_SWCT_01225]	RunnableEntity implements the functionality of two or more ClientServerOperations
[TPS_SWCT_01226]	initValue on the level of a ComSpec is relevant for connections to the corresponding PortPrototype
[TPS_SWCT_01227]	Unconnected RPortPrototype typed by NvDataInterface
[TPS_SWCT_01228]	NvProvideComSpec
[TPS_SWCT_01229]	Three different levels of abstraction regarding the definition of data types
[TPS_SWCT_01230]	Application Data Level
[TPS_SWCT_01231]	Application level may impose strong requirements on the design of the corresponding implementation level
[TPS_SWCT_01232]	Implementation Data Level
[TPS_SWCT_01233]	Use case for the Implementation Data Level
[TPS_SWCT_01234]	Base Level
[TPS_SWCT_01235]	Mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level
[TPS_SWCT_01236]	Big picture of data types
[TPS_SWCT_01237]	SwDataDefProps
[TPS_SWCT_01238]	Attribute category used in the context of AutosarDataType

▽

△

Number	Heading
[TPS_SWCT_01239]	default value for attribute category used in the context of AutosarDataType
[TPS_SWCT_01240]	Subclasses of ApplicationDataType
[TPS_SWCT_01241]	Applicable categorys for subclasses ApplicationDataType
[TPS_SWCT_01242]	category characterizes the nature of a data type on application level
[TPS_SWCT_01243]	Definition of enumeration types
[TPS_SWCT_01244]	Data types for calibration parameters are also described as primitive types
[TPS_SWCT_01245]	SwDataDefProps control the structure of calibration parameters
[TPS_SWCT_01246]	SwRecordLayout may be required for A2L generation
[TPS_SWCT_01247]	ApplicationArrayDataType and ApplicationRecordDataType
[TPS_SWCT_01248]	Nested definition of ImplementationDataType
[TPS_SWCT_01249]	ApplicationRecordDataType
[TPS_SWCT_01250]	ImplementationDataType has been introduced to optimize the formal support for data type handling on the implementation level
[TPS_SWCT_01251]	Limited set of values for category are applicable for ImplementationDataType
[TPS_SWCT_01252]	ImplementationDataType can express concepts not available on application level
[TPS_SWCT_01253]	Rules apply for the usage of the attribute ImplementationDataType.typeEmitter
[TPS_SWCT_01254]	ImplementationDataType with array semantics
[TPS_SWCT_01255]	Indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time
[TPS_SWCT_01256]	Definition of multi-dimensional array data types
[TPS_SWCT_01257]	ImplementationDataType or the aggregated ImplementationDataTypeElements do not form closed sets
[TPS_SWCT_01258]	Definition of a pointer to data
[TPS_SWCT_01259]	Definition of a pointer to a function
[TPS_SWCT_01260]	SwBaseType
[TPS_SWCT_01261]	Use case for SwBaseType
[TPS_SWCT_01262]	memAlignment and byteOrder are platform specific
[TPS_SWCT_01263]	Further use cases for SwBaseType
[TPS_SWCT_01264]	Data prototypes implement a role of a data type
[TPS_SWCT_01265]	DataPrototype aggregates an own set of SwDataDefProps
[TPS_SWCT_01266]	Three non-abstract classes derived from AutosarDataPrototype
[TPS_SWCT_01267]	DataPrototype can be aggregated in different roles
[TPS_SWCT_01268]	Definition of initValue for a VariableDataPrototype or a ParameterDataPrototype
[TPS_SWCT_01269]	In PortInterfaces, initial values defined for DataPrototypes are ignored
[TPS_SWCT_01270]	AutosarVariableRef
[TPS_SWCT_01271]	AutosarParameterRef
[TPS_SWCT_01272]	Semantics of swComparisonVariable

▽

△

Number	Heading
[TPS_SWCT_01273]	Precedence rules for the application of SwDataDefProps
[TPS_SWCT_01274]	SwDataDefProps used to support calibration and measurement
[TPS_SWCT_01275]	values of the attribute swImplPolicy are restricted depending on the context
[TPS_SWCT_01276]	Computation methods
[TPS_SWCT_01277]	Computation methods are used for the conversion of *internal* values into their *physical* representation and vice versa
[TPS_SWCT_01278]	CompuMethods can also be used to assign symbolic names to internal values
[TPS_SWCT_01279]	Preferred conversion direction depends on the use case
[TPS_SWCT_01280]	CompuMethod applied to values outside of its limits
[TPS_SWCT_01281]	Unit associated with a PhysicalDimension
[TPS_SWCT_01283]	Rational function
[TPS_SWCT_01284]	CompuScale might require a representation in the generated RTE C code
[TPS_SWCT_01285]	Physical dimension
[TPS_SWCT_01286]	DataConstr
[TPS_SWCT_01287]	Standard limits and extended limits in the ASAM-MCD2 (ASAP2) specification
[TPS_SWCT_01288]	Interpretation of PhysConstrs and InternalConstrs by tools
[TPS_SWCT_01289]	Semantics of Limit
[TPS_SWCT_01290]	SwAddrMethod
[TPS_SWCT_01291]	Association of MemorySection with SwAddrMethod
[TPS_SWCT_01292]	Usage of SwAddrMethod in the context of a DataPrototype
[TPS_SWCT_01293]	RTE Generator has to derive the Memory Allocation Keyword
[TPS_SWCT_01294]	Missing SwDataDefProps.swAddrMethod
[TPS_SWCT_01295]	SwRecordLayout
[TPS_SWCT_01296]	Different approaches of ASAM MCD-2MC and AUTOSAR with respect to SwRecordLayout
[TPS_SWCT_01297]	Compliance of ApplicationDataTypes or ImplementationDataTypes to swDataDefProps
[TPS_SWCT_01298]	Computing SwRecordLayout from ImplementationDataTypes is not possible
[TPS_SWCT_01299]	Relation of swRecordLayoutGroup to subElement
[TPS_SWCT_01300]	Relationship between record layouts and interpolation routines
[TPS_SWCT_01301]	Importance of initial values
[TPS_SWCT_01302]	Semantics of minimumStartInterval
[TPS_SWCT_01303]	symbol attribute describes the RunnableEntity's entry point
[TPS_SWCT_01304]	Cat. 1A and 1B RunnableEntitys will eventually terminate
[TPS_SWCT_01305]	RunnableEntity as one that cannot be invoked concurrently
[TPS_SWCT_01306]	Software-component description itself does not put any bounds on the number of concurrent invocations of a RunnableEntity
[TPS_SWCT_01307]	supportsMultipleInstantiation vs. canBeInvokedConcurrently

▽

△

Number	Heading
[TPS_SWCT_01308]	Combination of supportsMultipleInstantiation=false and canBeInvokedConcurrently=false
[TPS_SWCT_01309]	signature of a RunnableEntity depends on the connected RTEEvent
[TPS_SWCT_01310]	Categories of RunnableEntitys
[TPS_SWCT_01311]	Name of an operation argument
[TPS_SWCT_01312]	RunnableEntity has a mapping to BswModuleEntry
[TPS_SWCT_01313]	Conditions for a transition from suspended to to be started
[TPS_SWCT_01314]	RTEEvent
[TPS_SWCT_01315]	Interaction of RunnableEntity with RTEEvent
[TPS_SWCT_01316]	Abstract base class RTEEvent
[TPS_SWCT_01317]	RTE triggers RunnableEntity in response to occurring RTEEvent
[TPS_SWCT_01318]	RunnableEntity and WaitPoint
[TPS_SWCT_01319]	RTEEvent can be used to trigger WaitPoints in different RunnableEntitys
[TPS_SWCT_01320]	RunnableEntitys of category 2
[TPS_SWCT_01321]	Communication among RunnableEntitys
[TPS_SWCT_01322]	Interaction patterns for the application of the sender-receiver paradigm
[TPS_SWCT_01323]	Read and write access to a dataElement
[TPS_SWCT_01324]	Mode switches need to be completed in finite time
[TPS_SWCT_01325]	Read and write access is only applicable for RunnableEntitys of category 1
[TPS_SWCT_01326]	Constrain the scope of a specific communication
[TPS_SWCT_01327]	RTE generator can omit the creation of checks at run-time
[TPS_SWCT_01328]	Default value of attribute scope
[TPS_SWCT_01329]	Access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles
[TPS_SWCT_01330]	RunnableEntity can also have dataSendPoints
[TPS_SWCT_01331]	dataWriteAccess vs. dataSendPoint
[TPS_SWCT_01332]	dataReceivePointByValue vs. dataReceivePointByArgument
[TPS_SWCT_01333]	dataReceivePointByValue/dataReceivePointByArgument vs. dataReadAccess
[TPS_SWCT_01334]	RunnableEntitys of category 1 may have dataReceivePointByValues/ dataReceivePointByArguments
[TPS_SWCT_01335]	Combine dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint
[TPS_SWCT_01336]	dataSendPoint also allows for the definition of a DataSendCompletedEvent
[TPS_SWCT_01337]	DataReceivedEvent
[TPS_SWCT_01338]	DataReceiveErrorEvent
[TPS_SWCT_01339]	RTE activates RunnableEntity in response to DataReceiveErrorEvent
[TPS_SWCT_01340]	DataReceiveErrorEvent cannot be combined with a WaitPoint

▽

△

Number	Heading
[TPS_SWCT_01341]	DataReceiveErrorEvent is directly associated with the corresponding VariableDataPrototype
[TPS_SWCT_01342]	Invocation of a server operation
[TPS_SWCT_01343]	Synchronous vs. asynchronous invocation
[TPS_SWCT_01344]	Consistency of values of timeout
[TPS_SWCT_01345]	Synchronous operation invocation
[TPS_SWCT_01346]	Asynchronous operation invocation
[TPS_SWCT_01347]	Blocking access to operation result in an asynchronous operation invocation
[TPS_SWCT_01348]	Trigger source
[TPS_SWCT_01349]	Trigger sink
[TPS_SWCT_01350]	Calibration Parameters shared among several SwComponentTypes
[TPS_SWCT_01351]	Access to a ParameterDataPrototype
[TPS_SWCT_01352]	Requested mode is just sent and received as an ordinary data value
[TPS_SWCT_01353]	RunnableEntitys react on a mode request via a corresponding RTEEvent
[TPS_SWCT_01354]	PortAPIOption
[TPS_SWCT_01355]	enableTakeAddress = true
[TPS_SWCT_01356]	indirectAPI option switches the generation of the RTE's indirect API functionality
[TPS_SWCT_01357]	Definition of implicit values that are passed by the RTE to the server's entry point
[TPS_SWCT_01358]	Values are hidden from the client components
[TPS_SWCT_01359]	Private memory per instance
[TPS_SWCT_01360]	Arbitrary number of per-instance memory blocks
[TPS_SWCT_01361]	attribute supportsMultipleInstantiation == false
[TPS_SWCT_01362]	Initialization of PerInstanceMemory
[TPS_SWCT_01363]	PerInstanceMemory typed by 'C' Data Types
[TPS_SWCT_01364]	Initial value of a PerInstanceMemory typed by 'C' Data Types
[TPS_SWCT_01365]	PerInstanceMemory typed by AUTOSAR Data Types
[TPS_SWCT_01366]	Initial value of a PerInstanceMemory typed by AUTOSAR Data Types
[TPS_SWCT_01367]	Typed by AUTOSAR data type vs. typed by C data type
[TPS_SWCT_01368]	Describe static and constant memory
[TPS_SWCT_01369]	Static and constant memory is not instantiated by the RTE
[TPS_SWCT_01370]	VariationPointProxy
[TPS_SWCT_01371]	VariationPointProxy vs. VariationPoint
[TPS_SWCT_01372]	bindingTime = preCompileTime
[TPS_SWCT_01373]	RTE generator shall evaluate the SwSystemconstDependentFormula
[TPS_SWCT_01374]	Implementation of AutosarParameterRef
[TPS_SWCT_01375]	Implementation of AutosarVariableRef
[TPS_SWCT_01376]	Software-components need to be capable of reacting to state changes

▽

△

Number	Heading
[TPS_SWCT_01377]	Two mechanisms to define how SwcInternalBehavior should interact with the mode management
[TPS_SWCT_01378]	AtomicSwComponentType can define an SwcModeSwitchEvent to execute RunnableEntity
[TPS_SWCT_01379]	AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode
[TPS_SWCT_01380]	Mode management behavior on the sender side
[TPS_SWCT_01381]	Read the currently active mode
[TPS_SWCT_01382]	Mode switch requests are handled asynchronously by the RTE
[TPS_SWCT_01383]	ModeSwitchPoint
[TPS_SWCT_01384]	Execution of initialization code for software-components
[TPS_SWCT_01385]	Execution of initialization code for software-components
[TPS_SWCT_01386]	Initialization by mode management
[TPS_SWCT_01387]	Finalization by mode management
[TPS_SWCT_01388]	Initial modes of AtomicSwComponentTypes are defined by the initialMode
[TPS_SWCT_01389]	I/O Hardware Abstraction interfaces MCAL drivers
[TPS_SWCT_01390]	I/O Hardware Abstraction might have sub-structures
[TPS_SWCT_01391]	I/O Hardware Abstraction abstracts from the location of peripheral I/O devices
[TPS_SWCT_01392]	Mapping between the EcuAbstractionSwComponentType and the corresponding BswModuleDescription
[TPS_SWCT_01393]	Complex Driver
[TPS_SWCT_01394]	Complex Driver is represented by the ComplexDeviceDriverSwComponentType
[TPS_SWCT_01395]	ComplexDeviceDriverSwComponentType has dependencies to ECU Hardware
[TPS_SWCT_01396]	Mapping between the ComplexDeviceDriverSwComponentType and the corresponding BswModuleDescription
[TPS_SWCT_01397]	Hybrid concept between Basic Software Modules and a SwComponentType
[TPS_SWCT_01398]	Communication patterns for AUTOSAR services
[TPS_SWCT_01399]	Dependency is modeled by aggregating required and provided PortPrototypes
[TPS_SWCT_01400]	PortInterface selected from the set of standardized Service Interfaces
[TPS_SWCT_01401]	Form a top-level RootSwCompositionPrototype
[TPS_SWCT_01402]	Mapping of all AtomicSwComponentType instances to EcuInstances
[TPS_SWCT_01403]	Impact of AUTOSAR services on the methodology
[TPS_SWCT_01404]	Creation of the EcuExtract
[TPS_SWCT_01405]	Creation of the ServiceSwComponentTypes
[TPS_SWCT_01406]	Creation of SwComponentPrototype typed by a ServiceSwComponentType

▽

△

Number	Heading
[TPS_SWCT_01407]	Creation of InternalBehavior typed by a ServiceSwComponentType
[TPS_SWCT_01408]	Creation of SwcBswMapping
[TPS_SWCT_01409]	Update of PortDefinedArgumentValues
[TPS_SWCT_01410]	Dcm and Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface
[TPS_SWCT_01411]	Use cases for a ServiceSwComponentType to express ServiceNeeds
[TPS_SWCT_01412]	ServiceSwComponentType shall be added in ECU Configuration phase
[TPS_SWCT_01413]	Local communication with services
[TPS_SWCT_01414]	Mode manager needs to communicate with application software components located on other ECUs
[TPS_SWCT_01415]	Interfaces of ServiceProxySwComponentType
[TPS_SWCT_01416]	Difference between a ServiceProxySwComponentType and an ApplicationSwComponentType
[TPS_SWCT_01417]	Define calibration parameters common to all SwComponentPrototypes of the same SwComponentType
[TPS_SWCT_01418]	Ways to define a calibration parameter
[TPS_SWCT_01419]	ParameterSwComponentType shall never aggregate a SwcInternalBehavior
[TPS_SWCT_01420]	SwComponentType requiring access to shared calibration parameters needs RPortPrototype typed by a ParameterInterface
[TPS_SWCT_01421]	ParameterInterface is not restricted to parameters which can actually can be calibrated
[TPS_SWCT_01422]	Delegation of PortPrototypes typed by a ParameterInterface
[TPS_SWCT_01423]	ParameterDataPrototype aggregated in the role constantMemory
[TPS_SWCT_01424]	ParameterDataPrototype aggregated in the role perInstanceParameter
[TPS_SWCT_01425]	AtomicSwComponentType provides one callback per event if diagnostic event data change
[TPS_SWCT_01426]	AtomicSwComponentType provides callback if any diagnostic event data and/or status changed
[TPS_SWCT_01427]	AtomicSwComponentType provides data for diagnostic purposes via ClientServerInterface
[TPS_SWCT_01428]	ServiceSwComponentType representing the Dem provides a PPortPrototype for the Dcm
[TPS_SWCT_01429]	[constr_1135] only applies for BITFIELD_TEXTTABLE
[TPS_SWCT_01430]	Conversion specification from internal to physical values as well as the reverse conversion
[TPS_SWCT_01431]	Finding the symbol for the representation of a CompuScale in C code
[TPS_SWCT_01432]	Keep the invalidValue transparent to the sending and receiving software components
[TPS_SWCT_01433]	Invalid values outside the range limits
[TPS_SWCT_01434]	Sender and receiver have knowledge of invalid value
[TPS_SWCT_01435]	Invalid values outside the range limits

▽

△

Number	Heading
[TPS_SWCT_01436]	Different receivers require different handling of data invalidation
[TPS_SWCT_01437]	invalidValue can also be specified without setting a compuMethod
[TPS_SWCT_01438]	Handling of invalidation in the sending RTE
[TPS_SWCT_01439]	Handling of invalidation in the receiving RTE
[TPS_SWCT_01440]	Measurement is not limited to primitive objects
[TPS_SWCT_01441]	Nature of a TYPE_REFERENCE
[TPS_SWCT_01442]	ImplementationDataType of category TYPE_REFERENCE does not define own properties
[TPS_SWCT_01443]	ImplementationDataType of category TYPE_REFERENCE overwrites properties of refined ImplementationDataType
[TPS_SWCT_01444]	Size of SwBaseType is specified in bits
[TPS_SWCT_01445]	Applicability of SwDataDefProps for DataPrototypes
[TPS_SWCT_01446]	References to a DataPrototype may or may not imply the necessity for using an instanceRef
[TPS_SWCT_01447]	Applicable binding times for model elements in the scope of the Software Component Template
[TPS_SWCT_01448]	Pre-defined values for the category of VariationPointProxy
[TPS_SWCT_02000]	Default value for attribute swImplPolicy
[TPS_SWCT_02001]	Values of SwAxisCont with the category COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_02002]	AtomicSwComponentType offers a PPortPrototypes typed by ClientServerInterface to read/write current value via diagnostic services
[TPS_SWCT_02003]	AtomicSwComponentType offers PortPrototypes typed by Sender-ReceiverInterfaces to read/write current values via diagnostic services
[TPS_SWCT_02004]	AtomicSwComponentType offers a PortPrototype typed by a ClientServerInterface to start/stop or request routine results of diagnostic routines
[TPS_SWCT_02005]	AtomicSwComponentType offers PortPrototypes typed by ClientServerInterfaces to adjust the IO signal via diagnostic services
[TPS_SWCT_02006]	AtomicSwComponentType offers sender receiver ports to adjust the IO signal via diagnostic services
[TPS_SWCT_02007]	AtomicSwComponentType implements a OBD system monitor with In-Use-Monitor Performance Ratio
[TPS_SWCT_02008]	AtomicSwComponentType offers a server port to read/write current value via OBD services
[TPS_SWCT_02009]	AtomicSwComponentType offers sender receiver ports to read/write current values via OBD services
[TPS_SWCT_02010]	AtomicSwComponentType offers a server port to read vehicle information values via OBD services
[TPS_SWCT_02011]	AtomicSwComponentType offers a server port to read DTR value via OBD services
[TPS_SWCT_02012]	AtomicSwComponentType offers a server port for request control of on-board system, test or component via OBD services

▽

△

Number	Heading
[TPS_SWCT_02013]	AtomicSwComponentType offers a server port to get protocol, session and security information or to request a Reset to Default Session
[TPS_SWCT_02014]	AtomicSwComponentType supports Response On Event (ROE) via diagnostic services
[TPS_SWCT_02015]	AtomicSwComponentType verifies the access to security level via diagnostic services
[TPS_SWCT_02016]	AtomicSwComponentType requires information on the status of the protocol communication and may disallow a protocol
[TPS_SWCT_02017]	AtomicSwComponentType requires the notification about a Service Request via diagnostic services
[TPS_SWCT_02018]	Setup for AtomicSwComponentType which contains a Supervised Entity
[TPS_SWCT_02019]	Setup for AtomicSwComponentType which requires *Global Supervision Status* notification
[TPS_SWCT_02020]	AtomicSwComponentType uses the hash calculation of the Crypto Service
[TPS_SWCT_02021]	AtomicSwComponentType uses the message authentication code (MAC) calculation of the Crypto Service
[TPS_SWCT_02022]	AtomicSwComponentType uses the message authentication code (MAC) verification of the Crypto Service
[TPS_SWCT_02023]	AtomicSwComponentType uses the generation of random seed of the Crypto Service
[TPS_SWCT_02024]	AtomicSwComponentType uses the generation of random numbers of the Crypto Service
[TPS_SWCT_02025]	AtomicSwComponentType uses the symmetrical block encryption of the Crypto Service
[TPS_SWCT_02026]	AtomicSwComponentType uses the symmetrical block decryption of the Crypto Service
[TPS_SWCT_02027]	AtomicSwComponentType uses the symmetrical encryption of the Crypto Service
[TPS_SWCT_02028]	AtomicSwComponentType uses the symmetrical decryption of the Crypto Service
[TPS_SWCT_02029]	AtomicSwComponentType uses the asymmetrical encryption of the Crypto Service
[TPS_SWCT_02030]	AtomicSwComponentType uses the asymmetrical decryption of the Crypto Service
[TPS_SWCT_02031]	AtomicSwComponentType uses the signature generation of the Crypto Service
[TPS_SWCT_02032]	AtomicSwComponentType uses the signature verification of the Crypto Service
[TPS_SWCT_02033]	AtomicSwComponentType uses the checksum calculation of the Crypto Service
[TPS_SWCT_02034]	AtomicSwComponentType uses the key derivation of the Crypto Service
[TPS_SWCT_02035]	AtomicSwComponentType uses the symmetric key derivation of the Crypto Service
[TPS_SWCT_02036]	AtomicSwComponentType uses the key exchange interface for public value calculation of the Crypto Service

▽

△

Number	Heading
[TPS_SWCT_02037]	AtomicSwComponentType uses the key exchange interface for secret value calculation of the Crypto Service
[TPS_SWCT_02038]	AtomicSwComponentType uses the key exchange interface to calculate symmetric key with the Crypto Service
[TPS_SWCT_02039]	AtomicSwComponentType uses the symmetrical key extraction of the Crypto Service
[TPS_SWCT_02040]	AtomicSwComponentType uses the symmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a symmetric key
[TPS_SWCT_02041]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a asymmetric key
[TPS_SWCT_02042]	AtomicSwComponentType uses the asymmetrical public key extraction of the Crypto Service
[TPS_SWCT_02043]	AtomicSwComponentType uses the asymmetrical private key extraction of the Crypto Service
[TPS_SWCT_02044]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a symmetrical wrapping key
[TPS_SWCT_02045]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a asymmetrical wrapping key

Table C.7: Added Specification Items in 4.0.3

C.3.4 Deleted Constraints in R4.0.3

Number	Heading
[constr_1023]	Specification of Units in CompuMethods (the text is still there but it does no longer represent a constraint)
[constr_1062]	Compatibility of data types with category BIT
[constr_1122]	Existence of attributes in PROFILE_03
[constr_1123]	Constraints of dataLength in PROFILE_03
[constr_1124]	Constraints of dataId in PROFILE_03
[constr_1125]	Constraints of maxDeltaCounterInit in PROFILE_03
[constr_1127]	ServiceSwComponentType shall not have ServiceNeeds
[constr_1136]	Compatibility of introduction of blueprint and blueprinted element
	The following constraints are moved to [1]
[constr_2500]	PortInterfaces shall be of same kind
[constr_2526]	PortInterfaces need to be compatible to the blueprints
[constr_2527]	Blueprints shall live in package of a proper category
[constr_2528]	PortPrototypes shall not refer to blueprints of PortInterfaces

▽

△

Number	Heading
[constr_2529]	Blueprints of ports and interfaces shall be compatible
[constr_4001]	Content of ModeRequestTypeMap

Table C.8: Deleted Constraints in R4.0.3

C.3.5 Deleted Specification Items in R4.0.3

N/A

C.4 Constraint History of this Document according to AUTOSAR R4.1.1

C.4.1 Changed Constraints in R4.1.1

Number	Heading
[constr_1012]	Value of category is FIXED_LENGTH
[constr_1013]	Value of category is VARIABLE_LENGTH
[constr_1016]	Restriction of invalidValue for ImplementationDataType and ImplementationDataTypeElement
[constr_1026]	Compatibility of Units
[constr_1047]	Compatibility of ApplicationPrimitiveDataTypes
[constr_1048]	Compatibility of ApplicationRecordDataTypes
[constr_1049]	Compatibility of ApplicationArrayDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1075]	Compatibility of ModeDeclarationGroups
[constr_1079]	Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector
[constr_1080]	Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector
[constr_1081]	Compatibility of TriggerInterfaces in the context of an AssemblySwConnector
[constr_1082]	Compatibility of TriggerInterfaces in the context of an DelegationSwConnector

▽

△

Number	Heading
[constr_1068]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types
[constr_1069]	Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors
[constr_1070]	Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1079]	Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector
[constr_1080]	Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector
[constr_1081]	Compatibility of TriggerInterfaces in the context of an AssemblySwConnector
[constr_1082]	Compatibility of TriggerInterfaces in the context of an DelegationSwConnector
[constr_1108]	Value of ApplicationError.errorCode
[constr_1177]	Allowed targetCategory for SwPointerTargetProps
[constr_1187]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types

Table C.9: Changed Constraints in R4.1.1

C.4.2 Added Constraints in R4.1.1

Number	Heading
[constr_1191]	Value of Limit shall yield a numerical value
[constr_1192]	Compatibility of "IDENTICAL" to "RAT_FUNC" or "LINEAR"
[constr_1193]	ModeDeclaration shall be referenced by at least one ModeTransition in the role enteredMode
[constr_1194]	Identical ModeTransitions
[constr_1195]	SwcModeSwitchEvent and the definition of ModeTransition
[constr_1196]	Existence of networkRepresentation vs. compositeNetworkRepresentation
[constr_1197]	Existence of compositeNetworkRepresentation shall be comprehensive
[constr_1200]	Queued communication is not applicable for dataElements owned by PRPortPrototype
[constr_1201]	initValue shall exist in an RPortPrototype

▽

△

Number	Heading
[constr_1202]	Supported connections by AssemblySwConnector for PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
[constr_1203]	Supported connections by DelegationSwConnector for PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
[constr_1204]	Supported connections by AssemblySwConnector for PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface
[constr_1205]	Supported connections by DelegationSwConnector for PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface
[constr_1209]	Mapping of ModeDeclarations of mode user to ModeDeclaration of mode manager
[constr_1210]	Mapping of ModeDeclarations of mode user to **all** ModeDeclarations of mode manager
[constr_1211]	Constraints of maxNoNewOrRepeatedData in PROFILE_01
[constr_1212]	Constraints of syncCounterInit in PROFILE_01
[constr_1213]	Constraints of maxNoNewOrRepeatedData in PROFILE_02
[constr_1214]	Constraints of syncCounterInit in PROFILE_02
[constr_1215]	Interpretation of attribute maxNoNewOrRepeatedData owned by EndToEndDescription in PROFILE_01
[constr_1216]	Interpretation of attribute syncCounterInit owned by EndToEndDescription in PROFILE_01
[constr_1217]	Interpretation of attribute maxNoNewOrRepeatedData owned by EndToEndDescription in PROFILE_02
[constr_1218]	Interpretation of attribute syncCounterInit owned by EndToEndDescription in PROFILE_02
[constr_1219]	Invalidation depends on the value of swImplPolicy
[constr_1220]	Compatibility of SwBaseType
[constr_1221]	DataPrototype is typed by an ApplicationPrimitiveDataType
[constr_1222]	category of an AutosarDataType used to type a DataPrototype is set to STRING
[constr_1223]	DataPrototype is typed by an ApplicationRecordDataType
[constr_1224]	DataPrototype is typed by an ApplicationArrayDataType
[constr_1225]	DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE
[constr_1226]	Applicable range for ExecutableEntityActivationReason.bitPosition
[constr_1227]	Value of attribute ExecutableEntityActivationReason.bitPosition shall be unique
[constr_1228]	RTEEvent that is referenced by a WaitPoint in the role trigger shall not reference ExecutableEntityActivationReason
[constr_1229]	category of ImplementationDataType boils down to VALUE
[constr_1230]	ApplicationDataType that qualifies for Integral Primitive Type
[constr_1231]	ConsistencyNeeds aggregated by CompositionSwComponentType
[constr_1232]	ConsistencyNeeds aggregated by AtomicSwComponentType

▽

△

Number	Heading
[constr_1233]	InstantiationTimingEventProps shall only reference TimingEvent
[constr_1234]	Value of RunnableEntity.symbol
[constr_1237]	Scope of mapped ClientServerOperations in the context of a ClientServerOperationMapping
[constr_1238]	Scope of mapped ApplicationErrors in the context of a ClientServerOperationMapping
[constr_1239]	RuleBasedValueSpecification shall not exceed the number of values required
[constr_1240]	Consistency of ArgumentDataPrototypes within the context of a ClientServerOperationMapping
[constr_1241]	Compound Primitive Data Types and invalidValue
[constr_1242]	Restriction of invalidValue for ApplicationPrimitiveDataType
[constr_1243]	NumericalOrText shall either define vf or vt
[constr_1244]	DataPrototypes used in application software shall not be typed by C enums
[constr_1245]	Consideration of ModeTransitions for the compatibility of ModeDeclarationGroups
[constr_1246]	Consistency of firstMode and secondMode in the scope of one ModeDeclarationMappingSet
[constr_1247]	Consistency of ModeDeclarationMappingSet with respect to the referenced firstModeGroup and secondModeGroup
[constr_1248]	Compatibility of PortPrototypes of different DataInterfaces in the context of a PassThroughSwConnector
[constr_1249]	Compatibility of ModeSwitchInterfaces in the context of a PassThroughSwConnector
[constr_1250]	Compatibility of ClientServerInterfaces in the context of a PassThroughSwConnector
[constr_1251]	Compatibility of PortPrototypes of TriggerInterfaces in the context of a PassThroughSwConnector
[constr_1252]	Creation of a loop involving a PassThroughSwConnector is not allowed
[constr_1253]	Supported usage of VariationPointProxy
[constr_1254]	Definition of a pointer to a pointer
[constr_1255]	ApplicationPrimitiveDataTypes of category BOOLEAN and STRING
[constr_1256]	Acknowledgement feedback in n:1 writer case
[constr_1257]	No WaitPoints allowed
[constr_1258]	Value of minimumStartInterval for RunnableEntitys triggered by an InitEvent
[constr_1259]	Aggregation of AsynchronousServerCallPoint **and** AsynchronousServerCallResultPoint
[constr_1260]	No mode disabling for InitEvents
[constr_1261]	Applicability for EndToEndDescription.dataIdNibbleOffset
[constr_1263]	Existence of ModeErrorBehavior.defaultMode
[constr_1264]	Iteration along output axis is only supported for VALUE and VAL_BLK
[constr_1268]	ArgumentDataPrototype.direction shall be preserved in a ClientServerOperationMapping

▽

△

Number	Heading
[constr_1269]	Number of arguments shall be preserved in a ClientServerOperationMapping
[constr_1270]	ArgumentDataPrototype shall be mapped only once in a ClientServerOperationMapping
[constr_1271]	ArrayValueSpecification.elements shall be identical to the number of ApplicationRecordDataType.element
[constr_1272]	ArrayValueSpecification.elements shall be identical to the number of subElements of ImplementationDataType of category STRUCTURE
[constr_1273]	ArrayValueSpecification.elements shall be identical to the value of ApplicationArrayDataType.element.maxNumberOfElements
[constr_1274]	ArrayValueSpecification.elements shall be identical to the value of ImplementationDataType.subElement.arraySize of category ARRAY
[constr_2054]	Valid targets of rptSystem
[constr_2055]	Valid targets of byPassPoint and rptHook reference
[constr_2056]	Consistency of RapidPrototypingScenario with respect to rptSystem and rptArHook references
[constr_2057]	Mandatory information of a RuleBasedAxisCont
[constr_2058]	Mandatory information of a RuleBasedValueCont
[constr_4082]	RunnableEntity.reentrancyLevel shall not be set.

Table C.10: Added Constraints in R4.1.1

C.4.3 Changed Specification Items in R4.1.1

Number	Heading
[TPS_SWCT_01000]	Usage of attribute symbol of the symbolProps
[TPS_SWCT_01001]	Prefix symbols generated for the RunnableEntity
[TPS_SWCT_01085]	Variation on the behavior level
[TPS_SWCT_01112]	Semantics of PortPrototypes
[TPS_SWCT_01113]	Connecting two PortPrototypes
[TPS_SWCT_01128]	SwRecordLayout needed for ApplicationPrimitiveDataType of category STRING
[TPS_SWCT_01179]	Compound Primitive Data Type
[TPS_SWCT_01368]	Describe static and constant memory

Table C.11: Changed Specification Items in R4.1.1

C.4.4 Added Specification Items in R4.1.1

Number	Heading
[TPS_SWCT_01448]	Pre-defined values for the category of VariationPointProxy
[TPS_SWCT_01449]	Semantics of a ModeDeclarationGroupPrototypeMapping
[TPS_SWCT_01450]	Semantics of a ModeTransition
[TPS_SWCT_01451]	Relations between ModeTransition and ModeDeclaration
[TPS_SWCT_01452]	Applicability of networkRepresentation for ApplicationCompositeDataType
[TPS_SWCT_01454]	PRPortPrototype can own both RPortComSpecs and PPortComSpecs
[TPS_SWCT_01455]	Duplicate existence of initValue in the context of a PRPortPrototype
[TPS_SWCT_01456]	Predefined values for MemorySection.option and SwAddrMethod.option
[TPS_SWCT_01457]	ExclusiveAreaNestingOrder
[TPS_SWCT_01458]	Indicate that the locking behavior is fully described for RunnableEntity
[TPS_SWCT_01459]	Locking behavior is not described for this RunnableEntity
[TPS_SWCT_01460]	Relation of SynchronousServerCallPoint to ExclusiveAreaNestingOrder
[TPS_SWCT_01461]	Existence of ImplementationDataType
[TPS_SWCT_01462]	ModeDeclarationMapping defines the explicit correlation of ModeDeclarations
[TPS_SWCT_01463]	ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet defines the applicable set of ModeDeclarationMappings
[TPS_SWCT_01464]	ModeDeclaration of a mode user is mapped to exactly one ModeDeclaration of a mode manager
[TPS_SWCT_01465]	ModeDeclaration of a mode user is mapped to several ModeDeclarations of a mode manager
[TPS_SWCT_01466]	ConsistencyNeeds applied on RunnableEntitys that do not use implicit communication
[TPS_SWCT_01467]	ImplementationDataType references an SwBaseType with a string encoding
[TPS_SWCT_01469]	RTE API for retrieving the current activation reason
[TPS_SWCT_01470]	RunnableEntityGroup
[TPS_SWCT_01471]	DataPrototypeGroup
[TPS_SWCT_01472]	Receiving SwComponentType owns a DataPrototypeGroup in the role regRequiresStability
[TPS_SWCT_01473]	Receiving SwComponentType owns a RunnableEntityGroup in the role regRequiresStability
[TPS_SWCT_01474]	Receiving SwComponentType owns a RunnableEntityGroup in the role regRequiresStability and also owns one or several DataPrototypeGroups in the role regRequiresStability
[TPS_SWCT_01475]	Sending SwComponentType owns a DataPrototypeGroup in the role regRequiresStability

▽

△

Number	Heading
[TPS_SWCT_01476]	Sender and receiver of the same implicitly communicated VariableDataPrototypes are associated with the same RunnableEntityGroup
[TPS_SWCT_01477]	Integral Primitive Types
[TPS_SWCT_01478]	Array size is defined as an attribute of the ImplementationDataTypeElement
[TPS_SWCT_01479]	Applicability of ConsistencyNeeds
[TPS_SWCT_01480]	*Stability* and/or *coherence* is not required
[TPS_SWCT_01481]	The meaning of the term *stability* with respect to ConsistencyNeeds
[TPS_SWCT_01482]	The meaning of the term *coherence* with respect to ConsistencyNeeds
[TPS_SWCT_01483]	Use static and constant memory to support Measurement and Calibration
[TPS_SWCT_01484]	Meaning of ApplicationRuleBasedValueSpecification
[TPS_SWCT_01485]	RuleBasedValueSpecification shall initialize first elements in an array
[TPS_SWCT_01486]	ApplicationPrimitiveDataType of category STRING may have invalidValue
[TPS_SWCT_01487]	Correspondence of invalidValue for ApplicationPrimitiveDataType and ImplementationDataType
[TPS_SWCT_01488]	ApplicationPrimitiveDataType shall be interpreted as a string of a particular encoding
[TPS_SWCT_01489]	Standardized values of SwRecordLayoutV.swRecordLayoutVProp
[TPS_SWCT_01490]	AUTOSAR supports ApplicationErrors only for ClientServerInterfaces
[TPS_SWCT_01491]	AUTOSAR system does not need to explicitly describe infrastructure errors
[TPS_SWCT_01492]	Default values for factorSiToUnit and offsetSiToUnit
[TPS_SWCT_01493]	The number of RuleArguments.arguments shall not exceed the array size
[TPS_SWCT_01494]	A RuleBasedValueSpecification of rule FILL_UNITIL_END shall fill the value of the last RuleArguments.argument until the last element of the array
[TPS_SWCT_01495]	Standardized value of RuleBasedValueSpecification.category
[TPS_SWCT_01496]	General precedence rule for attributes of SwDataDefProps
[TPS_SWCT_01497]	Precedence of the unit of value axis
[TPS_SWCT_01498]	Precedence of the DataConstr of value axis
[TPS_SWCT_01499]	Precedence of the CompuMethod of value axis
[TPS_SWCT_01500]	Precedence of the display format of value axis
[TPS_SWCT_01501]	Precedence of the calibration access of value axis
[TPS_SWCT_01502]	Precedence of the Unit of the input axis
[TPS_SWCT_01503]	Precedence of the DataConstr of the input axis
[TPS_SWCT_01504]	Precedence of the display format of the input axis
[TPS_SWCT_01505]	Precedence of calibration access along structure hierarchies in complex types
[TPS_SWCT_01506]	Precedence of the calibration access of input axis
[TPS_SWCT_01507]	The role of PassThroughSwConnector
[TPS_SWCT_01508]	Scope of end-to-end protection

▽

△

Number	Heading
[TPS_SWCT_01509]	Implicit communication behavior
[TPS_SWCT_01510]	The role of pretended networking
[TPS_SWCT_01511]	Configuration option is encoded into ModeDeclaration
[TPS_SWCT_01512]	Request change of Pretended Networking mode
[TPS_SWCT_01513]	React on the change of Pretended Networking mode
[TPS_SWCT_01514]	Duplicate existence of initValue in the context of a PRPortPrototype
[TPS_SWCT_01515]	PPortInCompositionInstanceRef shall be used for attaching DelegationSwConnector to an inner PRPortPrototype
[TPS_SWCT_01516]	PortInterface describes the static structure of information interchange
[TPS_SWCT_01517]	ClientServerOperation cannot be passed as a reference
[TPS_SWCT_01518]	Priority of initial value definition with respect to conceptual levels
[TPS_SWCT_01519]	RTE executes certain RunnableEntity periodically
[TPS_SWCT_01520]	Implication of the existence of possibleError on compatibility of ClientServerOperations
[TPS_SWCT_01521]	Use AutosarVariableRef.localVariable for referencing inter-runnable variables
[TPS_SWCT_01522]	No initial value is specified for implicitInterRunnableVariable or explicitInterRunnableVariable
[TPS_SWCT_01523]	Internal trigger event
[TPS_SWCT_01524]	Usage of IoHwAbstractionServerAnnotation
[TPS_SWCT_01525]	InitEvent references a RunnableEntity in the role startOnEvent
[TPS_SWCT_01528]	Meaning of NumericalRuleBasedValueSpecification
[TPS_SWCT_01529]	Default value for EndToEndDescription.dataIdNibbleOffset
[TPS_SWCT_01530]	Error behavior of mode manager and mode user
[TPS_SWCT_01531]	The semantics of ModeErrorReactionPolicyEnum
[TPS_SWCT_01532]	The role of ModeErrorBehavior.defaultMode
[TPS_SWCT_01533]	ModeDeclarationGroup.initialMode shall be assumed in the absence of ModeDeclarationGroup.modeManagerErrorBehavior
[TPS_SWCT_01534]	ModeDeclarationGroup.initialMode shall be assumed in the absence of ModeDeclarationGroup.modeUserErrorBehavior
[TPS_SWCT_01535]	Mode manager reacts on mode error
[TPS_SWCT_01536]	Coherent behavior of all mode users in case of errors in the mode switch communication
[TPS_SWCT_01541]	Preferential selection of modeUserErrorBehavior
[TPS_SWCT_01542]	Preferential selection of modeManagerErrorBehavior
[TPS_SWCT_01543]	PortInterfaceMapping overrides all other compatibility rules
[TPS_SWCT_01544]	*prefix* used for the actual name of the used PortInterface for the routing activation
[TPS_SWCT_01545]	ModeDeclaration of a *mode user* that is not mapped to a ModeDeclaration of a *mode manager*
[TPS_SWCT_01546]	Notification when an external tester is attached or activated

▽

△

Number	Heading
[TPS_SWCT_01547]	Ability to set and reset the Warning Indicator Status bit
[TPS_SWCT_02046]	byPassPoint specifies the rapid prototyping capability
[TPS_SWCT_02047]	RptHook specifies the link to rapid prototyping algorithm
[TPS_SWCT_02048]	Implicit SwComponentPrototype selection for Rapid Prototyping Scenario
[TPS_SWCT_02049]	Implicit RunnableEntity selection for Rapid Prototyping Scenario
[TPS_SWCT_02050]	Explicit access point selection for Rapid Prototyping Scenario
[TPS_SWCT_02051]	Explicit DataPrototype selection for Rapid Prototyping Scenario
[TPS_SWCT_02052]	Definition of Rapid Prototyping Scenario is splittable
[TPS_SWCT_02053]	Values of RuleBasedAxisCont with the category COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_02507]	Instantiation-specific RTEEvents

Table C.12: Added Specification Items in 4.1.1

C.4.5 Deleted Constraints in R4.1.1

Number	Heading
[constr_1239]	RuleBasedValueSpecification shall not exceed the number of values required
[constr_1145]	Finding the symbol for the representation of a CompuScale in C code
[constr_2533]	Iteration along output axis is only supported for VALUE and VAL_BLK

Table C.13: Deleted Constraints in R4.1.1

Please note that [constr_2533] has been retagged to [constr_1264] to fix a duplicate constraint ID.

C.4.6 Deleted Specification Items in R4.1.1

Number	Heading
[TPS_SWCT_01210]	IoHwAbstractionServerAnnotation

Table C.14: Deleted Specification Items in R4.1.1

C.5 Constraint History of this Document according to AUTOSAR R4.1.2

C.5.1 Changed Constraints in R4.1.2

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1015]	Prioritization of SwDataDefProps
[constr_1043]	PortInterface vs. ComSpec
[constr_1058]	ImplementationDataType has category FUNCTION_REFERENCE
[constr_1102]	ApplicationError in the scope of one SwComponentType
[constr_1146]	Applicability of a symbol for a CompuScale in C code
[constr_1163]	Compatibility of CompuMethods
[constr_1133]	Identical CompuScale Symbolic Names shall have the same range
[constr_1158]	Applicable categorys for attribute ImplementationDataType.swDataDef-Props.compuMethod
[constr_1225]	DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE
[constr_2013]	Compatibility of ImplementationDataTypes for NvBlockDataMapping
[constr_2051]	Mandatory information of a SwValueCont

Table C.15: Changed Constraints in R4.1.2

C.5.2 Added Constraints in R4.1.2

Number	Heading
[constr_1277]	SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue
[constr_1278]	PhysConstrs references a Unit
[constr_1279]	Unmapped elements of ApplicationCompositeDataTypes or Implementa-tionDataTypes and the attribute swImplPolicy
[constr_1280]	Unmapped dataElement on the receiver side shall have an initValue
[constr_1281]	invalidValue shall be inside the scope of the compuMethod
[constr_1282]	Restriction concerning the usage of RuleBasedValueSpecification or a Ref-erenceValueSpecification for the specification of an invalidValue
[constr_1283]	invalidValue is outside the scope of the compuMethod
[constr_1284]	Limitation of the use of TextValueSpecification
[constr_1285]	Applicability of roles vs. PortPrototypes

▽

△

Number	Heading
[constr_1286]	serverArgumentImplPolicy and ArgumentDataPrototype typed by primitive data types
[constr_1287]	Compatibility of SenderReceiverInterfaces with respect to invalidationPolicy
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_1290]	Limitation on the number of PPortComSpecs in the context of one PPortPrototype
[constr_1291]	Limitation on the number of RPortComSpecs in the context of one PPortPrototype
[constr_1292]	Limitation on the number of RPortComSpecs/PPortComSpecs in the context of one PRPortPrototype
[constr_1293]	Existence of DiagnosticEventNeeds.dtcNumber
[constr_1294]	Existence of DiagnosticEventInfoNeeds.dtcNumber
[constr_1295]	PortInterfaces and category DATA_REFERENCE
[constr_1296]	DataPrototypes used as explicitInterRunnableVariable or implicitInterRunnableVariable and category DATA_REFERENCE

Table C.16: Added Constraints in R4.1.2

C.5.3 Changed Specification Items in R4.1.2

Number	Heading
[TPS_SWCT_01006]	ImplementationDataType.subElement.arraySize shall be used to define the size of the array
[TPS_SWCT_01175]	Actively query the status of a partial network
[TPS_SWCT_01219]	ComSpec for queued and non-queued sender-receiver communication
[TPS_SWCT_01154]	Attribute prefix of IncludedModeDeclarationGroupSet
[TPS_SWCT_02011]	AtomicSwComponentType offers a server port to read DTR value via OBD services

Table C.17: Changed Specification Items in R4.1.2

C.5.4 Added Specification Items in R4.1.2

Number	Heading
[TPS_SWCT_01549]	Definition of linear data scaling
[TPS_SWCT_01550]	Definition of reciprocal linear data scaling
[TPS_SWCT_01551]	Mapping of elements on the sender side to elements on the receiver side
[TPS_SWCT_01552]	Software-component acts as a mode manager
[TPS_SWCT_01553]	Software-component acts as a mode user
[TPS_SWCT_01554]	Software-component acts as a mode requester
[TPS_SWCT_01555]	ModeSwitchedAckEvent is triggered by the RTE regardless
[TPS_SWCT_01556]	Rule for setting RoleBasedPortAssignment.role
[TPS_SWCT_01557]	dataWriteAccess also allows for the definition of a DataWriteCompletedEvent
[TPS_SWCT_01558]	DataWriteCompletedEvent cannot be combined with a WaitPoint
[TPS_SWCT_01559]	Default value for attribute SwDataDefProps.swCalibrationAccess
[TPS_SWCT_01560]	Supported categorys of CompuMethods for data conversion
[TPS_SWCT_01561]	Application of data conversion to composite AutosarDataTypes
[TPS_SWCT_01562]	Specification of values of an enumeration
[TPS_SWCT_01563]	Applicable values for nativeDeclaration
[TPS_SWCT_01564]	Non-recursive definition of a primitive data type
[TPS_SWCT_01565]	Recursive definition of a primitive data type
[TPS_SWCT_01566]	Define literals for an MCD system in the context of a FlatInstanceDescriptor
[TPS_SWCT_01567]	Default behavior for invalidationPolicy
[TPS_SWCT_01568]	Consideration of RPortComSpec or PPortComSpec depending on the ownership
[TPS_SWCT_01569]	Definition of CompuScale Symbolic Name
[TPS_SWCT_01570]	DataTypeMap is mandatory in the presence of ApplicationPrimitiveDataType.swDataDefProps.swRecordLayout

Table C.18: Added Specification Items in 4.1.2

C.5.5 Deleted Constraints in R4.1.2

Number	Heading
[constr_1042]	Definition of a linear data scaling
[constr_1094]	Usage of symbol of RunnableEntity
[constr_2025]	Uniqueness of symbol attributes (This constraint has been relocated to [10])
[constr_2032]	transmissionAcknowledge requires a DataSendCompletedEvent
[constr_4011]	ComSpec and ModeSwitchedAckEvent

Table C.19: Deleted Constraints in R4.1.2

C.5.6 Deleted Specification Items in R4.1.2

Number	Heading
[TPS_SWCT_01327]	RTE generator can omit the creation of checks at run-time

Table C.20: Deleted Specification Items in R4.1.2

C.6 Constraint History of this Document according to AUTOSAR R4.1.3

C.6.1 Added Traceables in R4.1.3

Number	Heading
[TPS_SWCT_01573]	A PRPortPrototype is never considered unconnected
[TPS_SWCT_01574]	PerInstanceMemory.typeDefinition shall not contain a function pointer

Table C.21: Added Traceabless in 4.1.3

C.6.2 Changed Traceables in R4.1.3

Number	Heading
[TPS_SWCT_01009]	The numerical values used to define the values of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue can be arbitrarily defined
[TPS_SWCT_01049]	Two ways to use the ExclusiveAreas

Table C.22: Changed Traceables in R4.1.3

C.6.3 Deleted Traceables in R4.1.3

Number	Heading
[TPS_SWCT_01417]	Define calibration parameters common to all SwComponentPrototypes of the same SwComponentType
[TPS_SWCT_01419]	ParameterSwComponentType shall never aggregate a SwcInternalBehavior
[TPS_SWCT_02006]	AtomicSwComponentType offers sender receiver ports to adjust the IO signal via diagnostic services

Table C.23: Deleted Traceables in R4.1.3

C.6.4 Added Constraints in R4.1.3

Number	Heading
[constr_1297]	Applicability of serverArgumentImplPolicy set to useArrayBaseType
[constr_1298]	Existence of attributes if category of a ModeDeclarationGroup is set to EXPLICIT_ORDER
[constr_1299]	Existence of attributes if category of a ModeDeclarationGroup is set to other than EXPLICIT_ORDER

Table C.24: Added Constraints in R4.1.3

C.6.5 Changed Constraints in R4.1.3

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1230]	ApplicationDataType that qualifies for Integral Primitive Type

Table C.25: Changed Constraints in R4.1.3

C.6.6 Deleted Constraints in R4.1.3

Number	Heading
[constr_1179]	Existence of ModeDeclaration.value within a ModeDeclarationGroup
[constr_1180]	Existence of ModeDeclarationGroup.onTransitionValue

Table C.26: Deleted Constraints in R4.1.3

C.7 Constraint History of this Document according to AUTOSAR R4.2.1

C.7.1 Added Traceables in R4.2.1

Number	Heading
[TPS_SWCT_01572]	Application Mode Manager interacts with both BswM and other ApplicationSwComponentTypes
[TPS_SWCT_01577]	AtomicSwComponentType requires the notification about a Service Request via diagnostic services with *manufacturer* characteristics
[TPS_SWCT_01578]	AtomicSwComponentType requires the notification about a Service Request via diagnostic services with *supplier* characteristics
[TPS_SWCT_01579]	Dcm can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface
[TPS_SWCT_01580]	Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface
[TPS_SWCT_01581]	Communication patterns for mode-related communication
[TPS_SWCT_01582]	Semantics of DiagnosticEventNeeds.deferringFid
[TPS_SWCT_01583]	Completeness of TextTableMapping is not a requirement
[TPS_SWCT_01584]	InternalBehavior of a NvBlockSwComponentType for implementing a writing strategy
[TPS_SWCT_01585]	Relevance of NvBlockDescriptor.timingEvent.period
[TPS_SWCT_01586]	Writing strategies for *nv data*
[TPS_SWCT_01587]	The cyclic writing of *nv data* requires the existence of a TimingEvent
[TPS_SWCT_01588]	DataReceivedEvent for storing *nv data* immediately
[TPS_SWCT_01589]	Implementation of emergency storing of *nv data*
[TPS_SWCT_01590]	Combination of writing strategies for *nv data* is possible
[TPS_SWCT_01591]	Existence of attribute DiagnosticEventNeeds.reportBehavior
[TPS_SWCT_01592]	Communication among RunnableEntitys of different instances of the same AtomicSwComponentType
[TPS_SWCT_01593]	Semantics of attribute ReceiverComSpec.transformationComSpecProps
[TPS_SWCT_01594]	Semantics of TransformationComSpecProps
[TPS_SWCT_01595]	Semantics of attribute ClientComSpec.transformationComSpecProps
[TPS_SWCT_01596]	Semantics of attribute ServerComSpec.transformationComSpecProps
[TPS_SWCT_01597]	PortPrototype-specific data transformation configuration
[TPS_SWCT_01598]	More than one user-defined transformer is used within one transformer chain
[TPS_SWCT_01599]	PortPrototype-specific configuration for custom transformers
[TPS_SWCT_01600]	PortPrototype-specific configuration for data transformers related to end-to-end protection
[TPS_SWCT_01601]	Size Indicator shall be updated by software-component

▽

△

Number	Heading
[TPS_SWCT_01602]	Size Indicator shall be read by the software-component
[TPS_SWCT_01603]	Variable size array with Size Indicator
[TPS_SWCT_01604]	Enable Size Indicator
[TPS_SWCT_01605]	Semantics of ApplicationArrayElement.arraySizeHandling
[TPS_SWCT_01606]	Internal structure of mapped ImplementationDataType
[TPS_SWCT_01607]	Profiles for internal structure of mapped ImplementationDataType
[TPS_SWCT_01608]	Custom profiles for internal structure of mapped ImplementationDataType
[TPS_SWCT_01609]	A RuleBasedValueSpecification of rule FILL_UNTIL_MAX_SIZE shall fill the value of the last RuleArguments.argument until the number of elements specified in maxSizeToFill
[TPS_SWCT_01610]	Modeling of a Variable-Size Array Data Type with Size Indicator enabled
[TPS_SWCT_01611]	Enable Size Indicator
[TPS_SWCT_01612]	arraySizeHandling specifies how the size is determined
[TPS_SWCT_01613]	Internal structure of mapped ImplementationDataType
[TPS_SWCT_01614]	Profiles for internal structure of mapped ImplementationDataType
[TPS_SWCT_01615]	Custom profiles for internal structure of mapped ImplementationDataType
[TPS_SWCT_01616]	Semantics of TransformerHardErrorEvent
[TPS_SWCT_01617]	Structure of an ImplementationDataType that represents a variable-sized array data type
[TPS_SWCT_01618]	Size Indicator for dynamicArraySizeProfile set to VSA_LINEAR, VSA_SQUARE, or VSA_FULLY_FLEXIBLE
[TPS_SWCT_01619]	Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR
[TPS_SWCT_01620]	Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR
[TPS_SWCT_01621]	Payload for dynamicArraySizeProfile
[TPS_SWCT_01622]	Modeling of a Variable-Size Array Data Type only with ImplementationDataType
[TPS_SWCT_01623]	Justification for the existence of attributes ApplicationArrayDataType.dynamicArraySizeProfile and ApplicationArrayElement.arraySizeHandling
[TPS_SWCT_01624]	Hard error occurs during the execution of a transformer chain
[TPS_SWCT_01625]	Sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency and also RunnableEntityGroups
[TPS_SWCT_01626]	Error notification of data transformer errors
[TPS_SWCT_01627]	Suffix used for the resulting name of the PortInterface for the Security Access
[TPS_SWCT_01628]	Suffix used for the resulting name of the PortInterface for the Data Services
[TPS_SWCT_01629]	Suffix used for the resulting name of the PortInterface for the Data Services

▽

△

Number	Heading
[TPS_SWCT_01630]	Suffix used for the resulting name of the PortInterface for the Data Services
[TPS_SWCT_01631]	Suffix used for the resulting name of the PortInterface for the Infotype Services
[TPS_SWCT_01632]	Suffix used for the resulting name of the PortInterface for the Routine Services
[TPS_SWCT_01633]	Suffix used for the resulting name of the PortInterface for the Request Control Services
[TPS_SWCT_01634]	Suffix used for the resulting name of the PortInterface for the Data Services
[TPS_SWCT_01635]	Naming conventions may support the effectiveness of SymbolProps
[TPS_SWCT_01636]	Definition of profiles for the definition of Variable-Size Array Data Types

Table C.27: Added Traceables in 4.2.1

C.7.2 Changed Traceables in R4.2.1

Number	Heading
[TPS_SWCT_01044]	ServiceNeeds
[TPS_SWCT_01053]	Relationship of interchanged data with RunnableEntitys
[TPS_SWCT_01141]	AtomicSwComponentType may have AbstractRequiredPortPrototypes typed by an NvDataInterface
[TPS_SWCT_01150]	InternalBehavior of a NvBlockSwComponentType to enable access to the NVRAM Block management API
[TPS_SWCT_01162]	Existence of TextTableMapping
[TPS_SWCT_01180]	Maximum possible size of Compound Primitive Data Type
[TPS_SWCT_01227]	Unconnected AbstractRequiredPortPrototype typed by NvDataInterface
[TPS_SWCT_01228]	NvProvideComSpec
[TPS_SWCT_01239]	default value for attribute category used in the context of SwSystemconst
[TPS_SWCT_01274]	SwDataDefProps used to support calibration and measurement
[TPS_SWCT_01321]	Communication among RunnableEntitys
[TPS_SWCT_01330]	RunnableEntity can also have dataSendPoints
[TPS_SWCT_01335]	Combine dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint
[TPS_SWCT_01348]	Trigger source
[TPS_SWCT_01385]	Execution of finalization code for software-components
[TPS_SWCT_01456]	Predefined values for MemorySection.option and SwAddrMethod.option
[TPS_SWCT_01475]	Sending SwComponentType owns a DataPrototypeGroup in the role dpgRequiresCoherency

▽

△

Number	Heading
[TPS_SWCT_01494]	A RuleBasedValueSpecification of rule FILL_UNTIL_END shall fill the value of the last RuleArguments.argument until the last element of the array
[TPS_SWCT_01495]	Standardized value of RuleBasedValueSpecification.rule
[TPS_SWCT_01549]	Definition of linear data scaling
[TPS_SWCT_01560]	Supported categorys of CompuMethods for data conversion
[TPS_SWCT_02501]	Setup for Nvm Use Case: Permanent RAM Block
[TPS_SWCT_02502]	Setup for Nvm Use Case: Temporary RAM Block
[TPS_SWCT_02503]	Setup for NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType
[TPS_SWCT_02504]	Setup for Nvm Use Case: RAM Block with explicit synchronization using Mirror Interfaces

Table C.28: Changed Traceables in R4.2.1

C.7.3 Deleted Traceables in R4.2.1

Number	Heading
Id	**Heading**
[TPS_SWCT_01131]	AtomicSwComponentType accepts a request to restart an entire function
[TPS_SWCT_01386]	Initialization by mode management
[TPS_SWCT_01387]	Finalization by mode management
[TPS_SWCT_01410]	Dcm and Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface
[TPS_SWCT_01438]	Handling of invalidation in the sending RTE
[TPS_SWCT_01439]	Handling of invalidation in the receiving RTE
[TPS_SWCT_02017]	AtomicSwComponentType requires the notification about a Service Request via diagnostic services

Table C.29: Deleted Traceables in R4.2.1

C.7.4 Added Constraints in R4.2.1

Number	Heading
[constr_1300]	Primitive DataPrototype on the provider side shall not be mapped to element of a composite data type on the requester side
[constr_1301]	Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role

▽

△

Number	Heading
[constr_1302]	Restriction of data invalidation
[constr_1303]	Applicability of TextTableMapping depending on the value of CompuMethod.category
[constr_1304]	Existence of attribute bitfieldTextTableMaskFirst
[constr_1305]	Existence of attribute bitfieldTextTableMaskSecond
[constr_1306]	Limitation of TextTableMapping for CompuMethods that have the value of category set to BITFIELD_TEXTTABLE
[constr_1307]	Consistency of values and masks in TextTableMapping
[constr_1308]	Existence of NvBlockNeeds.cyclicWritingPeriod
[constr_1309]	Existence of NvBlockDescriptor.timingEvent
[constr_1310]	Existence of attributes of meta-class NvBlockNeeds
[constr_1311]	Appearance of safety-related possible values of MemorySection.option or SwAddrMethod.option according to [TPS_SWCT_01456]
[constr_1312]	PortPrototypes typed by a ParameterInterface
[constr_1313]	Completeness of TextTableMapping for the values of a given bit mask on the sender side
[constr_1314]	Profile VSA_LINEAR for ApplicationArrayDataType
[constr_1315]	Profile VSA_SQUARE for ApplicationArrayDataType
[constr_1316]	Profile VSA_RECTANGULAR for ApplicationArrayDataType
[constr_1317]	Profile VSA_FULLY_FLEXIBLE for ApplicationArrayDataType
[constr_1318]	Profile VSA_LINEAR for ImplementationDataType
[constr_1319]	Profile VSA_SQUARE for ImplementationDataType
[constr_1320]	Profile VSA_RECTANGULAR for ImplementationDataType
[constr_1321]	Profile VSA_FULLY_FLEXIBLE for ImplementationDataType
[constr_1322]	Size Indicator for undefined dynamicArraySizeProfile
[constr_1323]	Applicability of attribute ReceiverComSpec.usesEndToEndProtection
[constr_1363]	Existence of attributes of DiagnosticValueNeeds
[constr_1364]	Existence of attributes of DiagnosticIoControlNeeds
[constr_1375]	Existence of attributes of CompuMethod and related meta-classes

Table C.30: Added Constraints in R4.2.1

C.7.5 Changed Constraints in R4.2.1

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes

▽

△

Number	Heading
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDataDefProps
[constr_1037]	Client shall not be connected to multiple servers
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1101]	Mode-related communication
[constr_1112]	Constraints of dataIdMode in PROFILE_01
[constr_1149]	PortPrototypes used for NV data management
[constr_1188]	Existence of ReceiverComSpec.replaceWith
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_2009]	Supported kinds of PortPrototypes of a NvBlockSwComponentType
[constr_2015]	Limitation of SwcInternalBehavior of a NvBlockSwComponentType
[constr_2019]	ServiceSwComponentType shall have service ports only
[constr_2022]	Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints
[constr_2051]	Mandatory information of a SwValueCont

Table C.31: Changed Constraints in R4.2.1

C.7.6 Deleted Constraints in R4.2.1

Number	Heading
[constr_1189]	Allowed targets of externalReplacement
[constr_1293]	Existence of DiagnosticEventNeeds.dtcNumber
[constr_1294]	Existence of DiagnosticEventInfoNeeds.dtcNumber
[constr_2551]	SwCalprmAxis.baseType shall be ignored

Table C.32: Deleted Constraints in R4.2.1

C.8 Constraint History of this Document according to AUTOSAR R4.2.2

C.8.1 Added Traceables in R4.2.2

Number	Heading
[TPS_SWCT_01637]	Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable
[TPS_SWCT_01638]	Existence of SwConnector between two PRPortPrototypes
[TPS_SWCT_01639]	AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write current value via diagnostic services where the applicable DID is passed as an argument to the access functions
[TPS_SWCT_01640]	Suffix used for the resulting name of the PortInterface for the Data Services
[TPS_SWCT_01641]	Definition of an “old-world” dynamic-size array data type by means of an ApplicationArrayDataType
[TPS_SWCT_01642]	Definition of an “old-world” dynamic-size array data type by means of an ImplementationDataType
[TPS_SWCT_01644]	Definition of a “new-world” variable-size array data type by means of an ApplicationArrayDataType
[TPS_SWCT_01645]	Definition of a “new-world” variable-size array data type by means of an ImplementationDataType
[TPS_SWCT_01646]	Sending invalidValue without invalidation applied by RTE/Com
[TPS_SWCT_01647]	Size Indicator for dynamicArraySizeProfile set to VSA_LINEAR, VSA_SQUARE, or VSA_FULLY_FLEXIBLE if only ImplementationDataType is present
[TPS_SWCT_01648]	Size Indicator for dynamicArraySizeProfile set to VSA_RECTANGULAR if only ImplementationDataType is present
[TPS_SWCT_01649]	Payload for dynamicArraySizeProfile if only ImplementationDataType is present
[TPS_SWCT_01650]	Structure of the VSA ImplementationDataType
[TPS_SWCT_01651]	UTF-16BE
[TPS_SWCT_01652]	UTF-16LE
[TPS_SWCT_01653]	UTF-16-encoded strings are not allowed to start with a BOM
[TPS_SWCT_01654]	AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces to adjust the IO signal via diagnostic services
[TPS_SWCT_01655]	Reference from DiagnosticIoControlNeeds to DiagnosticValueNeeds
[TPS_SWCT_01656]	Suffix used for the resulting name of the PortInterface for IOControlRequest and IOControlResponse
[TPS_SWCT_01657]	NamingRule for RPortPrototype referenced by a RoleBasedPortAssignment with attribute role set to “IOControlRequest”

▽

△

Number	Heading
[TPS_SWCT_01658]	NamingRule for PPortPrototype referenced by a RoleBasedPortAssignment with attribute role set to "IOControlResponse"
[TPS_SWCT_01659]	Mapping of VariableDataPrototype to a NvBlockDescriptor

Table C.33: Added Traceabless in 4.2.2

C.8.2 Changed Traceables in R4.2.2

Number	Heading
[TPS_SWCT_01133]	AtomicSwComponentType provides information about aging cycles
[TPS_SWCT_01158]	Three cases for PortInterfaceMapping
[TPS_SWCT_01179]	Compound Primitive Data Type
[TPS_SWCT_01244]	Data types for calibration parameters are also described as primitive types
[TPS_SWCT_01370]	VariationPointProxy
[TPS_SWCT_01431]	Finding the symbol for the representation of a CompuScale with a point-range in C code
[TPS_SWCT_01432]	Keep the invalidValue transparent to the sending and receiving software components
[TPS_SWCT_01434]	Sender and receiver have knowledge of invalid value
[TPS_SWCT_01610]	Modeling of a Variable-Size Array Data Type with Size Indicator enabled
[TPS_SWCT_01616]	Semantics of TransformerHardErrorEvent
[TPS_SWCT_01617]	Structure of an ImplementationDataType that represents a variable-sized array data type
[TPS_SWCT_01624]	Hard error occurs during the execution of a transformer chain
[TPS_SWCT_02001]	Values of SwAxisCont with the category, COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_02053]	Values of RuleBasedAxisCont with the category COM_AXIS, RES_AXIS are for display only

Table C.34: Changed Traceables in R4.2.2

C.8.3 Deleted Traceables in R4.2.2

Number	Heading
[TPS_SWCT_01308]	Combination of supportsMultipleInstantiation=false and canBeInvokedConcurrently=false
[TPS_SWCT_01433]	Invalid values outside the range limits

▽

△

Number	Heading
[TPS_SWCT_01435]	Invalid values outside the range limits
[TPS_SWCT_01603]	Variable size array with Size Indicator
[TPS_SWCT_01611]	Enable Size Indicator

Table C.35: Deleted Traceables in R4.2.2

C.8.4 Added Constraints in R4.2.2

Number	Heading
[constr_1381]	Appearance of core-related possible values of MemorySection.option or SwAddrMethod.option according to [TPS_SWCT_01456]
[constr_1382]	Mutually exclusive existence of attributes SwVariableRefProxy.autosarVariable vs. SwVariableRefProxy.mcDataInstanceVar
[constr_1383]	Existence of CompuMethod and DataConstr for ImplementationDataTypes of category TYPE_REFERENCE
[constr_1384]	Definition of invalidValue for DataPrototype typed by ApplicationPrimitiveDataType of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK
[constr_1385]	DataPrototype is typed by an ImplementationDataType
[constr_1386]	PortDefinedArgumentValue shall only be defined for AbstractProvidedPortPrototype
[constr_1388]	VariationPointProxy of category VALUE shall not mix "pre-build" and "post-build" use-cases
[constr_1389]	Restriction regarding the value of category of VariationPointProxy.implementationDataType
[constr_1390]	Restriction to the value of SenderReceiverInterface.invalidationPolicy.handleInvalid
[constr_1391]	Compatibility of Units in the context of assignment using an ApplicationValueSpecification
[constr_1392]	Compatibility of Units in the context of assignment using an ApplicationRuleBasedValueSpecification
[constr_1393]	Existence of RuleBasedValueCont.unit
[constr_1395]	NvBlockDataMapping shall be complete
[constr_1396]	Restriction for the value of attribute category for non-terminating ImplementationDataTypeElements taken to model a Variable-Size Array Data Type
[constr_1397]	Existence of attributes of TransformerHardErrorEvent
[constr_1398]	Existence of attributes of BaseTypeDirectDefinition
[constr_1399]	Standardized values of ModeDeclarationGroup.category
[constr_1400]	Reference to a specific DataTransformation
[constr_1401]	Restrictions on the relation between DataPrototypeMapping and DataTransformation

▽

△

Number	Heading
[constr_1402]	Applicability of core-related possible values of MemorySection.option or SwAddrMethod.option related to SwAddrMethod.sectionInitializationPolicy
[constr_1403]	NvBlockDataMappings to a given nvData shall be unambiguous
[constr_1404]	All NvDataInterface.nvData of PortPrototypes in the context of a specific SwcServiceDependency shall be mapped to the same NvBlockDescriptor

Table C.36: Added Constraints in R4.2.2

C.8.5 Changed Constraints in R4.2.2

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1040]	Conversion of SenderReceiverInterfaces
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK
[constr_1064]	Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, or CUBE_5
[constr_1066]	Forbidden mappings to ImplementationDataType
[constr_1075]	Compatibility of ModeDeclarationGroups
[constr_1164]	Number of arguments owned by a RunnableEntity
[constr_1234]	Value of RunnableEntity.symbol
[constr_1318]	Profile VSA_LINEAR for ImplementationDataType
[constr_1319]	Profile VSA_SQUARE for ImplementationDataType
[constr_1320]	Profile VSA_RECTANGULAR for ImplementationDataType
[constr_1321]	Profile VSA_FULLY_FLEXIBLE for ImplementationDataType
[constr_1322]	Size Indicator for undefined dynamicArraySizeProfile
[constr_2051]	Mandatory information of a SwValueCont
[constr_2058]	Mandatory information of a RuleBasedValueCont

Table C.37: Changed Constraints in R4.2.2

C.8.6 Deleted Constraints in R4.2.2

Number	Heading
[constr_1002]	End-to-end protection does not support n:1 communication
[constr_1067]	ApplicationDataType is or is not compatible to specific ImplementationDataType
[constr_2001]	Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable

Table C.38: Deleted Constraints in R4.2.2

C.9 Constraint History of this Document according to AUTOSAR R4.3.0

C.9.1 Added Traceables in R4.3.0

Number	Heading
[TPS_SWCT_01660]	Values of SwcServiceDependency.category reserved by the standard
[TPS_SWCT_01661]	Default value of SwcServiceDependency.category
[TPS_SWCT_01662]	Applicability of DataTypeMappingSets inside an NvBlockSwComponentType
[TPS_SWCT_01663]	dataReadAccess vs. dataReceivePointByValue or dataReceivePointByArgument
[TPS_SWCT_01664]	BswM acts as a mode requester towards an application mode manager
[TPS_SWCT_01665]	Usage of SwcModeSwitchEvent for triggering a write procedure of *nv data*
[TPS_SWCT_01666]	Semantics of ModeSwitchEventTriggeredActivity.role
[TPS_SWCT_01667]	Avoidance of overlapping of directly adjacent intervals within CompuMethods
[TPS_SWCT_01668]	SecOc Use Case: obtain the verification status of secure communication
[TPS_SWCT_01672]	SecOc Use Case: software component retires from secure communication for a given period
[TPS_SWCT_01673]	Application Software Component sends requests using the J1939Rm
[TPS_SWCT_01674]	Application Software Component accepts requests using the J1939Rm
[TPS_SWCT_01675]	Recommendations for attributes of NvBlockNeeds or for NvBlockDescriptor
[TPS_SWCT_01676]	Preferred approach to checking the compatibility of input value and axis
[TPS_SWCT_01677]	Fall-back approach to checking the compatibility of input value and axis
[TPS_SWCT_01678]	StbM use Case: Application software component accesses the Synchronized Time-Base Manager
[TPS_SWCT_01679]	StbM use Case: Synchronized Time-Base Manager notifies application software component

▽

△

Number	Heading
[TPS_SWCT_01680]	Dem Use Case: Atomic Software-Component implements a Hardware Shutdown
[TPS_SWCT_01681]	Context path in ArVariableInImplementationDataInstanceRef
[TPS_SWCT_01682]	The meaning of E2E-related attributes in a ReceiverComSpec if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined.
[TPS_SWCT_01683]	Specification of an array of input variable for an array of axes
[TPS_SWCT_01684]	Specification of a single input variable for an array of axes
[TPS_SWCT_01685]	Specification of an array of group axes for an array of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5
[TPS_SWCT_01686]	Specification of a single group axis for an array of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5
[TPS_SWCT_01687]	Support of locked communication buffers
[TPS_SWCT_01688]	initValue should exist in an RPortPrototype
[TPS_SWCT_01689]	Relation between SwcServiceDependencys and PortPrototypes
[TPS_SWCT_01690]	AtomicSwComponentType offers a PPortPrototype typed by ClientServerInterface to read/write and IOCtrl current value via diagnostic services
[TPS_SWCT_01691]	Suffix used for the resulting name of the PortInterface for the Data Services
[TPS_SWCT_01692]	Meaning of CompositeRuleBasedValueSpecification
[TPS_SWCT_01693]	Usage of VendorSpecificServiceNeeds
[TPS_SWCT_01694]	Setup for Default Error Tracer Service use Case: development errors or runtime error
[TPS_SWCT_01695]	Relation between ValueSpecification and the definition of CompuScales
[TPS_SWCT_01696]	CompuScale Value Symbolic Name
[TPS_SWCT_01697]	Supported development approach for software-components that interact with AUTOSAR services
[TPS_SWCT_01698]	Attributes that are subject to development approach
[TPS_SWCT_01699]	Usage of ApplicationArrayElement.indexDataType
[TPS_SWCT_01700]	Definition of unions that can be transmitted over a communication network
[TPS_SWCT_01701]	Wrapped Union Data Type
[TPS_SWCT_01702]	Initialization of the "payload" of a Wrapped Union Data Type
[TPS_SWCT_01703]	Setup for AtomicSwComponentType which sets or gets the Global Supervision Status
[TPS_SWCT_01704]	Definition of supervised entity
[TPS_SWCT_01705]	Definition of Checkpoints
[TPS_SWCT_01706]	AtomicSwComponentType supports *DiagnosticSessionControl* to get informed about the diagnostic session
[TPS_SWCT_01707]	AtomicSwComponentType supports *EcuReset* service via diagnostic services
[TPS_SWCT_01708]	AtomicSwComponentType supports *EcuReset ModeRapidPowerShutDown* service via diagnostic services

▽

△

Number	Heading
[TPS_SWCT_01709]	AtomicSwComponentType supports *CommunicationControl* service via diagnostic services
[TPS_SWCT_01710]	Suffix used for the resulting name of the PortInterface for the Communication Control
[TPS_SWCT_01711]	AtomicSwComponentType supports *ControlDTCSetting* service via diagnostic services
[TPS_SWCT_01712]	AtomicSwComponentType supports SecurityAccess to get informed about the security level
[TPS_SWCT_01713]	ExclusiveArea is entered and exited by a common set of APIs
[TPS_SWCT_01714]	ExclusiveArea is entered and exited by an individual set of APIs
[TPS_SWCT_01715]	Software-Component wants to be triggered on Monitor Status Changes
[TPS_SWCT_01716]	SecOc Use Case: deliver freshness to SecOC I
[TPS_SWCT_01717]	SecOc Use Case: deliver freshness to SecOC II
[TPS_SWCT_01718]	SecOc Use Case: deliver freshness to SecOC III
[TPS_SWCT_01719]	Selection of applicable RptProfiles
[TPS_SWCT_01720]	Preparation Levels
[TPS_SWCT_01721]	References from RptContainer
[TPS_SWCT_01722]	Semantics of RP Service Point
[TPS_SWCT_01723]	Semantics of RP Service Function
[TPS_SWCT_01724]	Semantics of RapidPrototypingScenario
[TPS_SWCT_01725]	AtomicSwComponentType uses the secure counter of the Crypto Service
[TPS_SWCT_01726]	AtomicSwComponentType uses the key management of the Crypto Service
[TPS_SWCT_01727]	Suffix used for the resulting name of the PortInterface for crypto PortInterfaces
[TPS_SWCT_01728]	V2xFac Use Case: Application software component provides Vehicle specific data to the V2X-Stack for CAM transmission
[TPS_SWCT_01729]	V2xFac Use Case: V2xFac notifies application software component about received messages
[TPS_SWCT_01730]	V2xFac Use Case: Application software component triggers transmission of DENM message
[TPS_SWCT_01731]	V2xM Use Case: Application software component provides Vehicle specific data to the V2X-Stack for Position and Time information
[TPS_SWCT_01732]	V2xM Use Case: Application software component needs V2X specific data from the V2X Manager
[TPS_SWCT_01733]	V2xM Use Case: Application software component has soft-control over Pseudonym-Change within V2X Manager
[TPS_SWCT_01734]	V2xM Use Case: Application software component has the ability to do Verification-on-Demand
[TPS_SWCT_01735]	V2xM Use Case: Application software component do location based calculations

Table C.39: Added Traceabless in 4.3.0

C.9.2 Changed Traceables in R4.3.0

Number	Heading
[TPS_SWCT_01005]	Usage of SwcServiceDependencys for vendor-specific services
[TPS_SWCT_01054]	Semantics of the explicitInterRunnableVariable
[TPS_SWCT_01134]	AtomicSwComponentType enables reporting of DTCs in general
[TPS_SWCT_01158]	Cases for PortInterfaceMapping
[TPS_SWCT_01173]	Control port shall **not** become a part of the PortGroup
[TPS_SWCT_01174]	Status port shall **not** become a member of the PortGroup
[TPS_SWCT_01182]	Conceptual levels for the definition of initial values
[TPS_SWCT_01275]	values of the attribute swImplPolicy are restricted depending on the context
[TPS_SWCT_01331]	dataWriteAccess vs. dataSendPoint
[TPS_SWCT_01355]	enableTakeAddress = true
[TPS_SWCT_01463]	ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet defines the applicable set of ModeDeclarationMappings
[TPS_SWCT_01493]	The number of RuleBasedValueSpecification.arguments shall not exceed the array size
[TPS_SWCT_01494]	A RuleBasedValueSpecification of rule FILL_UNTIL_END shall fill the value of the last RuleBasedValueSpecification.arguments until the last element of the array
[TPS_SWCT_01552]	Software-component acts as a mode manager
[TPS_SWCT_01553]	Software-component acts as a mode user
[TPS_SWCT_01554]	Software-component acts as a mode requester
[TPS_SWCT_01569]	Definition of CompuScale Code Symbolic Name
[TPS_SWCT_01586]	Writing strategies for *nv data*
[TPS_SWCT_01609]	A RuleBasedValueSpecification of rule FILL_UNTIL_MAX_SIZE shall fill the value of the last RuleBasedValueSpecification.arguments until the number of elements specified in maxSizeToFill
[TPS_SWCT_01635]	Naming conventions may support the effectiveness of SymbolProps
[TPS_SWCT_02001]	Values of SwAxisCont with the category COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_02014]	AtomicSwComponentType supports Response On Event (ROE) via diagnostic services
[TPS_SWCT_02020]	AtomicSwComponentType uses the hash calculation of the Crypto Service
[TPS_SWCT_02021]	AtomicSwComponentType uses the message authentication code (MAC) calculation of the Crypto Service
[TPS_SWCT_02022]	AtomicSwComponentType uses the message authentication code (MAC) verification of the Crypto Service
[TPS_SWCT_02024]	AtomicSwComponentType uses the generation of random numbers of the Crypto Service
[TPS_SWCT_02025]	AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service

▽

△

Number	Heading
[TPS_SWCT_02026]	AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service
[TPS_SWCT_02027]	AtomicSwComponentType uses the encryption of the Crypto Service
[TPS_SWCT_02028]	AtomicSwComponentType uses the decryption of the Crypto Service
[TPS_SWCT_02031]	AtomicSwComponentType uses the signature generation of the Crypto Service
[TPS_SWCT_02032]	AtomicSwComponentType uses the signature verification of the Crypto Service
[TPS_SWCT_02053]	Values of RuleBasedAxisCont with the category COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_02503]	Setup for NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType
[TPS_SWCT_02506]	Setup for Dlt use Case: Application software component accesses the Dlt module

Table C.40: Changed Traceables in R4.3.0

C.9.3 Deleted Traceables in R4.3.0

Number	Heading
[TPS_SWCT_01279]	Preferred conversion direction depends on the use case
[TPS_SWCT_01428]	ServiceSwComponentType representing the Dem provides a PPortPrototype for the Dcm
[TPS_SWCT_02018]	Setup for AtomicSwComponentType which contains a Supervised Entity
[TPS_SWCT_02023]	AtomicSwComponentType uses the generation of random seed of the Crypto Service
[TPS_SWCT_02029]	AtomicSwComponentType uses the asymmetrical encryption of the Crypto Service
[TPS_SWCT_02030]	AtomicSwComponentType uses the asymmetrical decryption of the Crypto Service
[TPS_SWCT_02033]	AtomicSwComponentType uses the checksum calculation of the Crypto Service
[TPS_SWCT_02034]	AtomicSwComponentType uses the key derivation of the Crypto Service
[TPS_SWCT_02035]	AtomicSwComponentType uses the symmetric key derivation of the Crypto Service
[TPS_SWCT_02036]	AtomicSwComponentType uses the key exchange interface for public value calculation of the Crypto Service
[TPS_SWCT_02037]	AtomicSwComponentType uses the key exchange interface for secret value calculation of the Crypto Service
[TPS_SWCT_02038]	AtomicSwComponentType uses the key exchange interface to calculate symmetric key with the Crypto Service

▽

△

Number	Heading
[TPS_SWCT_02039]	AtomicSwComponentType uses the symmetrical key extraction of the Crypto Service
[TPS_SWCT_02040]	AtomicSwComponentType uses the symmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a symmetric key
[TPS_SWCT_02041]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a asymmetric key
[TPS_SWCT_02042]	AtomicSwComponentType uses the asymmetrical public key extraction of the Crypto Service
[TPS_SWCT_02043]	AtomicSwComponentType uses the asymmetrical private key extraction of the Crypto Service
[TPS_SWCT_02044]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a symmetrical wrapping key
[TPS_SWCT_02045]	AtomicSwComponentType uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a asymmetrical wrapping key

Table C.41: Deleted Traceables in R4.3.0

C.9.4 Added Constraints in R4.3.0

Number	Heading
[constr_1407]	Definition of SwDataDefProps.dataConstr depending on the capabilities of the data type
[constr_1408]	Definition of SwDataDefProps.displayFormat depending on the capabilities of the data type
[constr_1409]	Definition of SwDataDefProps.dataConstr depending on the capabilities of the element data type
[constr_1410]	Definition of SwDataDefProps.displayFormat depending on the capabilities of the element data type
[constr_1413]	Definition of SwDataDefProps.stepSize depending on the capabilities of the data type
[constr_1414]	Definition of SwDataDefProps.stepSize depending on the capabilities of the element data type
[constr_1415]	Supported values of ModeSwitchEventTriggeredActivity.role
[constr_1416]	Existence of ApplicationArrayElement.maxNumberOfElements
[constr_1417]	Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (I)
[constr_1418]	Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (II)
[constr_1420]	Existence of SwAxisIndividual.inputVariableType

▽

△

Number	Heading
[constr_1422]	Value of category is VOID
[constr_1423]	Completeness of references ArVariableInImplementationDataInstanceRef.contextDataPrototype
[constr_1424]	Existence of ArVariableInImplementationDataInstanceRef.contextDataPrototype
[constr_1425]	Definition of swCalprmAxisSet.swCalprmAxis/ SwAxisIndividual.swVariableRef depending on the capabilities of the data type
[constr_1426]	Consistency of array sizes for axes and input variable array
[constr_1427]	Definition of swCalprmAxisSet.swCalprmAxis/ SwAxisGrouped.swCalprmRef depending on the capabilities of the data type
[constr_1428]	Consistency of array sizes for arrays of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 arrays and used group axes arrays
[constr_1429]	Access to data within PortPrototypes from within RunnableEntitys
[constr_1430]	Access to local data from within RunnableEntitys
[constr_1431]	Access to parameters from within RunnableEntitys
[constr_1432]	Multiplicity of CommunicationBufferLocking
[constr_1433]	Transient faults are not applicable to software-components
[constr_1434]	CompuScales shall not have identical CompuScale Value Symbolic Names
[constr_1436]	DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to requestCallbackTypeSupplier
[constr_1437]	DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to requestCallbackTypeManufacturer
[constr_1438]	ApplicationArrayElement.indexDataType needs to refer to a CompuMethod of category TEXTTABLE
[constr_1439]	Requirements on ApplicationArrayElement if attribute indexDataType exists
[constr_1440]	Size of the CompuMethod of category TEXTTABLE referenced by ApplicationArrayElement.indexDataType
[constr_1442]	category TYPE_REFERENCE shall not be used for modeling the “payload” of a Wrapped Union Data Type
[constr_1443]	category UNION shall not be used for ImplementationDataType
[constr_1444]	Limited applicability of Wrapped Union Data Type
[constr_1445]	Initialization of the Member Selector of a Wrapped Union Data Type
[constr_1446]	No definition of invalidValue for a Wrapped Union Data Type
[constr_1468]	Limitation on the number of SwcExclusiveAreaPolicys
[constr_1469]	Applicability of constraints depending on the existence of a data transformation

Table C.42: Added Constraints in R4.3.0

C.9.5 Changed Constraints in R4.3.0

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1011]	category of SwBaseType
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDataDefProps
[constr_1024]	Stepwise definition of CompuMethods
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1105]	Value of arraySize
[constr_1240]	Consistency of ArgumentDataPrototypes within the context of a ClientServer-OperationMapping
[constr_1271]	RecordValueSpecification.fields shall be identical to the number of ApplicationRecordDataType.elements
[constr_1272]	RecordValueSpecification.fields shall be identical to the number of subElements of ImplementationDataType of category STRUCTURE
[constr_1284]	Limitation of the use of TextValueSpecification
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by Implementation-DataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by Application-DataTypes
[constr_1302]	Restriction of data invalidation
[constr_1375]	Existence of attributes of CompuMethod and related meta-classes
[constr_1400]	Reference to a specific DataTransformation
[constr_2006]	Number of AsynchronousServerCallResultPoint referencing to one AsynchronousServerCallPoint
[constr_2015]	Limitation of SwcInternalBehavior of a NvBlockSwComponentType
[constr_2041]	swImplPolicy for VariableDataPrototype in the role staticMemory
[constr_2051]	Mandatory information of a SwValueCont
[constr_2058]	Mandatory information of a RuleBasedValueCont

Table C.43: Changed Constraints in R4.3.0

C.9.6 Deleted Constraints in R4.3.0

Number	Heading
[constr_1019]	Compatibility of input value and axis
[constr_1021]	A CompuMethod shall specify instructions for both directions
[constr_1133]	Identical CompuScale Symbolic Names shall have the same range
[constr_1201]	initValue shall exist in an RPortPrototype
[constr_1323]	Applicability of attribute ReceiverComSpec.usesEndToEndProtection

Table C.44: Deleted Constraints in R4.3.0

C.10 Constraint History of this Document according to AUTOSAR R4.3.1

C.10.1 Added Traceables in R4.3.1

Number	Heading
[TPS_SWCT_01736]	Default values for Unit.physicalDimension
[TPS_SWCT_01737]	Default values for physical exponents
[TPS_SWCT_01738]	Context path in ArParameterInImplementationDataInstanceRef
[TPS_SWCT_01739]	Function Inhibition Manager Use Case: react on suppressed or unavailable events
[TPS_SWCT_01740]	StbM use Case: Process time snapshot obtained from global time slave for diagnostics purposes
[TPS_SWCT_01741]	Suffix used for the resulting name of the PortInterface for measurement notification
[TPS_SWCT_01742]	StbM use Case: Software-component represents a global time master
[TPS_SWCT_01743]	Suffix used for the resulting name of the PortInterface for the global time master role
[TPS_SWCT_01744]	StbM use Case: Software-component represents a global time slave
[TPS_SWCT_01745]	Suffix used for the resulting name of the PortInterface for the global time slave role
[TPS_SWCT_01746]	Atomic Software-Component provides the further action byte to the DoIP Service Component
[TPS_SWCT_01747]	Value of category for fix axis
[TPS_SWCT_01748]	Sub-categories of fix axes
[TPS_SWCT_01749]	Semantics of SwAxisGeneric.swAxisType in the definition of a fix axis
[TPS_SWCT_01750]	Semantics of SwAxisGeneric.swGenericAxisParam in the definition of a fix axis

Table C.45: Added Traceables in R4.3.1

C.10.2 Changed Traceables in R4.3.1

Number	Heading
[TPS_SWCT_01193]	Mappings between application and implementation types do not necessarily have to form a 1:1 relation
[TPS_SWCT_01222]	Applicability of DataFilter
[TPS_SWCT_01225]	RunnableEntity implements the functionality of more than one ClientServerOperations
[TPS_SWCT_01288]	Interpretation of PhysConstrs and InternalConstrs by tools
[TPS_SWCT_01444]	Size of SwBaseType is specified in bits
[TPS_SWCT_01560]	Supported categorys of CompuMethods for data conversion
[TPS_SWCT_01675]	Recommendations for attributes of NvBlockNeeds or for NvBlockDescriptor
[TPS_SWCT_01678]	StbM use Case: start timer and potentially get notified about its expiration
[TPS_SWCT_01679]	StbM use Case: Software-Components wants to get notifications of status changes
[TPS_SWCT_01714]	ExclusiveArea is entered and exited by an individual set of APIs
[TPS_SWCT_02506]	Setup for Dlt use Case: Application software component accesses the Dlt module

Table C.46: Changed Traceables in R4.3.1

C.10.3 Deleted Traceables in R4.3.1

Number	Heading
[TPS_SWCT_01490]	AUTOSAR supports ApplicationErrors only for ClientServerInterfaces

Table C.47: Deleted Traceables in R4.3.1

C.10.4 Added Constraints in R4.3.1

Number	Heading
[constr_1515]	Existence of ImplementationDataTypeSubElementRef.implementationDataTypeElement as opposed to ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement
[constr_1516]	Completeness of references ArParameterInImplementationDataInstanceRef.contextDataPrototype
[constr_1517]	Existence of ArParameterInImplementationDataInstanceRef.contextDataPrototype

▽

△

Number	Heading
[constr_1518]	Consistency of data types in the context of ArParameterInImplementationDataInstanceRef
[constr_1519]	Existence of attributes vs. category of ApplicationValueSpecification
[constr_1520]	Semantics of ObdRatioServiceNeeds.rateBasedMonitoredEvent
[constr_1521]	Reference from AsynchronousServerCallReturnsEvent to AsynchronousServerCallResultPoint
[constr_1523]	No mode disabling for OperationInvokedEvents
[constr_1538]	Restriction for ReceiverComSpec.dataElement
[constr_1539]	Restriction for SenderComSpec.dataElement
[constr_1540]	Existence of ClientComSpec.operation
[constr_1541]	Existence of ServerComSpec.operation
[constr_1544]	Modeling of SwAxisGeneric for the definition of a fix axis
[constr_1545]	No initialization for fix axis

Table C.48: Added Constraints in R4.3.1

C.10.5 Changed Constraints in R4.3.1

Number	Heading
[constr_1004]	Mapping of ApplicationDataTypes in the scope of single AtomicSwComponentTypes
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1011]	category of SwBaseType
[constr_1012]	Value of category is FIXED_LENGTH
[constr_1015]	Prioritization of SwDataDefProps
[constr_1038]	Reference to ApplicationError
[constr_1044]	Applicability of DataFilter
[constr_1090]	WaitPoint and RunnableEntity
[constr_1102]	ApplicationError in the scope of one SwComponentType
[constr_1126]	Compatibility of DataConstrs
[constr_1220]	Compatibility of SwBaseType
[constr_1229]	category of ImplementationDataType boils down to VALUE
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_1422]	Value of category is VOID
[constr_2027]	SwcServiceDependency shall be defined for service ports only

▽

△

Number	Heading
[constr_2043]	swImplPolicy for ParameterDataPrototype in the role romBlock
[constr_2044]	swImplPolicy for ParameterDataPrototype in the role sharedParameter
[constr_2045]	swImplPolicy for ParameterDataPrototype in the role perInstanceParameter
[constr_2046]	swImplPolicy for ParameterDataPrototype in the role constantMemory

Table C.49: Changed Constraints in R4.3.1

C.10.6 Deleted Constraints in R4.3.1

Number	Heading
[constr_1013]	Value of category is VARIABLE_LENGTH
[constr_1436]	DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to requestCallbackTypeSupplier
[constr_1437]	DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to requestCallbackTypeManufacturer

Table C.50: Deleted Constraints in R4.3.1

C.11 Constraint History of this Document according to AUTOSAR R4.4.0

C.11.1 Added Traceables in R4.4.0

Number	Heading
[TPS_SWCT_01751]	The meaning of E2E-related attributes in a SenderComSpec if a TransformationComSpecProps of type EndToEndTransformationComSpecProps is defined
[TPS_SWCT_01752]	Initialization of a variable-size array
[TPS_SWCT_01753]	Application of compatibility rules for ReceiverComSpec.replaceWith
[TPS_SWCT_01754]	initValue defined in the context of a ComSpec
[TPS_SWCT_01755]	Duplicate existence of initValue in the context of a PRPortPrototype typed by an NvDataInterface
[TPS_SWCT_01756]	Semantics of SwDataDefProps.displayPresentation
[TPS_SWCT_01757]	Not-applicable scenario for presentationContinuous
[TPS_SWCT_01758]	Applicable value range of SwDataDefProps.displayPresentation
[TPS_SWCT_01759]	Use cases for unions

▽

△

Number	Heading
[TPS_SWCT_01760]	Defining the dimension of an ApplicationPrimitiveDataType of category VAL_BLK
[TPS_SWCT_01761]	Physical limits of pure textual conversions
[TPS_SWCT_01762]	Physical limits of mixed textual conversions
[TPS_SWCT_01763]	HtssM Service Use Case: Query results of hardware tests
[TPS_SWCT_01764]	V2xFac Use Case: Application software component shall be able to process the *MAP (topology) Extended Message*
[TPS_SWCT_01765]	Dem Service Use Case: In-Use-Monitoring Performance Ratio Denominator interface
[TPS_SWCT_01766]	Software-component computes the PIDs, and pushes them to Dem for storage and reporting to Dcm
[TPS_SWCT_01767]	Software-component located on an OBD master ECU reads the PID 21, and and sends the value around via “regular” sender-receiver communication
[TPS_SWCT_01768]	Semantics of DataPrototypeMapping.secondToFirstDataTransformation
[TPS_SWCT_01769]	Dcm Use Case: Upload and download of data
[TPS_SWCT_01770]	V2xFac Use Case: Application software component processes Infrastructure to Vehicle Information Message
[TPS_SWCT_01771]	Definition of optional elements on the level of ApplicationDataType
[TPS_SWCT_01772]	Semantics of attribute ImplementationDataType.isStructWithOptionalElement
[TPS_SWCT_01773]	Definition of Optional Element Structure on the level of ImplementationDataType
[TPS_SWCT_01774]	Modeling of ImplementationDataType with optional elements
[TPS_SWCT_01775]	Structured data types with optional elements
[TPS_SWCT_01776]	AtomicSwComponentType uses the API of the Crypto Service to set a key valid
[TPS_SWCT_01777]	AtomicSwComponentType uses the API of the Crypto Service to create a random seed
[TPS_SWCT_01778]	AtomicSwComponentType uses the API of the Crypto Service to generate a key
[TPS_SWCT_01779]	AtomicSwComponentType uses the API of the Crypto Service to derive a key
[TPS_SWCT_01780]	AtomicSwComponentType uses the API of the Crypto Service to execute calculation of the public value for key exchange
[TPS_SWCT_01781]	AtomicSwComponentType uses the API of the Crypto Service to execute calculation of shared secret for key exchange
[TPS_SWCT_01782]	AtomicSwComponentType uses the API of the Crypto Service to execute certificate parsing
[TPS_SWCT_01783]	AtomicSwComponentType uses the API of the Crypto Service to execute certificate verification
[TPS_SWCT_01784]	SecOc Use Case: enable the sending of Pdus even if computation of the MAC is not possible

▽

△

Number	Heading
[TPS_SWCT_01785]	Initial value for ImplementationDataType of category STRUCTURE where attribute isStructWithOptionalElement set to the value True
[TPS_SWCT_01786]	Initial value for the ImplementationDataTypeElement where the shortName is set to the value availabilityBitfield
[TPS_SWCT_01787]	Initialization of not-available ImplementationDataTypeElement
[TPS_SWCT_01788]	V2xFac Use Case: Application software component processes Signal Phase And Timing Extended Message
[TPS_SWCT_01789]	AtomicSwComponentType implements a Diagnostic Monitor that provides monitor data, debouncing by Dem
[TPS_SWCT_01790]	AtomicSwComponentType implements a Diagnostic Monitor that provides monitor data, debouncing by software-component
[TPS_SWCT_01791]	AtomicSwComponentType acts as a "file server" to a diagnostic tester

Table C.51: Added Traceables in R4.4.0

C.11.2 Changed Traceables in R4.4.0

Number	Heading
[TPS_SWCT_01016]	AtomicSwComponentType wants to select a shutdown target
[TPS_SWCT_01017]	AtomicSwComponentType wants to select a boot target
[TPS_SWCT_01018]	AtomicSwComponentType wants to use an alarm clock
[TPS_SWCT_01028]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_01029]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_01132]	AtomicSwComponentType provides information about operating cycles
[TPS_SWCT_01136]	AtomicSwComponentType retrieves information of the lamp status
[TPS_SWCT_01180]	Maximum possible size of Compound Primitive Data Type
[TPS_SWCT_01275]	values of the attribute swImplPolicy are restricted depending on the context
[TPS_SWCT_01374]	Implementation of AutosarParameterRef
[TPS_SWCT_01375]	Implementation of AutosarVariableRef
[TPS_SWCT_01398]	Communication patterns for AUTOSAR services
[TPS_SWCT_01405]	Creation of the ServiceSwComponentTypes
[TPS_SWCT_01426]	AtomicSwComponentType provides callback if any diagnostic event data and/or status changed
[TPS_SWCT_01456]	Predefined values for MemorySection.option and SwAddrMethod.option
[TPS_SWCT_01678]	StbM use Case: start timer and potentially get notified about its expiration
[TPS_SWCT_01698]	Attributes that are subject to development approach
[TPS_SWCT_01706]	AtomicSwComponentType supports *DiagnosticSessionControl* to get informed about the diagnostic session

▽

△

Number	Heading
[TPS_SWCT_01707]	AtomicSwComponentType supports *EcuReset* service via diagnostic services
[TPS_SWCT_01708]	AtomicSwComponentType supports *EcuReset ModeRapidPowerShutDown* service via diagnostic services
[TPS_SWCT_01709]	AtomicSwComponentType supports *CommunicationControl* service via diagnostic services
[TPS_SWCT_01711]	AtomicSwComponentType supports *ControlDTCSetting* service via diagnostic services
[TPS_SWCT_01712]	AtomicSwComponentType supports SecurityAccess to get informed about the security level
[TPS_SWCT_01715]	Software-Component wants to be triggered on Monitor Status Changes
[TPS_SWCT_01727]	Suffix used for the resulting name of the PortInterface for crypto PortInterfaces
[TPS_SWCT_01728]	V2xFac Use Case: Application software component provides Vehicle specific data to the V2X-Stack for CAM transmission
[TPS_SWCT_01729]	V2xFac Use Case: V2xFac notifies application software component about received messages
[TPS_SWCT_01730]	V2xFac Use Case: Application software component triggers transmission of DENM message
[TPS_SWCT_02000]	Default value for attribute swImplPolicy
[TPS_SWCT_02014]	AtomicSwComponentType supports Response On Event (ROE) via diagnostic services
[TPS_SWCT_02020]	AtomicSwComponentType uses the hash calculation of the Crypto Service
[TPS_SWCT_02021]	AtomicSwComponentType uses the message authentication code (MAC) calculation of the Crypto Service
[TPS_SWCT_02022]	AtomicSwComponentType uses the message authentication code (MAC) verification of the Crypto Service
[TPS_SWCT_02024]	AtomicSwComponentType uses the generation of random numbers of the Crypto Service
[TPS_SWCT_02025]	AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service
[TPS_SWCT_02026]	AtomicSwComponentType uses the Authenticated Encryption with Associated Data (AEAD) of the Crypto Service
[TPS_SWCT_02027]	AtomicSwComponentType uses the encryption of the Crypto Service
[TPS_SWCT_02028]	AtomicSwComponentType uses the decryption of the Crypto Service
[TPS_SWCT_02031]	AtomicSwComponentType uses the signature generation of the Crypto Service
[TPS_SWCT_02032]	AtomicSwComponentType uses the signature verification of the Crypto Service
[TPS_SWCT_02506]	Setup for Dlt use Case: Application software component accesses the Dlt module

Table C.52: Changed Traceables in R4.4.0

C.11.3 Deleted Traceables in R4.4.0

Number	Heading
[TPS_SWCT_01012]	AtomicSwComponentType reads the current ECU mode (fixed variant)
[TPS_SWCT_01013]	AtomicSwComponentType shall keep the ECU alive (fixed variant)
[TPS_SWCT_01014]	AtomicSwComponentType wants to select a shutdown target (fixed variant)
[TPS_SWCT_01015]	AtomicSwComponentType wants to select a boot target (fixed variant)
[TPS_SWCT_01125]	serverArgumentImplPolicy
[TPS_SWCT_01133]	AtomicSwComponentType provides information about aging cycles
[TPS_SWCT_01658]	NamingRule for PPortPrototype referenced by a RoleBasedPortAssignment with attribute role set to “IOControlResponse”
[TPS_SWCT_01710]	Suffix used for the resulting name of the PortInterface for the Communication Control
[TPS_SWCT_01725]	AtomicSwComponentType uses the secure counter of the Crypto Service

Table C.53: Deleted Traceables in R4.4.0

C.11.4 Added Constraints in R4.4.0

Number	Heading
[constr_1583]	PortInterfaceMapping for DataPrototype typed by Compound Primitive Data Type
[constr_1592]	Definition of SwDataDefProps.displayPresentation depending on the capabilities of the data type
[constr_1602]	Definition of SwDataDefProps.displayPresentation depending on the capabilities of the element
[constr_1607]	Only Wrapped Union Data Types in PortInterface
[constr_1608]	Existence of rootParameterDataPrototype
[constr_1609]	Existence of rootVariableDataPrototype
[constr_1610]	Existence of SwDataDefProps.swValueBlockSize and SwDataDefProps.swValueBlockSizeMult
[constr_1611]	Existence of ImplementationDataTypeSubElementRef.implementationDataTypeElement as opposed to ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement
[constr_1622]	Value of TimingEvent.offset vs. TimingEvent.period
[constr_1631]	Applicability of DataPrototypeMapping.secondToFirstDataTransformation
[constr_1632]	Restriction for firstToSecondDataTransformation and secondToFirstDataTransformation
[constr_1634]	Allowed combinations of ApplicationDataType.category vs. CompuMethod.category
[constr_1635]	Relevance of attribute isOptional

▽

△

Number	Heading
[constr_1636]	Mapping of data types that represent an Optional Element Structure
[constr_1637]	Existence of ImplementationDataTypeElement.isOptional vs. ImplementationDataType.isStructWithOptionalElement
[constr_1638]	First ImplementationDataTypeElement of ImplementationDataType that represents an Optional Element Structure
[constr_1639]	ImplementationDataTypeElement with attribute isOptional set to True
[constr_1640]	No use of Optional Element Structure for interaction with the diagnostic stack
[constr_1662]	Compatibility of ApplicationRecordDataType and ImplementationDataType that both represent an Optional Element Structure

Table C.54: Added Constraints in R4.4.0

C.11.5 Changed Constraints in R4.4.0

Number	Heading
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1015]	Prioritization of SwDataDefProps
[constr_1024]	Stepwise definition of CompuMethods
[constr_1048]	Compatibility of ApplicationRecordDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1273]	Rules for the initialization of ApplicationArrayDataType by means of ArrayValueSpecification
[constr_1274]	Rules for the initialization of array-shaped ImplementationDataType by means of ArrayValueSpecification
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_1400]	Reference to a specific DataTransformation
[constr_1444]	Limited applicability of Wrapped Union Data Type
[constr_1519]	Existence of attributes vs. category of ApplicationValueSpecification
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity

Table C.55: Changed Constraints in R4.4.0

C.11.6 Deleted Constraints in R4.4.0

Number	Heading
[constr_1032]	DelegationSwConnector can only connect PortPrototypes of the same kind
[constr_1297]	Applicability of serverArgumentImplPolicy set to useArrayBaseType
[constr_1443]	category UNION shall not be used for ImplementationDataType
[constr_1515]	Existence of ImplementationDataTypeSubElementRef.implementationDataTypeElement as opposed to ImplementationDataTypeSubElementRef.parameterImplementationDataTypeElement

Table C.56: Deleted Constraints in R4.4.0

C.12 Constraint History of this Document according to AUTOSAR R19-11

C.12.1 Added Traceables in R19-11

Number	Heading
[TPS_SWCT_01792]	Initialization of a DataPrototype associated with a CompuMethod of category BITFIELD_TEXTTABLE
[TPS_SWCT_01793]	Initialization of a *variable-size array* typed by an ImplementationDataType
[TPS_SWCT_01794]	Initialization of a *variable-size array* typed by an ApplicationArrayDataType
[TPS_SWCT_01795]	Further specification to facilitate the association of writing strategies to the corresponding RunnableEntity
[TPS_SWCT_01796]	Prioritization of SwDataDefProps.dataConstr for a DataPrototype of category ARRAY
[TPS_SWCT_01797]	Prioritization of SwDataDefProps.displayFormat for a DataPrototype of category ARRAY
[TPS_SWCT_01798]	Prioritization of SwDataDefProps.stepSize for a DataPrototype of category ARRAY
[TPS_SWCT_01799]	Mapping of bitfields between NvBlockDescriptor and PortPrototype
[TPS_SWCT_01801]	Support for Meta-Data
[TPS_SWCT_01802]	Definition of meta-data in the context of a SenderReceiverInterface
[TPS_SWCT_01803]	MetaDataItems define the same value of attribute length
[TPS_SWCT_01804]	Standardized values of attribute MetaDataItem.metaDataItemType.value
[TPS_SWCT_01805]	Semantics of the aggregation NvBlockSwComponentType.bulkNvDataDescriptor

▽

△

Number	Heading
[TPS_SWCT_01806]	Simultaneous aggregation of NvBlockSwComponentType.bulkNvDataDescriptor and NvBlockSwComponentType.nvBlockDescriptor
[TPS_SWCT_01807]	Application of NvBlockDataMapping on BulkNvDataDescriptor
[TPS_SWCT_03500]	Status forwarding to data transformer
[TPS_SWCT_03501]	Applicability of status forwarding to data transformer

Table C.57: Added Traceables in R19-11

C.12.2 Changed Traceables in R19-11

Number	Heading
[TPS_SWCT_01029]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_01030]	RunnableEntity
[TPS_SWCT_01132]	AtomicSwComponentType provides information about operating cycles
[TPS_SWCT_01148]	NvBlockDataMapping
[TPS_SWCT_01182]	Conceptual levels for the definition of initial values
[TPS_SWCT_01220]	initValue defines an initial value that shall be taken if the corresponding dataElement has not yet been received
[TPS_SWCT_01243]	Definition of enumeration types
[TPS_SWCT_01309]	signature of a RunnableEntity depends on the connected RTEEvent
[TPS_SWCT_01310]	Categories of RunnableEntitys
[TPS_SWCT_01654]	AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces to adjust the IO signal via diagnostic services
[TPS_SWCT_01656]	Suffix used for the resulting name of the PortInterface for DataServices, IOControlRequest, and IOControlResponse
[TPS_SWCT_01789]	AtomicSwComponentType implements a Diagnostic Monitor that provides monitor data, debouncing by Dem
[TPS_SWCT_01790]	AtomicSwComponentType implements a Diagnostic Monitor that provides monitor data, debouncing by software-component

Table C.58: Changed Traceables in R19-11

C.12.3 Deleted Traceables in R19-11

Number	Heading
[TPS_SWCT_01344]	Consistency of values of timeout
[TPS_SWCT_01429]	[constr_1135] only applies for BITFIELD_TEXTTABLE
[TPS_SWCT_01752]	Initialization of a variable-size array

Table C.59: Deleted Traceables in R19-11

C.12.4 Added Constraints in R19-11

Number	Heading
[constr_1679]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = signalBasedDiagnostics
[constr_1680]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = AppModeRequestInterface
[constr_1681]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = VerificationStatus
[constr_1682]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xFacVdp
[constr_1683]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationCam
[constr_1684]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationMapem
[constr_1685]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationIvim
[constr_1686]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationSpatem
[constr_1694]	Allowed target of SwDataDefProps.implementationDataType
[constr_1706]	Definition of initial value for data transmission
[constr_1712]	Existence of attribute ArrayValueSpecification.intendedPartialInitializationCount
[constr_1713]	NvBlockDescriptor.writingStrategyRole.usedDataElement shall refer to AutosarDataPrototype
[constr_1714]	AutosarDataPrototype shall only be referenced by a single NvBlockDescriptor.writingStrategyRole
[constr_1715]	Possible values of attribute NvBlockDescriptor.writingStrategyRole.role

▽

△

Number	Heading
[constr_1716]	Consistency of attribute NvBlockDescriptor.writingStrategyRole.role set to storeAtShutdown
[constr_1717]	Consistency of attribute NvBlockDescriptor.writingStrategyRole.role set to storeImmediate
[constr_1718]	Inheritance of SwDataDefProps.dataConstr from an array data type to the array elements
[constr_1719]	Inheritance of SwDataDefProps.displayFormat from an array data type to the array elements
[constr_1720]	Inheritance of SwDataDefProps.stepSize from an array data type to the array elements
[constr_1724]	Usage of attribute ClientServerOperation.diagArgIntegrity
[constr_1726]	Ordering of MetaDataItemSet.metaDataItem
[constr_1735]	Limitation of the aggregation of AutosarVariableRef in the context of an NvBlockDataMapping owned by a BulkNvDataDescriptor
[constr_1741]	Restriction to explicit sending semantics for the usage of DataServices in the context of a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds

Table C.60: Added Constraints in R19-11

C.12.5 Changed Constraints in R19-11

Number	Heading
[constr_1051]	Compatibility of SwDataDefProps
[constr_1059]	Compatibility of data types with category VALUE
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1064]	Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, or CUBE_5
[constr_1078]	Compatibility of ClientServerOperations
[constr_1137]	Applicability of ParameterInterface
[constr_1273]	Rules for the initialization of ApplicationArrayDataType by means of ArrayValueSpecification
[constr_1274]	Rules for the initialization of array-shaped ImplementationDataType by means of ArrayValueSpecification
[constr_1607]	Only Wrapped Union Data Types in PortInterface
[constr_1662]	Compatibility of ApplicationRecordDataType and ImplementationDataType that both represent an Optional Element Structure
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2013]	Compatibility of ImplementationDataTypes for NvBlockDataMapping

Table C.61: Changed Constraints in R19-11

C.12.6 Deleted Constraints in R19-11

none

C.13 Constraint History of this Document according to AUTOSAR R20-11

C.13.1 Added Traceables in R20-11

Number	Heading
[TPS_SWCT_01808]	Dem Service Use Case: software-component checks whether an event is suppressed
[TPS_SWCT_01809]	J1939Dcm wants to retrieve calibration verification numbers from an application software-component
[TPS_SWCT_01810]	Software-component analyzes predictions about the time synchronization process
[TPS_SWCT_01811]	AtomicSwComponentType reads the current PNC ComM mode
[TPS_SWCT_01812]	Conditional relevance of attribute EndToEndTransformationComSpecProps.disableEndToEndStateMachine
[TPS_SWCT_01813]	Software-Component wants check a certificate on KeyM
[TPS_SWCT_01814]	AtomicSwComponentType wants to retrieve a certificate from KeyM
[TPS_SWCT_01815]	AtomicSwComponentType wants to retrieve elements of a certificate from KeyM
[TPS_SWCT_01816]	AtomicSwComponentType wants to check the existence of a certificate from KeyM
[TPS_SWCT_01817]	AtomicSwComponentType wants to store a (derived) key in KeyM
[TPS_SWCT_01818]	AtomicSwComponentType wants to store a container with encrypted keys (e.g. She-keys) in KeyM
[TPS_SWCT_01819]	AtomicSwComponentType wants to verify if cryptographic operation was executed using a specific key
[TPS_SWCT_01820]	Existence of attribute SenderComSpec.handleOutOfRange
[TPS_SWCT_01821]	Semantics of attribute NotAvailableValueSpecification.defaultPattern
[TPS_SWCT_01822]	Application of attribute NotAvailableValueSpecification.defaultPattern happens only during initialization
[TPS_SWCT_01823]	Definition of ValueSpecification for an ApplicationRecordDataType with unavailable optional elements
[TPS_SWCT_01826]	Application Software Component reports security event
[TPS_SWCT_01827]	Suffix used for the resulting name of the PortInterface for the IdsM Services
[TPS_SWCT_01828]	Application Software Component reports security event using Smart Sensor API

▽

△

Number	Heading
[TPS_SWCT_01829]	Suffix used for the resulting name of the PortInterface for the IdsM Services
[TPS_SWCT_01830]	Application Software Component provides time stamp to IdsM

Table C.62: Added Traceables in R20-11

C.13.2 Changed Traceables in R20-11

Number	Heading
[TPS_SWCT_01004]	Specific default value if serviceKind is not defined
[TPS_SWCT_01078]	Configurable array size
[TPS_SWCT_01178]	Specialized subclasses of ValueSpecification
[TPS_SWCT_01221]	DataFilter
[TPS_SWCT_01495]	Standardized value of RuleBasedValueSpecification.rule
[TPS_SWCT_01642]	Definition of an “old-world” dynamic-size array data type by means of an ImplementationDataType
[TPS_SWCT_01673]	Application Software Component sends requests using the J1939Rm
[TPS_SWCT_01674]	Application Software Component accepts requests using the J1939Rm
[TPS_SWCT_02003]	AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces or NvDataInterfaces to read/write current values via diagnostic services
[TPS_SWCT_02052]	Definition of Rapid Prototyping Scenario is splittable

Table C.63: Changed Traceables in R20-11

C.13.3 Deleted Traceables in R20-11

Number	Heading
[TPS_SWCT_01039]	Purpose of variant handling
[TPS_SWCT_01510]	The role of pretended networking
[TPS_SWCT_01511]	Configuration option is encoded into ModeDeclaration
[TPS_SWCT_01512]	Request change of Pretended Networking mode
[TPS_SWCT_01513]	React on the change of Pretended Networking mode

Table C.64: Deleted Traceables in R20-11

C.13.4 Added Constraints in R20-11

Number	Heading
[constr_1754]	Aggregation of NumericalRuleBasedValueSpecification
[constr_1755]	Aggregation of CompositeRuleBasedValueSpecification
[constr_1771]	Existence of SwValueCont.unit
[constr_1773]	Value of attribute dataSendPoint.returnValueProvision
[constr_1774]	Value of attribute dataReceivePointByArgument.returnValueProvision
[constr_1775]	Value of attribute serverCallPoint.returnValueProvision
[constr_1776]	Value of attribute asynchronousServerCallResultPoint.returnValueProvision
[constr_1777]	Value of attribute externalTriggeringPoint.returnValueProvision
[constr_1778]	Value of attribute modeSwitchPoint.returnValueProvision
[constr_1779]	Scope of the definition of an AbstractRuleBasedValueSpecification
[constr_1783]	Existence of attribute ImplementationDataTypeElement.arrayImplPolicy
[constr_1860]	Multiplicity of DelegationSwConnector.innerPort
[constr_1861]	Multiplicity of DelegationSwConnector.outerPort
[constr_1862]	Multiplicity of PassThroughSwConnector.requiredOuterPort
[constr_1863]	Multiplicity of PassThroughSwConnector.providedOuterPort
[constr_1864]	Multiplicity of InstantiationRTEEventProps.refinedEvent
[constr_1865]	Existence of InvalidationPolicy.dataElement
[constr_1866]	Existence of MetaDataItem.length
[constr_1867]	Existence of MetaDataItem.metaDataItemType
[constr_1868]	Existence of MetaDataItemSet.dataElement
[constr_1869]	Existence of attribute ArgumentDataPrototype.direction
[constr_1870]	Existence of attribute ApplicationError.errorCode
[constr_1871]	Existence of attribute ModeRequestTypeMap.implementationDataType
[constr_1872]	Existence of attribute ModeRequestTypeMap.modeGroup
[constr_1873]	Existence of DataPrototypeMapping.firstDataPrototype
[constr_1874]	Existence of DataPrototypeMapping.secondDataPrototype
[constr_1875]	Existence of reference ClientServerOperationMapping.firstOperation
[constr_1876]	Existence of reference ClientServerOperationMapping.secondOperation
[constr_1877]	Existence of reference ModeDeclarationGroupPrototypeMapping.firstModeGroup
[constr_1878]	Existence of reference ModeDeclarationGroupPrototypeMapping.secondModeGroup
[constr_1879]	Existence of reference ModeDeclarationMapping.firstMode
[constr_1880]	Existence of reference ModeDeclarationMapping.secondMode
[constr_1881]	Existence of reference TriggerMapping.firstTrigger
[constr_1882]	Existence of reference TriggerMapping.secondTrigger

▽

△

Number	Heading
[constr_1883]	Existence of ApplicationCompositeDataTypeSubElementRef.applicationCompositeElement
[constr_1884]	Existence of attribute TextTableMapping.identicalMapping
[constr_1885]	Existence of attribute TextTableMapping.mappingDirection
[constr_1886]	Existence of attribute TextTableValuePair.firstValue
[constr_1887]	Existence of attribute TextTableValuePair.secondValue
[constr_1888]	Existence of attribute DataTransformation.executeDespiteDataUnavailability
[constr_1889]	Existence of attribute QueuedReceiverComSpec.queueLength
[constr_1890]	Existence of attribute DataFilter.dataFilterType
[constr_1891]	Existence of attribute NonqueuedReceiverComSpec.initValue
[constr_1892]	Existence of attribute TransmissionAcknowledgementRequest.timeout
[constr_1893]	Existence of attribute ServerComSpec.queueLength
[constr_1894]	Existence of attribute ModeSwitchSenderComSpec.queueLength
[constr_1895]	Existence of attribute ModeSwitchSenderComSpec.modeGroup
[constr_1896]	Existence of attribute ModeSwitchReceiverComSpec.modeGroup
[constr_1897]	Existence of reference ParameterProvideComSpec.parameter
[constr_1898]	Existence of reference ParameterRequireComSpec.parameter
[constr_1899]	Existence of reference NvRequireComSpec.variable
[constr_1900]	Existence of reference NvProvideComSpec.variable
[constr_1901]	Existence of attribute EndToEndDescription.category
[constr_1902]	Existence of attribute EndToEndProtection.endToEndProfile
[constr_1903]	Existence of reference DataTypeMap.applicationDataType
[constr_1904]	Existence of reference DataTypeMap.implementationDataType
[constr_1905]	Existence of attribute SwTextProps.arraySizeSemantics
[constr_1906]	Existence of attribute SwTextProps.swMaxTextSize
[constr_1907]	Existence of attribute ApplicationArrayDataType.element
[constr_1908]	Existence of attribute ApplicationRecordDataType.element
[constr_1909]	Existence of attribute ImplementationProps.symbol
[constr_1910]	Existence of attribute BaseType.baseTypeDefinition
[constr_1911]	Existence of ArVariableInImplementationDataInstanceRef.targetDataPrototype
[constr_1912]	Existence of reference ArParameterInImplementationDataInstanceRef.targetDataPrototype
[constr_1913]	Existence of attribute CompuRationalCoeffs.compuDenominator
[constr_1914]	Existence of attribute CompuRationalCoeffs.compuNumerator
[constr_1915]	Existence of attribute PhysicalDimensionMapping.firstPhysicalDimension
[constr_1916]	Existence of attribute PhysicalDimensionMapping.secondPhysicalDimension
[constr_1917]	Existence of ConstantSpecification.valueSpec

▽

△

Number	Heading
[constr_1918]	Existence of RecordValueSpecification.field
[constr_1919]	Existence of TextValueSpecification.value
[constr_1920]	Existence of NumericalValueSpecification.value
[constr_1921]	Existence of ReferenceValueSpecification.referenceValue
[constr_1922]	Existence of ApplicationRuleBasedValueSpecification.category
[constr_1923]	Existence of RuleBasedAxisCont.ruleBasedValues
[constr_1924]	Existence of RuleBasedValueCont.ruleBasedValues
[constr_1925]	Existence of NumericalRuleBasedValueSpecification.ruleBasedValues
[constr_1926]	Existence of RuleBasedValueSpecification.rule
[constr_1927]	Existence of RuleBasedValueSpecification.arguments
[constr_1928]	Existence of CompositeRuleBasedValueSpecification.rule
[constr_1929]	Existence of CompositeRuleBasedValueSpecification.argument
[constr_1930]	Existence of ConstantReference.constant
[constr_1931]	Existence of ConstantSpecificationMapping.applConstant
[constr_1932]	Existence of ConstantSpecificationMapping.implConstant
[constr_1933]	Existence of CalibrationParameterValue.initializedParameter
[constr_1934]	Existence of attribute SwcInternalBehavior.handleTerminationAndRestart
[constr_1935]	Existence of attribute SwcInternalBehavior.supportsMultipleInstantiation
[constr_1936]	Existence of attribute RunnableEntity.symbol
[constr_1937]	Existence of attribute TimingEvent.period
[constr_1938]	Existence of attribute RunnableEntityArgument.symbol
[constr_1939]	Existence of attribute ExecutableEntityActivationReason.bitPosition
[constr_1940]	Existence of attribute AsynchronousServerCallReturnsEvent.eventSource
[constr_1941]	Existence of attribute DataSendCompletedEvent.eventSource
[constr_1942]	Existence of attribute DataWriteCompletedEvent.eventSource
[constr_1943]	Existence of attribute DataReceivedEvent.data
[constr_1944]	Existence of attribute DataReceiveErrorEvent.data
[constr_1945]	Existence of attribute OperationInvokedEvent.operation
[constr_1946]	Existence of attribute SwcModeSwitchEvent.activation
[constr_1947]	Existence of reference SwcModeSwitchEvent.mode
[constr_1948]	Existence of attribute ModeSwitchedAckEvent.eventSource
[constr_1949]	Existence of attribute ExternalTriggerOccurredEvent.trigger
[constr_1950]	Existence of attribute InternalTriggerOccurredEvent.eventSource
[constr_1951]	Existence of attribute WaitPoint.timeout
[constr_1952]	Existence of reference WaitPoint.trigger
[constr_1953]	Existence of attribute SwcExclusiveAreaPolicy.apiPrinciple
[constr_1954]	Existence of attribute VariableAccess.accessedVariable
[constr_1955]	Existence of attribute ServerCallPoint.operation

▽

△

Number	Heading
[constr_1956]	Existence of attribute ServerCallPoint.timeout
[constr_1957]	Existence of attribute AsynchronousServerCallResultPoint.asynchronousServerCallPoint
[constr_1958]	Existence of attribute ParameterAccess.accessedParameter
[constr_1959]	Existence of attribute InstantiationDataDefProps.swDataDefProps
[constr_1960]	Existence of attribute PortAPIOption.port
[constr_1961]	Existence of attribute PortDefinedArgumentValue.value
[constr_1962]	Existence of attribute PortDefinedArgumentValue.valueType
[constr_1963]	Existence of attribute CommunicationBufferLocking.supportBufferLocking
[constr_1964]	Existence of attribute PerInstanceMemory.type
[constr_1965]	Existence of attribute PerInstanceMemory.typeDefinition
[constr_1966]	Existence of attribute Implementation.swVersion
[constr_1967]	Existence of attribute Implementation.vendorId
[constr_1968]	Existence of attribute Implementation.codeDescriptor
[constr_1969]	Existence of attribute SwcImplementation.behavior
[constr_1970]	Existence of attribute PerInstanceMemorySize.alignment
[constr_1971]	Existence of attribute PerInstanceMemorySize.perInstanceMemory
[constr_1972]	Existence of attribute PerInstanceMemorySize.size
[constr_1973]	Existence of attribute ModeDeclarationGroup.initialMode
[constr_1974]	Existence of attribute ModeDeclarationGroup.modeDeclaration
[constr_1975]	Existence of attribute ModeTransition.enteredMode
[constr_1976]	Existence of attribute ModeTransition.exitedMode
[constr_1977]	Existence of attribute ModeErrorBehavior.errorReactionPolicy
[constr_1978]	Existence of attribute SwcModeManagerErrorEvent.modeGroup
[constr_1979]	Existence of the reference SwcBswMapping.bswBehavior
[constr_1980]	Existence of the reference SwcBswMapping.swcBehavior
[constr_1981]	Existence of attribute NvBlockDescriptor.nvBlockNeeds
[constr_1982]	Existence of attribute ModeSwitchEventTriggeredActivity.role
[constr_1983]	Existence of attribute ModeSwitchEventTriggeredActivity.swcModeSwitchEvent
[constr_1984]	Existence of instance reference NvBlockDataMapping.nvRamBlockElement
[constr_1985]	Existence of the reference SupervisedEntityNeeds.toleratedFailedCycles
[constr_1986]	Existence of the reference DiagnosticRoutineNeeds.diagRoutineType
[constr_1987]	Existence of instance reference RapidPrototypingScenario.hostSystem
[constr_1988]	Existence of attribute RptProfile.maxServicePointId
[constr_1989]	Existence of attribute RptProfile.minServicePointId
[constr_1990]	Existence of attribute RptProfile.servicePointSymbolPost
[constr_1991]	Existence of attribute RptProfile.servicePointSymbolPre
[constr_1992]	Existence of attribute RptProfile.stimEnabler

▽

△

Number	Heading
[constr_1993]	Existence of attribute RptImplPolicy.rptEnablerImplType
[constr_1994]	Existence of attribute RptImplPolicy.rptPreparationLevel
[constr_1995]	Existence of attribute RptSwPrototypingAccess.rptHookAccess
[constr_1996]	Existence of attribute RptSwPrototypingAccess.rptReadAccess
[constr_1997]	Existence of attribute RptSwPrototypingAccess.rptWriteAccess
[constr_1998]	Existence of attribute RptExecutableEntityProperties.maxRptEventId
[constr_1999]	Existence of attribute RptExecutableEntityProperties.minRptEventId
[constr_5234]	Existence of attribute E2EProfileCompatibilityProps.transitToInvalidExtended is mandatory for each EndToEndTransformationComSpecProps
[constr_10000]	Existence of attribute RptExecutableEntityProperties.rptExecutionControl
[constr_10001]	Existence of attribute RptExecutableEntityProperties.rptServicePoint
[constr_10005]	Existence of attribute NotAvailableValueSpecification.defaultPattern
[constr_10006]	Valid interval of attribute NotAvailableValueSpecification.defaultPattern
[constr_10009]	Aggregation of ApplicationRuleBasedValueSpecification
[constr_10016]	Applicability of OsTaskExecutionEvent
[constr_10017]	Existence of attribute SwAxisCont.category
[constr_10018]	Existence of attribute SwAxisCont.swAxisIndex
[constr_10019]	Existence of attribute SwAxisCont.swValuesPhys
[constr_10020]	Existence of attribute RoleBasedDataTypeAssignment.usedImplementationDataType
[constr_10028]	Existence of reference stereotyped ≪isOfType≫

Table C.65: Added Constraints in R20-11

C.13.5 Changed Constraints in R20-11

Number	Heading
[constr_1051]	Compatibility of SwDataDefProps
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK
[constr_1066]	Forbidden mappings to ImplementationDataType
[constr_1197]	Existence of compositeNetworkRepresentation shall be comprehensive
[constr_1224]	DataPrototype is typed by an ApplicationArrayDataType
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_1296]	DataPrototypes used as explicitInterRunnableVariable or implicitInterRunnableVariable and category DATA_REFERENCE

▽

△

Number	Heading
[constr_1393]	Existence of RuleBasedValueCont.unit
[constr_1397]	Existence of attributes of TransformerHardErrorEvent
[constr_1422]	Value of category is VOID
[constr_1741]	Restriction to explicit sending semantics for the usage of DataServices in the context of a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2034]	SwAddrMethod referenced by RunnableEntitys, BswCalledEntitys, or BswSchedulableEntitys
[constr_2052]	Values of swArraysize and the number of values provided by swValuesPhys shall be consistent.
[constr_2058]	Mandatory information of a RuleBasedValueCont

Table C.66: Changed Constraints in R20-11

C.13.6 Deleted Constraints in R20-11

none

C.14 Constraint History of this Document according to AUTOSAR R21-11

C.14.1 Added Traceables in R21-11

Number	Heading
[TPS_SWCT_01831]	Dcm can directly access SenderReceiverInterface.dataElements or NvDataInterface.nvDatas in AbstractRequiredPortPrototypes
[TPS_SWCT_01832]	SecOc Use Case: Receive notification about an authentication attempt
[TPS_SWCT_01833]	Semantics of ServiceDependency.diagnosticRelevance
[TPS_SWCT_01834]	invalidValue is inside the scope of the compuMethod
[TPS_SWCT_01835]	invalidValue is outside the scope of the compuMethod
[TPS_SWCT_01836]	Attributes of CompositeRuleBasedValueSpecification
[TPS_SWCT_01837]	Types for record layouts
[TPS_SWCT_01838]	ValueSpecification shall fit into data type
[TPS_SWCT_01839]	Size of Compound Primitive Data Type is variant
[TPS_SWCT_01840]	A ParameterSwComponentType references a ConstantSpecificationMappingSet

▽

△

Number	Heading
[TPS_SWCT_01841]	A NvBlockSwComponentType references a ConstantSpecificationMappingSet
[TPS_SWCT_01842]	Applicability of constraints of CompuScales
[TPS_SWCT_01843]	Value of PassThroughSwConnector.category
[TPS_SWCT_01844]	Optional method arguments

Table C.67: Added Traceables in R21-11

C.14.2 Changed Traceables in R21-11

Number	Heading
[TPS_SWCT_01049]	Two ways to use the ExclusiveAreas
[TPS_SWCT_01195]	Mapping of composite element to primitive DataPrototype
[TPS_SWCT_01253]	Rules apply for the usage of the attribute ImplementationDataType.typeEmitter
[TPS_SWCT_01314]	RTEEvent
[TPS_SWCT_01551]	Mapping of elements on the “source” end to elements on the “target” end
[TPS_SWCT_01579]	Dcm can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in AbstractProvidedPortPrototype
[TPS_SWCT_01580]	Dem can directly access SenderReceiverInterface.dataElements, NvDataInterface.nvDatas, or ParameterInterface.parameters in PPortPrototypes
[TPS_SWCT_01586]	Writing strategies for *nv data*
[TPS_SWCT_01795]	Further specification to facilitate the association of writing strategies to the corresponding RunnableEntity
[TPS_SWCT_02011]	AtomicSwComponentType offers a client port to read DTR value

Table C.68: Changed Traceables in R21-11

C.14.3 Deleted Traceables in R21-11

Number	Heading
[TPS_SWCT_01591]	Existence of attribute DiagnosticEventNeeds.reportBehavior
[TPS_SWCT_01697]	Supported development approach for software-components that interact with AUTOSAR services
[TPS_SWCT_01698]	Attributes that are subject to development approach

Table C.69: Deleted Traceables in R21-11

C.14.4 Added Constraints in R21-11

Number	Heading
[constr_10032]	Restrictions for the usage of ServiceDependency.diagnosticRelevance
[constr_10033]	Existence of MemorySection.swAddrmethod
[constr_10034]	Existence of MemorySection.alignment
[constr_10040]	Value of ApplicationValueSpecification.swAxisCont.category
[constr_10041]	Value of ApplicationRuleBasedValueSpecification.swAxisCont.category
[constr_10067]	Creation of AssemblySwConnector for service communication
[constr_10068]	Standardized values for SectionInitializationPolicyType
[constr_10071]	Allowed multiplicities of SenderComSpec attributes for communication between ApplicationSwComponentType and NvBlockSwComponentType
[constr_10072]	Allowed multiplicities of SenderComSpec attributes for communication between NvBlockSwComponentType and ApplicationSwComponentType
[constr_10073]	Existence of DataReceiveErrorEvent
[constr_10074]	Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeOnChange
[constr_10075]	Existence of CompositeRuleBasedValueSpecification.argument vs. compoundPrimitiveArgument
[constr_10087]	Restriction for the existence of a SubElementMapping

Table C.70: Added Constraints in R21-11

C.14.5 Changed Constraints in R21-11

Number	Heading
[constr_1004]	Mapping of ApplicationDataTypes in the scope of single AtomicSwComponentTypes
[constr_1005]	Compatibility of ImplementationDataTypes mapped to the same ApplicationDataType
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1010]	If nativeDeclaration does not exist
[constr_1011]	category of SwBaseType
[constr_1012]	Value of category is FIXED_LENGTH
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDataDefProps
[constr_1016]	Restriction of invalidValue for ImplementationDataType and ImplementationDataTypeElement

▽

△

Number	Heading
[constr_1017]	Supported combinations of swImplPolicy and swCalibrationAccess
[constr_1018]	measurementPoint shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess
[constr_1020]	ParameterDataPrototype needs to be of compatible data type as referenced in sharedAxisType
[constr_1022]	Limits shall be defined for each direction of CompuMethod
[constr_1024]	Stepwise definition of CompuMethods
[constr_1025]	Avoid division by zero in rational formula
[constr_1026]	Compatibility of Units
[constr_1029]	ConstantSpecificationMapping and ConstantSpecification
[constr_1033]	Communication scenarios for sender/receiver communication
[constr_1035]	Recursive definition of CompositionSwComponentType
[constr_1036]	Connect kinds of PortInterfaces
[constr_1037]	Client shall not be connected to multiple servers
[constr_1038]	Reference to ApplicationError
[constr_1039]	Relevance of swImplPolicy
[constr_1040]	Conversion of SenderReceiverInterfaces
[constr_1041]	Conversion of ClientServerInterfaces
[constr_1043]	PortInterface vs. ComSpec
[constr_1044]	Applicability of DataFilter
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1046]	Applicability of [constr_1045]
[constr_1047]	Compatibility of ApplicationPrimitiveDataTypes
[constr_1048]	Compatibility of ApplicationRecordDataTypes
[constr_1049]	Compatibility of ApplicationArrayDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1051]	Compatibility of SwDataDefProps
[constr_1052]	Compatibility of Units
[constr_1053]	Compatibility of PhysicalDimensions
[constr_1054]	No DataConstr available at the provider
[constr_1055]	ImplementationDataType has category VALUE
[constr_1056]	ImplementationDataType has category TYPE_REFERENCE
[constr_1057]	ImplementationDataType has category DATA_REFERENCE
[constr_1058]	ImplementationDataType has category FUNCTION_REFERENCE
[constr_1059]	Compatibility of data types with category VALUE
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1064]	Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, or CUBE_5

▽

△

Number	Heading
[constr_1066]	Forbidden mappings to ImplementationDataType
[constr_1068]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types
[constr_1069]	Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors
[constr_1070]	Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors
[constr_1071]	compatibility of ParameterDataPrototype and VariableDataPrototype
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1075]	Compatibility of ModeDeclarationGroups
[constr_1076]	Compatibility of ArgumentDataPrototypes
[constr_1077]	Compatibility of ApplicationErrors
[constr_1078]	Compatibility of ClientServerOperations
[constr_1079]	Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector
[constr_1080]	Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector
[constr_1081]	Compatibility of TriggerInterfaces in the context of an AssemblySwConnector
[constr_1082]	Compatibility of TriggerInterfaces in the context of an DelegationSwConnector
[constr_1083]	Compatibility of Triggers
[constr_1084]	delegation of a provided outer PortPrototype
[constr_1086]	SwConnector between two specific PortPrototypes
[constr_1087]	AssemblySwConnector inside CompositionSwComponentType
[constr_1088]	DelegationSwConnector inside CompositionSwComponentType
[constr_1092]	ParameterSwComponentType
[constr_1093]	Definition of textual strings
[constr_1095]	Values of nDataSets vs. reliability
[constr_1096]	SwcModeSwitchEvent and WaitPoint
[constr_1097]	RunnableEntity that has a WaitPoint
[constr_1098]	Mode switch and mode disabling
[constr_1100]	Unconnected RPortPrototype typed by a DataInterface
[constr_1101]	Mode-related communication
[constr_1102]	ApplicationError in the scope of one SwComponentType
[constr_1103]	NonqueuedReceiverComSpec and enableUpdate
[constr_1104]	Trigger sink and trigger source

▽

△

Number	Heading
[constr_1105]	Value of arraySize
[constr_1106]	Structure shall have at least one element
[constr_1107]	Union shall have at least one element
[constr_1108]	Value of ApplicationError.errorCode
[constr_1109]	Mapping of SwComponentPrototypes typed by a SensorActuatorSwComponentType
[constr_1111]	Constraints of dataId in PROFILE_01
[constr_1112]	Constraints of dataIdMode in PROFILE_01
[constr_1113]	Existence of attributes in PROFILE_01
[constr_1114]	Constraints of crcOffset in PROFILE_01
[constr_1115]	Constraints of counterOffset in PROFILE_01
[constr_1116]	Constraints of dataLength in PROFILE_01
[constr_1117]	Constraints of maxDeltaCounterInit in PROFILE_01
[constr_1118]	Existence of attributes in PROFILE_02
[constr_1119]	Constraints of dataLength in PROFILE_02
[constr_1120]	Constraints of dataId in PROFILE_02
[constr_1121]	Constraints of maxDeltaCounterInit in PROFILE_02
[constr_1126]	Compatibility of DataConstrs
[constr_1128]	Queue length of ClientServerOperations associated with the same RunnableEntity
[constr_1129]	swImplPolicy and NonqueuedReceiverComSpec
[constr_1130]	swImplPolicy and QueuedReceiverComSpec
[constr_1131]	swImplPolicy and NonqueuedSenderComSpec
[constr_1132]	swImplPolicy and QueuedSenderComSpec
[constr_1134]	Allowed structure of TEXTTABLE
[constr_1135]	Limit of vt in BITFIELD_TEXTTABLE
[constr_1137]	Applicability of ParameterInterface
[constr_1138]	assignedPort and DiagEventDebounceMonitorInternal
[constr_1139]	assignedPort of DiagEventDebounceMonitorInternal shall refer to an RPortPrototype
[constr_1140]	Combination of invalidValue with the attribute handleInvalid
[constr_1141]	Applicability of the scope attribute
[constr_1142]	category of CompuMethod shall not be extended
[constr_1144]	SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType
[constr_1146]	Applicability of a symbol for a CompuScale in C code
[constr_1147]	Standardized values for the attribute category of meta-class PortGroup
[constr_1148]	PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes
[constr_1149]	PortPrototypes used for NV data management

▽

△

Number	Heading
[constr_1150]	Usage of valueType for PortDefinedArgumentValue
[constr_1151]	Applicability of PortInterfaceMapping
[constr_1153]	Applicability of compatibility requirements for CompuScales
[constr_1154]	Compatibility of CompuScales for sender-receiver communication and similar use cases
[constr_1155]	Compatibility of CompuScales for client-server communication
[constr_1156]	Relevance of "names" of CompuScales
[constr_1158]	Applicable categorys for attribute ImplementationDataType.swDataDefProps.compuMethod
[constr_1159]	Consistency of VariableAndParameterInterfaceMapping with respect to the referenced DataInterfaces
[constr_1161]	Applicability of the attribute Ref.index
[constr_1162]	Compatibility of SwRecordLayouts
[constr_1163]	Compatibility of CompuMethods
[constr_1164]	Number of arguments owned by a RunnableEntity
[constr_1165]	Applicability of RunnableEntityArgument
[constr_1166]	Restrictions of ModeRequestTypeMap
[constr_1167]	ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType
[constr_1168]	Compatibility of ImplementationDataTypes used in the ModeRequestTypeMap
[constr_1169]	Allowed values for Trigger.swImplPolicy
[constr_1172]	Allowed values of SwCalibrationAccessEnum for ModeDeclarationGroupPrototype
[constr_1173]	Applicability of AutosarParameterRef referencing a VariableDataPrototype
[constr_1174]	PortInterfaces used in the context of CompositionSwComponentTypes cannot refer to AUTOSAR services
[constr_1175]	Depending on its category, CompuMethod shall refer to a unit
[constr_1176]	Compatibility of CompuScales of category LINEAR and RAT_FUNC
[constr_1177]	Allowed targetCategory for SwPointerTargetProps
[constr_1178]	Existence of attributes of SwDataDefProps in the context of ImplementationDataType
[constr_1181]	Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue
[constr_1182]	Allowed values for InternalTriggeringPoint.swImplPolicy
[constr_1183]	EndToEndProtectionVariablePrototypes aggregated by EndToEndProtection
[constr_1184]	Consistency of rootDataPrototype and base in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1185]	Consistency of data types in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1186]	Consistency of data types in the context of ArVariableInImplementationDataInstanceRef

▽

△

Number	Heading
[constr_1187]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types
[constr_1188]	Existence of ReceiverComSpec.replaceWith
[constr_1191]	Value of Limit shall yield a numerical value
[constr_1192]	Compatibility of "IDENTICAL" to "RAT_FUNC" or "LINEAR"
[constr_1193]	ModeDeclaration shall be referenced by at least one ModeTransition in the role enteredMode
[constr_1194]	Identical ModeTransitions
[constr_1195]	SwcModeSwitchEvent and the definition of ModeTransition
[constr_1196]	Existence of networkRepresentation vs. compositeNetworkRepresentation
[constr_1197]	Existence of compositeNetworkRepresentation shall be comprehensive
[constr_1200]	Queued communication is not applicable for dataElements owned by PRPortPrototype
[constr_1202]	Supported connections by AssemblySwConnector for PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
[constr_1203]	Supported connections by DelegationSwConnector for PortPrototypes typed by a SenderReceiverInterface or NvDataInterface
[constr_1204]	Supported connections by AssemblySwConnector for PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface
[constr_1205]	Supported connections by DelegationSwConnector for PortPrototypes typed by a ClientServerInterface, ModeSwitchInterface, or TriggerInterface
[constr_1209]	Mapping of ModeDeclarations of mode user to ModeDeclaration of mode manager
[constr_1210]	Mapping of ModeDeclarations of mode user to **all** ModeDeclarations of mode manager
[constr_1211]	Constraints of maxNoNewOrRepeatedData in PROFILE_01
[constr_1212]	Constraints of syncCounterInit in PROFILE_01
[constr_1213]	Constraints of maxNoNewOrRepeatedData in PROFILE_02
[constr_1214]	Constraints of syncCounterInit in PROFILE_02
[constr_1219]	Invalidation depends on the value of swImplPolicy
[constr_1220]	Compatibility of SwBaseType
[constr_1221]	DataPrototype is typed by an ApplicationPrimitiveDataType
[constr_1222]	category of an AutosarDataType used to type a DataPrototype is set to STRING
[constr_1223]	DataPrototype is typed by an ApplicationRecordDataType
[constr_1224]	DataPrototype is typed by an ApplicationArrayDataType
[constr_1225]	DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE
[constr_1226]	Applicable range for ExecutableEntityActivationReason.bitPosition

▽

△

Number	Heading
[constr_1227]	Value of attribute ExecutableEntityActivationReason.bitPosition shall be unique
[constr_1228]	RTEEvent that is referenced by a WaitPoint in the role trigger shall not reference ExecutableEntityActivationReason
[constr_1229]	category of ImplementationDataType boils down to VALUE
[constr_1230]	ApplicationDataType that qualifies for Integral Primitive Type
[constr_1231]	ConsistencyNeeds aggregated by CompositionSwComponentType
[constr_1232]	ConsistencyNeeds aggregated by AtomicSwComponentType
[constr_1233]	InstantiationTimingEventProps shall only reference TimingEvent
[constr_1234]	Value of RunnableEntity.symbol
[constr_1237]	Scope of mapped ClientServerOperations in the context of a ClientServerOperationMapping
[constr_1238]	Scope of mapped ApplicationErrors in the context of a ClientServerOperationMapping
[constr_1240]	Consistency of ArgumentDataPrototypes within the context of a ClientServerOperationMapping
[constr_1241]	Compound Primitive Data Types and invalidValue
[constr_1242]	Restriction of invalidValue for ApplicationPrimitiveDataType of category STRING
[constr_1243]	NumericalOrText shall either define vf or vt
[constr_1244]	DataPrototypes used in application software shall not be typed by C enums
[constr_1245]	Consideration of ModeTransitions for the compatibility of ModeDeclarationGroups
[constr_1246]	Consistency of firstMode and secondMode in the scope of one ModeDeclarationMappingSet
[constr_1247]	Consistency of ModeDeclarationMappingSet with respect to the referenced firstModeGroup and secondModeGroup
[constr_1248]	Compatibility of PortPrototypes of different DataInterfaces in the context of a PassThroughSwConnector
[constr_1249]	Compatibility of ModeSwitchInterfaces in the context of a PassThroughSwConnector
[constr_1250]	Compatibility of ClientServerInterfaces in the context of a PassThroughSwConnector
[constr_1251]	Compatibility of PortPrototypes of TriggerInterfaces in the context of a PassThroughSwConnector
[constr_1252]	Creation of a loop involving a PassThroughSwConnector is not allowed
[constr_1253]	Supported usage of VariationPointProxy
[constr_1254]	Definition of a pointer to a pointer
[constr_1255]	ApplicationPrimitiveDataTypes of category BOOLEAN and STRING
[constr_1256]	Acknowledgement feedback in n:1 writer case
[constr_1257]	No WaitPoints allowed
[constr_1258]	Value of minimumStartInterval for RunnableEntitys triggered by an InitEvent

▽

△

Number	Heading
[constr_1259]	Aggregation of AsynchronousServerCallPoint **and** AsynchronousServerCallResultPoint
[constr_1260]	No mode disabling for InitEvents
[constr_1261]	Applicability for EndToEndDescription.dataIdNibbleOffset
[constr_1263]	Existence of ModeErrorBehavior.defaultMode
[constr_1264]	Iteration along output axis is only supported for VALUE and VAL_BLK
[constr_1268]	ArgumentDataPrototype.direction shall be preserved in a ClientServerOperationMapping
[constr_1269]	Number of arguments shall be preserved in a ClientServerOperationMapping
[constr_1270]	ArgumentDataPrototype shall be mapped only once in a ClientServerOperationMapping
[constr_1271]	RecordValueSpecification.fields shall be identical to the number of ApplicationRecordDataType.elements
[constr_1272]	RecordValueSpecification.fields shall be identical to the number of subElements of ImplementationDataType of category STRUCTURE
[constr_1273]	Rules for the initialization of ApplicationArrayDataType by means of ArrayValueSpecification
[constr_1274]	Rules for the initialization of array-shaped ImplementationDataType with a fixed size by means of ArrayValueSpecification
[constr_1277]	SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue
[constr_1278]	PhysConstrs references a Unit
[constr_1279]	Unmapped elements of ApplicationCompositeDataTypes or ImplementationDataTypes and the attribute swImplPolicy
[constr_1280]	Unmapped dataElement on the “target” end shall have an initValue
[constr_1282]	Restriction concerning the usage of RuleBasedValueSpecification or a ReferenceValueSpecification for the specification of an invalidValue
[constr_1284]	Limitation of the use of TextValueSpecification
[constr_1285]	Applicability of roles vs. PortPrototypes
[constr_1286]	serverArgumentImplPolicy and ArgumentDataPrototype typed by primitive data types
[constr_1287]	Compatibility of SenderReceiverInterfaces with respect to invalidationPolicy
[constr_1288]	Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes
[constr_1289]	Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes
[constr_1290]	Limitation on the number of PPortComSpecs in the context of one PPortPrototype
[constr_1291]	Limitation on the number of RPortComSpecs in the context of one PPortPrototype

▽

△

Number	Heading
[constr_1292]	Limitation on the number of RPortComSpecs/PPortComSpecs in the context of one PRPortPrototype
[constr_1295]	PortInterfaces and category DATA_REFERENCE
[constr_1296]	DataPrototypes used as explicitInterRunnableVariable or implicitInterRunnableVariable and category DATA_REFERENCE
[constr_1298]	Existence of attributes if category of a ModeDeclarationGroup is set to EXPLICIT_ORDER
[constr_1299]	Existence of attributes if category of a ModeDeclarationGroup is set to other than EXPLICIT_ORDER
[constr_1300]	Primitive DataPrototype on the "source" end shall not be mapped to element of a composite data type on the "target" end of the SwConnector
[constr_1301]	Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role
[constr_1302]	Restriction of data invalidation
[constr_1303]	Applicability of TextTableMapping depending on the value of CompuMethod.category
[constr_1304]	Existence of attribute bitfieldTextTableMaskFirst
[constr_1305]	Existence of attribute bitfieldTextTableMaskSecond
[constr_1306]	Limitation of TextTableMapping for CompuMethods that have the value of category set to BITFIELD_TEXTTABLE
[constr_1307]	Consistency of values and masks in TextTableMapping
[constr_1308]	Existence of NvBlockNeeds.cyclicWritingPeriod
[constr_1309]	Existence of NvBlockDescriptor.timingEvent
[constr_1310]	Existence of attributes of meta-class NvBlockNeeds
[constr_1311]	Appearance of safety-related possible values of MemorySection.option or SwAddrMethod.option
[constr_1312]	PortPrototypes typed by a ParameterInterface
[constr_1313]	Completeness of TextTableMapping for the values of a given bit mask on the sender side
[constr_1314]	Profile VSA_LINEAR for ApplicationArrayDataType
[constr_1315]	Profile VSA_SQUARE for ApplicationArrayDataType
[constr_1316]	Profile VSA_RECTANGULAR for ApplicationArrayDataType
[constr_1317]	Profile VSA_FULLY_FLEXIBLE for ApplicationArrayDataType
[constr_1318]	Profile VSA_LINEAR for ImplementationDataType
[constr_1319]	Profile VSA_SQUARE for ImplementationDataType
[constr_1320]	Profile VSA_RECTANGULAR for ImplementationDataType
[constr_1321]	Profile VSA_FULLY_FLEXIBLE for ImplementationDataType
[constr_1322]	Size Indicator for undefined dynamicArraySizeProfile
[constr_1363]	Existence of attributes of DiagnosticValueNeeds
[constr_1364]	Existence of attributes of DiagnosticIoControlNeeds
[constr_1375]	Existence of attributes of CompuMethod and related meta-classes

▽

△

Number	Heading
[constr_1381]	Appearance of core-related possible values of MemorySection.option or SwAddrMethod.option
[constr_1382]	Mutually exclusive existence of attributes SwVariableRefProxy.autosarVariable vs. SwVariableRefProxy.mcDataInstanceVar
[constr_1383]	Existence of CompuMethod and DataConstr for ImplementationDataTypes of category TYPE_REFERENCE
[constr_1384]	Definition of invalidValue for DataPrototype typed by ApplicationPrimitiveDataType of category CURVE, MAP, CUBOID, CUBE_4, CUBE_5, COM_AXIS, RES_AXIS, and VAL_BLK
[constr_1385]	DataPrototype is typed by an ImplementationDataType
[constr_1386]	PortDefinedArgumentValue shall only be defined for AbstractProvidedPortPrototype
[constr_1388]	VariationPointProxy of category VALUE shall not mix "pre-build" and "post-build" use-cases
[constr_1389]	Restriction regarding the value of category of VariationPointProxy.implementationDataType
[constr_1390]	Restriction to the value of SenderReceiverInterface.invalidationPolicy.handleInvalid
[constr_1391]	Compatibility of Units in the context of assignment using an ApplicationValueSpecification
[constr_1392]	Compatibility of Units in the context of assignment using an ApplicationRuleBasedValueSpecification
[constr_1393]	Existence of RuleBasedValueCont.unit
[constr_1395]	NvBlockDataMapping shall be complete
[constr_1396]	Restriction for the value of attribute category for non-terminating ImplementationDataTypeElements taken to model a Variable-Size Array Data Type
[constr_1397]	Existence of attributes of TransformerHardErrorEvent
[constr_1398]	Existence of attributes of BaseTypeDirectDefinition
[constr_1399]	Standardized values of ModeDeclarationGroup.category
[constr_1400]	Reference to a specific DataTransformation
[constr_1401]	Restrictions on the relation between DataPrototypeMapping and DataTransformation
[constr_1402]	Applicability of core-related possible values of MemorySection.option or SwAddrMethod.option related to SwAddrMethod.sectionInitializationPolicy
[constr_1403]	NvBlockDataMappings to a given nvData shall be unambiguous
[constr_1404]	All NvDataInterface.nvData of PortPrototypes in the context of a specific SwcServiceDependency shall be mapped to the same NvBlockDescriptor
[constr_1407]	Definition of SwDataDefProps.dataConstr depending on the capabilities of the data type
[constr_1408]	Definition of SwDataDefProps.displayFormat depending on the capabilities of the data type

▽

△

Number	Heading
[constr_1409]	Definition of SwDataDefProps.dataConstr depending on the capabilities of the element data type
[constr_1410]	Definition of SwDataDefProps.displayFormat depending on the capabilities of the element data type
[constr_1413]	Definition of SwDataDefProps.stepSize depending on the capabilities of the data type
[constr_1414]	Definition of SwDataDefProps.stepSize depending on the capabilities of the element data type
[constr_1415]	Supported values of ModeSwitchEventTriggeredActivity.role
[constr_1416]	Existence of ApplicationArrayElement.maxNumberOfElements
[constr_1417]	Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (I)
[constr_1418]	Invalid connection between NvBlockSwComponentType and other AtomicSwComponentType (II)
[constr_1420]	Existence of SwAxisIndividual.inputVariableType
[constr_1422]	Value of category is VOID
[constr_1423]	Completeness of references ArVariableInImplementationDataInstanceRef.contextDataPrototype
[constr_1424]	Existence of ArVariableInImplementationDataInstanceRef.contextDataPrototype
[constr_1425]	Definition of swCalprmAxisSet.swCalprmAxis / SwAxisIndividual.swVariableRef depending on the capabilities of the data type
[constr_1426]	Consistency of array sizes for axes and input variable array
[constr_1427]	Definition of swCalprmAxisSet.swCalprmAxis / SwAxisGrouped.swCalprmRef depending on the capabilities of the data type
[constr_1428]	Consistency of array sizes for arrays of elements of category CURVE, MAP, CUBOID, CUBE_4, or CUBE_5 arrays and used group axes arrays
[constr_1429]	Access to data within PortPrototypes from within RunnableEntitys
[constr_1430]	Access to local data from within RunnableEntitys
[constr_1431]	Access to parameters from within RunnableEntitys
[constr_1432]	Multiplicity of CommunicationBufferLocking
[constr_1433]	Transient faults are not applicable to software-components
[constr_1434]	CompuScales shall not have identical CompuScale Value Symbolic Names
[constr_1438]	ApplicationArrayElement.indexDataType needs to refer to a CompuMethod of category TEXTTABLE
[constr_1439]	Requirements on ApplicationArrayElement if attribute indexDataType exists
[constr_1440]	Size of the CompuMethod of category TEXTTABLE referenced by ApplicationArrayElement.indexDataType
[constr_1442]	category TYPE_REFERENCE shall not be used for modeling the “payload” of a Wrapped Union Data Type
[constr_1444]	Limited applicability of Wrapped Union Data Type

▽

△

Number	Heading
[constr_1445]	Initialization of the Member Selector of a Wrapped Union Data Type
[constr_1446]	No definition of invalidValue for a Wrapped Union Data Type
[constr_1468]	Limitation on the number of SwcExclusiveAreaPolicys
[constr_1469]	Applicability of constraints depending on the existence of a data transformation
[constr_1516]	Completeness of references ArParameterInImplementationDataInstanceRef.contextDataPrototype
[constr_1517]	Existence of ArParameterInImplementationDataInstanceRef.contextDataPrototype
[constr_1518]	Consistency of data types in the context of ArParameterInImplementationDataInstanceRef
[constr_1519]	Existence of attributes vs. category of ApplicationValueSpecification
[constr_1520]	Semantics of ObdRatioServiceNeeds.rateBasedMonitoredEvent
[constr_1521]	Reference from AsynchronousServerCallReturnsEvent to AsynchronousServerCallResultPoint
[constr_1523]	No mode disabling for OperationInvokedEvents
[constr_1538]	Restriction for ReceiverComSpec.dataElement
[constr_1539]	Restriction for SenderComSpec.dataElement
[constr_1540]	Existence of ClientComSpec.operation
[constr_1541]	Existence of ServerComSpec.operation
[constr_1544]	Modeling of SwAxisGeneric for the definition of a fix axis
[constr_1545]	No initialization for fix axis
[constr_1583]	PortInterfaceMapping for DataPrototype typed by Compound Primitive Data Type
[constr_1592]	Definition of SwDataDefProps.displayPresentation depending on the capabilities of the data type
[constr_1602]	Definition of SwDataDefProps.displayPresentation depending on the capabilities of the element
[constr_1607]	Only Wrapped Union Data Types in PortInterface
[constr_1608]	Existence of rootParameterDataPrototype
[constr_1609]	Existence of rootVariableDataPrototype
[constr_1610]	Existence of SwDataDefProps.swValueBlockSize and SwDataDefProps.swValueBlockSizeMult
[constr_1622]	Value of TimingEvent.offset vs. TimingEvent.period
[constr_1631]	Applicability of DataPrototypeMapping.secondToFirstDataTransformation
[constr_1632]	Restriction for firstToSecondDataTransformation and secondToFirstDataTransformation
[constr_1634]	Allowed combinations of ApplicationDataType.category vs. CompuMethod.category
[constr_1635]	Relevance of attribute isOptional
[constr_1636]	Mapping of data types that represent an Optional Element Structure

▽

△

Number	Heading
[constr_1637]	Existence of ImplementationDataTypeElement.isOptional vs. ImplementationDataType.isStructWithOptionalElement
[constr_1638]	First ImplementationDataTypeElement of ImplementationDataType that represents an Optional Element Structure
[constr_1639]	ImplementationDataTypeElement with attribute isOptional set to True
[constr_1640]	No use of Optional Element Structure for interaction with the diagnostic stack
[constr_1662]	Compatibility of ApplicationRecordDataType and ImplementationDataType that both represent an Optional Element Structure
[constr_1679]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = signalBasedDiagnostics
[constr_1680]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = AppModeRequestInterface
[constr_1681]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = VerificationStatus
[constr_1682]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xFacVdp
[constr_1683]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationCam
[constr_1684]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationMapem
[constr_1685]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationIvim
[constr_1686]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationSpatem
[constr_1694]	Allowed target of SwDataDefProps.implementationDataType
[constr_1706]	Definition of initial value for data transmission
[constr_1712]	Existence of attribute ArrayValueSpecification.intendedPartialInitializationCount
[constr_1713]	NvBlockDescriptor.writingStrategy.usedDataElement shall refer to AutosarDataPrototype
[constr_1714]	AutosarDataPrototype shall only be referenced by a single NvBlockDescriptor.writingStrategy
[constr_1715]	Possible values of attribute NvBlockDescriptor.writingStrategy.role
[constr_1716]	Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeAtShutdown
[constr_1717]	Consistency of attribute NvBlockDescriptor.writingStrategy.role set to storeImmediate

▽

△

Number	Heading
[constr_1718]	Inheritance of SwDataDefProps.dataConstr from an array data type to the array elements
[constr_1719]	Inheritance of SwDataDefProps.displayFormat from an array data type to the array elements
[constr_1720]	Inheritance of SwDataDefProps.stepSize from an array data type to the array elements
[constr_1724]	Usage of attribute ClientServerOperation.diagArgIntegrity
[constr_1726]	Ordering of MetaDataItemSet.metaDataItem
[constr_1735]	Limitation of the aggregation of AutosarVariableRef in the context of an NvBlockDataMapping owned by a BulkNvDataDescriptor
[constr_1741]	Restriction to explicit sending semantics for the usage of DataServices in the context of a SwcServiceDependency that aggregates DiagnosticValueNeeds that in turn is referenced by a DiagnosticIoControlNeeds
[constr_1754]	Aggregation of NumericalRuleBasedValueSpecification
[constr_1755]	Aggregation of CompositeRuleBasedValueSpecification
[constr_1771]	Existence of SwValueCont.unit
[constr_1773]	Value of attribute dataSendPoint.returnValueProvision
[constr_1774]	Value of attribute dataReceivePointByArgument.returnValueProvision
[constr_1775]	Value of attribute serverCallPoint.returnValueProvision
[constr_1776]	Value of attribute asynchronousServerCallResultPoint.returnValueProvision
[constr_1777]	Value of attribute externalTriggeringPoint.returnValueProvision
[constr_1778]	Value of attribute modeSwitchPoint.returnValueProvision
[constr_1779]	Scope of the definition of an AbstractRuleBasedValueSpecification
[constr_1783]	Existence of attribute ImplementationDataTypeElement.arrayImplPolicy
[constr_1860]	Multiplicity of DelegationSwConnector.innerPort
[constr_1861]	Multiplicity of DelegationSwConnector.outerPort
[constr_1862]	Multiplicity of PassThroughSwConnector.requiredOuterPort
[constr_1863]	Multiplicity of PassThroughSwConnector.providedOuterPort
[constr_1864]	Multiplicity of InstantiationRTEEventProps.refinedEvent
[constr_1865]	Existence of InvalidationPolicy.dataElement
[constr_1866]	Existence of MetaDataItem.length
[constr_1867]	Existence of MetaDataItem.metaDataItemType
[constr_1868]	Existence of MetaDataItemSet.dataElement
[constr_1869]	Existence of attribute ArgumentDataPrototype.direction
[constr_1870]	Existence of attribute ApplicationError.errorCode
[constr_1871]	Existence of attribute ModeRequestTypeMap.implementationDataType
[constr_1872]	Existence of attribute ModeRequestTypeMap.modeGroup
[constr_1888]	Existence of attribute DataTransformation.executeDespiteDataUnavailability
[constr_1889]	Existence of attribute QueuedReceiverComSpec.queueLength

▽

△

Number	Heading
[constr_1891]	Existence of attribute NonqueuedReceiverComSpec.initValue
[constr_1892]	Existence of attribute TransmissionAcknowledgementRequest.timeout
[constr_1895]	Existence of attribute ModeSwitchSenderComSpec.modeGroup
[constr_1896]	Existence of attribute ModeSwitchReceiverComSpec.modeGroup
[constr_1897]	Existence of reference ParameterProvideComSpec.parameter
[constr_1898]	Existence of reference ParameterRequireComSpec.parameter
[constr_1899]	Existence of reference NvRequireComSpec.variable
[constr_1900]	Existence of reference NvProvideComSpec.variable
[constr_1901]	Existence of attribute EndToEndDescription.category
[constr_1902]	Existence of attribute EndToEndProtection.endToEndProfile
[constr_1903]	Existence of reference DataTypeMap.applicationDataType
[constr_1904]	Existence of reference DataTypeMap.implementationDataType
[constr_1905]	Existence of attribute SwTextProps.arraySizeSemantics
[constr_1906]	Existence of attribute SwTextProps.swMaxTextSize
[constr_1909]	Existence of attribute ImplementationProps.symbol
[constr_1910]	Existence of attribute BaseType.baseTypeDefinition
[constr_1911]	Existence of ArVariableInImplementationDataInstanceRef.targetDataPrototype
[constr_1912]	Existence of reference ArParameterInImplementationDataInstanceRef.targetDataPrototype
[constr_1913]	Existence of attribute CompuRationalCoeffs.compuDenominator
[constr_1914]	Existence of attribute CompuRationalCoeffs.compuNumerator
[constr_1915]	Existence of attribute PhysicalDimensionMapping.firstPhysicalDimension
[constr_1916]	Existence of attribute PhysicalDimensionMapping.secondPhysicalDimension
[constr_1917]	Existence of ConstantSpecification.valueSpec
[constr_1918]	Existence of RecordValueSpecification.field
[constr_1919]	Existence of TextValueSpecification.value
[constr_1920]	Existence of NumericalValueSpecification.value
[constr_1921]	Existence of ReferenceValueSpecification.referenceValue
[constr_1923]	Existence of RuleBasedAxisCont.ruleBasedValues
[constr_1924]	Existence of RuleBasedValueCont.ruleBasedValues
[constr_1925]	Existence of NumericalRuleBasedValueSpecification.ruleBasedValues
[constr_1926]	Existence of RuleBasedValueSpecification.rule
[constr_1927]	Existence of RuleBasedValueSpecification.arguments
[constr_1928]	Existence of CompositeRuleBasedValueSpecification.rule
[constr_1929]	Existence of CompositeRuleBasedValueSpecification.argument
[constr_1930]	Existence of ConstantReference.constant
[constr_1931]	Existence of ConstantSpecificationMapping.applConstant

▽

△

Number	Heading
[constr_1932]	Existence of ConstantSpecificationMapping.implConstant
[constr_1933]	Existence of CalibrationParameterValue.initializedParameter
[constr_1935]	Existence of attribute SwcInternalBehavior.supportsMultipleInstantiation
[constr_1936]	Existence of attribute RunnableEntity.symbol
[constr_1938]	Existence of attribute RunnableEntityArgument.symbol
[constr_1939]	Existence of attribute ExecutableEntityActivationReason.bitPosition
[constr_1940]	Existence of attribute AsynchronousServerCallReturnsEvent.eventSource
[constr_1941]	Existence of attribute DataSendCompletedEvent.eventSource
[constr_1942]	Existence of attribute DataWriteCompletedEvent.eventSource
[constr_1943]	Existence of attribute DataReceivedEvent.data
[constr_1944]	Existence of attribute DataReceiveErrorEvent.data
[constr_1945]	Existence of attribute OperationInvokedEvent.operation
[constr_1952]	Existence of reference WaitPoint.trigger
[constr_1954]	Existence of attribute VariableAccess.accessedVariable
[constr_1955]	Existence of attribute ServerCallPoint.operation
[constr_1956]	Existence of attribute ServerCallPoint.timeout
[constr_1957]	Existence of attribute AsynchronousServerCallResultPoint.asynchronousServerCallPoint
[constr_1958]	Existence of attribute ParameterAccess.accessedParameter
[constr_1959]	Existence of attribute InstantiationDataDefProps.swDataDefProps
[constr_1960]	Existence of attribute PortAPIOption.port
[constr_1964]	Existence of attribute PerInstanceMemory.type
[constr_1965]	Existence of attribute PerInstanceMemory.typeDefinition
[constr_1973]	Existence of attribute ModeDeclarationGroup.initialMode
[constr_1974]	Existence of attribute ModeDeclarationGroup.modeDeclaration
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2002]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReadAccess
[constr_2003]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataWriteAccess
[constr_2004]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataSendPoint
[constr_2005]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument
[constr_2006]	Number of AsynchronousServerCallResultPoint referencing to one AsynchronousServerCallPoint
[constr_2007]	Consistency of typeDefinition attribute
[constr_2009]	Supported kinds of PortPrototypes of a NvBlockSwComponentType

▽

△

Number	Heading
[constr_2010]	Connections between SwComponentPrototypes of type NvBlockSwComponentType
[constr_2012]	Compatibility of ImplementationDataTypes used for ramBlock and romBlock
[constr_2013]	Compatibility of ImplementationDataTypes for NvBlockDataMapping
[constr_2015]	Limitation of SwcInternalBehavior of a NvBlockSwComponentType
[constr_2016]	Connections between SwComponentPrototypes of type ServiceProxySwComponentType
[constr_2017]	Ports of ServiceProxySwComponentTypes
[constr_2018]	Supported remote communication of a ServiceProxySwComponentType
[constr_2019]	ServiceSwComponentType shall have service ports only
[constr_2020]	dataReadAccess can not be used for queued communication
[constr_2021]	WaitPoint referencing a DataReceivedEvent can not be used for non-queued communication
[constr_2022]	Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints
[constr_2023]	Consistency of timeout values
[constr_2024]	enableTakeAddress is restricted to single instantiation
[constr_2026]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role writtenLocalVariable and readLocalVariable
[constr_2027]	SwcServiceDependency shall be defined for service ports only
[constr_2028]	staticMemory is restricted to single instantiation
[constr_2030]	AsynchronousServerCallResultPoint combined with WaitPoint shall belong to the same RunnableEntity
[constr_2031]	Period of TimingEvent shall be greater than 0
[constr_2033]	Timeout of DataSendCompletedEvent
[constr_2034]	SwAddrMethod referenced by RunnableEntitys, BswCalledEntitys, or BswSchedulableEntitys
[constr_2035]	swImplPolicy for VariableDataPrototype in SenderReceiverInterface
[constr_2036]	swImplPolicy for VariableDataPrototype in NvDataInterface
[constr_2037]	swImplPolicy for VariableDataPrototype in the role ramBlock
[constr_2038]	swImplPolicy for VariableDataPrototype in the role implicitInterRunnableVariable
[constr_2039]	swImplPolicy for VariableDataPrototype in the role explicitInterRunnableVariable
[constr_2040]	swImplPolicy for VariableDataPrototype in the role arTypedPerInstanceMemory
[constr_2041]	swImplPolicy for VariableDataPrototype in the role staticMemory
[constr_2042]	swImplPolicy for ParameterDataPrototype in ParameterInterface
[constr_2043]	swImplPolicy for ParameterDataPrototype in the role romBlock
[constr_2044]	swImplPolicy for ParameterDataPrototype in the role sharedParameter
[constr_2045]	swImplPolicy for ParameterDataPrototype in the role perInstanceParameter

▽

△

Number	Heading
[constr_2046]	swImplPolicy for ParameterDataPrototype in the role constantMemory
[constr_2047]	swImplPolicy for ArgumentDataPrototype
[constr_2048]	swImplPolicy for SwServiceArg
[constr_2049]	Different ModeDeclarationGroups shall have different shortNames.
[constr_2050]	Mandatory information of a SwAxisCont
[constr_2051]	Mandatory information of a SwValueCont
[constr_2052]	Values of swArraysize and the number of values provided by swValuesPhys shall be consistent.
[constr_2053]	Consistency between role IUMPRNumerator and ObdRatioServiceNeeds.connectionType
[constr_2054]	Valid targets of rptSystem
[constr_2055]	Valid targets of byPassPoint and rptHook reference
[constr_2056]	Consistency of RapidPrototypingScenario with respect to rptSystem and rptArHook references
[constr_2057]	Mandatory information of a RuleBasedAxisCont
[constr_2058]	Mandatory information of a RuleBasedValueCont
[constr_2535]	Target of an autosarParameter in AutosarParameterRef shall refer to a parameter
[constr_2536]	Target of an autosarVariable in AutosarVariableRef shall refer to a variable
[constr_2544]	Limits need to be consistent
[constr_2545]	invalidValue shall fit in the specified ranges
[constr_2548]	Data constraint of value axis shall match
[constr_2549]	Units of input axis shall be consistent
[constr_2550]	Units of value axis shall be consistent
[constr_2561]	Application of DataConstrRule.constrLevel
[constr_4002]	Unambiguous mapping of modes to data types
[constr_4003]	Semantics of SwcModeSwitchEvent
[constr_4004]	Context of SenderReceiverAnnotation
[constr_4005]	Context of ClientServerAnnotation
[constr_4006]	Context of ParameterPortAnnotation
[constr_4007]	Context of ModePortAnnotation
[constr_4008]	Context of TriggerPortAnnotation
[constr_4009]	Context of NvDataPortAnnotation
[constr_4012]	Timeout of ModeSwitchedAckEvent
[constr_4082]	RunnableEntity.reentrancyLevel shall not be set.
[constr_10005]	Existence of attribute NotAvailableValueSpecification.defaultPattern
[constr_10006]	Valid interval of attribute NotAvailableValueSpecification.defaultPattern
[constr_10009]	Aggregation of ApplicationRuleBasedValueSpecification
[constr_10016]	Applicability of OsTaskExecutionEvent

▽

△

Number	Heading
[constr_10017]	Existence of attribute SwAxisCont.category
[constr_10018]	Existence of attribute SwAxisCont.swAxisIndex
[constr_10019]	Existence of attribute SwAxisCont.swValuesPhys
[constr_10028]	Existence of reference stereotyped ≪isOfType≫

Table C.71: Changed Constraints in R21-11

C.14.6 Deleted Constraints in R21-11

Number	Heading
[constr_1001]	Value of dataId shall be unique
[constr_1008]	Applicability of categorys STRUCTURE and ARRAY
[constr_1027]	Types for record layouts
[constr_1030]	ParameterSwComponentType references ConstantSpecificationMappingSet
[constr_1031]	NvBlockSwComponentType references ConstantSpecificationMappingSet
[constr_1143]	category of AutosarDataType shall not be extended
[constr_1157]	Applicability of constraints of CompuScales
[constr_1160]	Size of Compound Primitive Data Type is variant
[constr_1281]	invalidValue is inside the scope of the compuMethod
[constr_1283]	invalidValue is outside the scope of the compuMethod
[constr_4000]	Local communication of mode switches
[constr_4035]	ValueSpecification shall fit into data type

Table C.72: Deleted Constraints in R21-11

C.15 Constraint History of this Document according to AUTOSAR R22-11

C.15.1 Added Traceables in R22-11

Number	Heading
[TPS_SWCT_01845]	Supported value encodings for SwBaseType
[TPS_SWCT_01846]	Use InstantiationDataDefProps to facilitate the definition of access to calibration parameters

▽

△

Number	Heading
[TPS_SWCT_01847]	Define role-specific data properties of elements of composite data types used for the definition of calibration parameters
[TPS_SWCT_01848]	Semantics of NvBlockDescriptor.instantiationDataDefProps
[TPS_SWCT_01849]	shortName of constantMemory and staticMemory
[TPS_SWCT_01850]	Usage of attribute RPortPrototype.requiredComSpec.maxDeltaCounterInit vs. EndToEndProtection.endToEndProfile.maxDeltaCounterInit
[TPS_SWCT_01851]	Usage of attribute RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData vs. EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData
[TPS_SWCT_01852]	Usage of attribute RPortPrototype.requiredComSpec.syncCounterInit vs. attribute EndToEndProtection.endToEndProfile.syncCounterInit
[TPS_SWCT_01853]	AtomicSwComponentType offers PortPrototype typed by SenderReceiverInterface or NvDataInterfaces to read/write the content of an entire DID via diagnostic services
[TPS_SWCT_01854]	Software-Components wants to initiate the cloning of a time base
[TPS_SWCT_01855]	Definition of initial value for data transmission
[TPS_SWCT_01856]	AtomicSwComponentType reads the current ECU mode
[TPS_SWCT_01857]	AtomicSwComponentType shall keep the ECU alive
[TPS_SWCT_01858]	Optional existence of reference VariationPointProxy.implementationDataType if the bindingTime is set to preCompileTime
[TPS_SWCT_01859]	Specification of constant value for a String modeled by ApplicationDataType
[TPS_SWCT_01860]	Specification of constant value for a String modeled by ImplementationDataType
[TPS_SWCT_01861]	Specification of baseTypeSize in the context of an ImplementationDataType where attribute category is set to ARRAY
[TPS_SWCT_01862]	AtomicSwComponentType wants information about DTC clearance
[TPS_SWCT_01863]	Existence of VariableInAtomicSWCTypeInstanceRef.rootVariableDataPrototype in the context of NvBlockDataMapping.nvRamBlockElement
[TPS_SWCT_01864]	Semantics of DiagnosticEventNeeds.inhibitingFid and DiagnosticEventNeeds.inhibitingSecondaryFid
[TPS_SWCT_01865]	Definition of attributes for an "embedded" axis if reference inputVariableType exists
[TPS_SWCT_01866]	Definition of attributes for an "embedded" axis if reference inputVariableType does not exist
[TPS_SWCT_01867]	Communication Use case of the V2X Data Manager where isService = true
[TPS_SWCT_01868]	Communication Use case of the V2X Data Manager where isService = false
[TPS_SWCT_01869]	V2x DM Use Case: Application software component receives portions of an ASN.1 message

▽

△

Number	Heading
[TPS_SWCT_01870]	AtomicSwComponentType wants to reset a degradation state
[TPS_SWCT_03502]	Software-component acts as a server and offers the service
[TPS_SWCT_03503]	Software-component receives notification on server event subscription status changes
[TPS_SWCT_03504]	Software-component acts as a client and subscribe to events and methods
[TPS_SWCT_03505]	Software-component receives notification on client event and method subscription status changes

Table C.73: Added Traceables in R22-11

C.15.2 Changed Traceables in R22-11

Number	Heading
[TPS_SWCT_01038]	Support for Variant Handling in the in Software Component Template
[TPS_SWCT_01094]	Standardized values of attribute EndToEndDescriptioncategory
[TPS_SWCT_01234]	Base Type Level
[TPS_SWCT_01236]	Big picture of data types
[TPS_SWCT_01260]	Applicability of SwBaseType for the definition of data types
[TPS_SWCT_01296]	Different approaches of ASAM MCD-2MC and AUTOSAR with respect to SwRecordLayout
[TPS_SWCT_01444]	Size of SwBaseType is specified in bits
[TPS_SWCT_01487]	Correspondence of invalidValue for ApplicationPrimitiveDataType and ImplementationDataType
[TPS_SWCT_01489]	Standardized values of SwRecordLayoutV.swRecordLayoutVProp
[TPS_SWCT_01549]	Definition of linear data scaling
[TPS_SWCT_01560]	Supported categorys of CompuMethods for data conversion
[TPS_SWCT_01654]	AtomicSwComponentType offers PortPrototypes typed by SenderReceiverInterfaces to adjust the IO signal via diagnostic services
[TPS_SWCT_01656]	Suffix used for the resulting name of the PortInterface for IOControlRequest and IOControlResponse
[TPS_SWCT_03500]	Status forwarding to data transformer

Table C.74: Changed Traceables in R22-11

C.15.3 Deleted Traceables in R22-11

Number	Heading
[TPS_SWCT_01261]	Use case for SwBaseType
[TPS_SWCT_01263]	Further use cases for SwBaseType
[TPS_SWCT_01275]	values of the attribute swImplPolicy are restricted depending on the context
[TPS_SWCT_01467]	ImplementationDataType references an SwBaseType with a string encoding
[TPS_SWCT_01517]	ClientServerOperation cannot be passed as a reference
[TPS_SWCT_01729]	V2xFac Use Case: V2xFac notifies application software component about received messages
[TPS_SWCT_01764]	V2xFac Use Case: Application software component shall be able to process the *MAP (topology) Extended Message*
[TPS_SWCT_01770]	V2xFac Use Case: Application software component processes Infrastructure to Vehicle Information Message
[TPS_SWCT_01788]	V2xFac Use Case: Application software component processes Signal Phase And Timing Extended Message
[TPS_SWCT_01803]	MetaDataItems define the same value of attribute length
[TPS_SWCT_02014]	AtomicSwComponentType supports Response On Event (ROE) via diagnostic services

Table C.75: Deleted Traceables in R22-11

C.15.4 Added Constraints in R22-11

Number	Heading
[constr_3688]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = ServerServiceOffer
[constr_3689]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = ClientEventSubscription
[constr_10096]	Shared axis shall not be a fixed axis
[constr_10097]	Buffer locking is only supported if returnValueProvision is set to returnValueProvided
[constr_10099]	Allowed values of the attribute SwDataDefProps.swImplPolicy vs. DataPrototypes and their roles
[constr_10104]	RoleBasedPortAssignment where attribute role is set to CallbackGetFaultDetectCounter shall refer to a PPortPrototype in the role portPrototype
[constr_10118]	Structural consistency of the modeling of InvalidationPolicy

▽

△

Number	Heading
[constr_10119]	SenderReceiverInterface.dataElement shall be referenced by at most one InvalidationPolicy
[constr_10120]	Structural consistency of the modeling of MetaDataItemSet
[constr_10121]	SenderReceiverInterface.dataElement shall be referenced by at most one MetaDataItemSet
[constr_10123]	Existence of attribute DtcStatusChangeNotificationNeeds.notificationTime
[constr_10196]	Definition of invalidValue for DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category TEXTTABLE or BITFIELD_TEXTTABLE
[constr_10372]	Relation between *Type of PortPrototype*, *Type of ComSpec*, and *Type of PortInterface*
[constr_10373]	ImplementationDataType of category VALUE shall not refer to SwBaseType of category VOID
[constr_10383]	Supported value encodings for SwBaseType in the context of PortInterfaces where attribute isService is set to false

Table C.76: Added Constraints in R22-11

C.15.5 Changed Constraints in R22-11

Number	Heading
[constr_1000]	End-to-end protection is limited to sender/receiver communication
[constr_1006]	Applicable data categories, depending on specific model elements related to data definition properties
[constr_1015]	Prioritization of SwDataDefProps
[constr_1026]	Compatibility of Units
[constr_1038]	Reference to ApplicationError
[constr_1043]	Allowed combinations of a specific *Type of PortInterface*, a specific *Type of PortPrototype*, and a specific *Type of ComSpec*
[constr_1046]	Applicability of [TPS_SWCT_01845]
[constr_1047]	Compatibility of ApplicationPrimitiveDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1052]	Compatibility of Units
[constr_1083]	Compatibility of Triggers
[constr_1091]	RTEEvents that may reference a WaitPoint
[constr_1092]	Restrictions for the ParameterSwComponentType
[constr_1104]	Trigger communication shall not implement an n:1 pattern
[constr_1108]	Existence and value of attribute ApplicationError.errorCode
[constr_1138]	SwcServiceDependency.assignedPort and DiagEventDebounceMonitorInternal

▽

△

Number	Heading
[constr_1170]	Existence of attribute EndToEndDescription.maxDeltaCounterInit for PROFILE_01
[constr_1171]	Existence of attribute EndToEndDescription.maxDeltaCounterInit for PROFILE_02
[constr_1177]	Allowed targetCategory for SwPointerTargetProps
[constr_1215]	Existence of attribute EndToEndDescription.maxNoNewOrRepeatedData for PROFILE_01
[constr_1216]	Existence of attribute EndToEndDescription.syncCounterInit for PROFILE_01
[constr_1217]	Existence of attribute EndToEndDescription.maxNoNewOrRepeatedData for PROFILE_02
[constr_1218]	Existence of attribute EndToEndDescription.syncCounterInit for PROFILE_02
[constr_1219]	Invalidation depends on the value of swImplPolicy
[constr_1290]	Limitation on the number of PPortComSpecs in the context of one PPortPrototype
[constr_1291]	Limitation on the number of RPortComSpecs in the context of one PPortPrototype
[constr_1292]	Limitation on the number of RPortComSpecs/PPortComSpecs in the context of one PRPortPrototype
[constr_1726]	Ordering of MetaDataItemSet.metaDataItem
[constr_2011]	Connections between SwComponentPrototypes typed by NvBlockSwComponentType and SwComponentPrototypes typed by other AtomicSwComponentTypes
[constr_2014]	Limitation of NvBlockDescriptor.clientServerPort.role
[constr_2031]	Value of TimingEvent.period shall be greater than 0

Table C.77: Changed Constraints in R22-11

C.15.6 Deleted Constraints in R22-11

Number	Heading
[constr_1014]	Supported value encodings for SwBaseType
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1090]	WaitPoint and RunnableEntity
[constr_1110]	Value of category in EndToEndDescription
[constr_1139]	assignedPort of DiagEventDebounceMonitorInternal shall refer to an RPortPrototype
[constr_1162]	Compatibility of SwRecordLayouts
[constr_1264]	Iteration along output axis is only supported for VALUE and VAL_BLK

▽

△

Number	Heading
[constr_1683]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationCam
[constr_1684]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationMapem
[constr_1685]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationIvim
[constr_1686]	Existence of attribute RoleBasedDataAssignment.usedDataElement.localVariable for RoleBasedDataAssignment.role = V2xApplRxIndicationSpatem
[constr_1706]	Definition of initial value for data transmission
[constr_1870]	Existence of attribute ApplicationError.errorCode
[constr_1937]	Existence of attribute TimingEvent.period
[constr_2029]	shortName of constantMemory and staticMemory
[constr_2048]	swImplPolicy for SwServiceArg
[constr_10028]	Existence of reference stereotyped ≪isOfType≫

Table C.78: Deleted Constraints in R22-11

D Modeling of InstanceRef

D.1 Introduction

The existence of so-called InstanceRefs is a direct consequence of the usage of the type-prototype pattern for modeling within AUTOSAR. When referencing a prototype, it is also necessary to include a reference to the prototypes typed by their corresponding types that in turn aggregate further prototypes to set up the context.

In other words, InstanceRefs are representing **structured references** that, on the one hand, consist of references to context prototypes (indicated by a subsetting or redefinition of atpContextElement) and finally a reference to the applicable target prototype (indicated by a redefinition of atpTarget).

Note that it is not uncommon to have more than a single context in the modeling of particular InstanceRefs.

For the reader of specifications, the modeling of InstanceRefs manifests as a UML dependency stereotyped ≪instanceRef≫ drawn from one meta-class to another. This is a simplified indication that the source of the dependency implements an InstanceRef to the meta-class at the target of the dependency.

Again, in most cases this is everything a reader needs to understand in order to figure out the modeling. The formal modeling of InstanceRefs is done by creating subclasses of the abstract meta-class AtpInstanceRef.

Wherever a more detailed understanding of the modeling is advised in the context of the specific chapter of this document, the modeling of a specific subclass of AtpInstanceRef is explained directly in the context of the corresponding chapter.

In all other cases, a deeper understanding of the modeling of particular subclasses of AtpInstanceRefs can be obtained from reading this chapter.

Class tables included in this chapter are not fully filled out in the sense that most of the notes inside the class tables are missing. The **primary** purpose of these class tables is to **provide information about the intended order in which InstanceRefs are serialized in M1 AUTOSAR models**.

In particular, the information about the order in serialized M1 models can be obtained from the value of the tag xml.sequenceOffset of each attribute of an InstanceRef meta-class.

For more information about the general concept of modeling AtpInstanceRef (e.g. the conceptual background of redefining or subsetting an association from a subclass of AtpInstanceRef to other meta-classes) please refer to [11].

D.2 Modeling

D.2.1 Components and Compositions

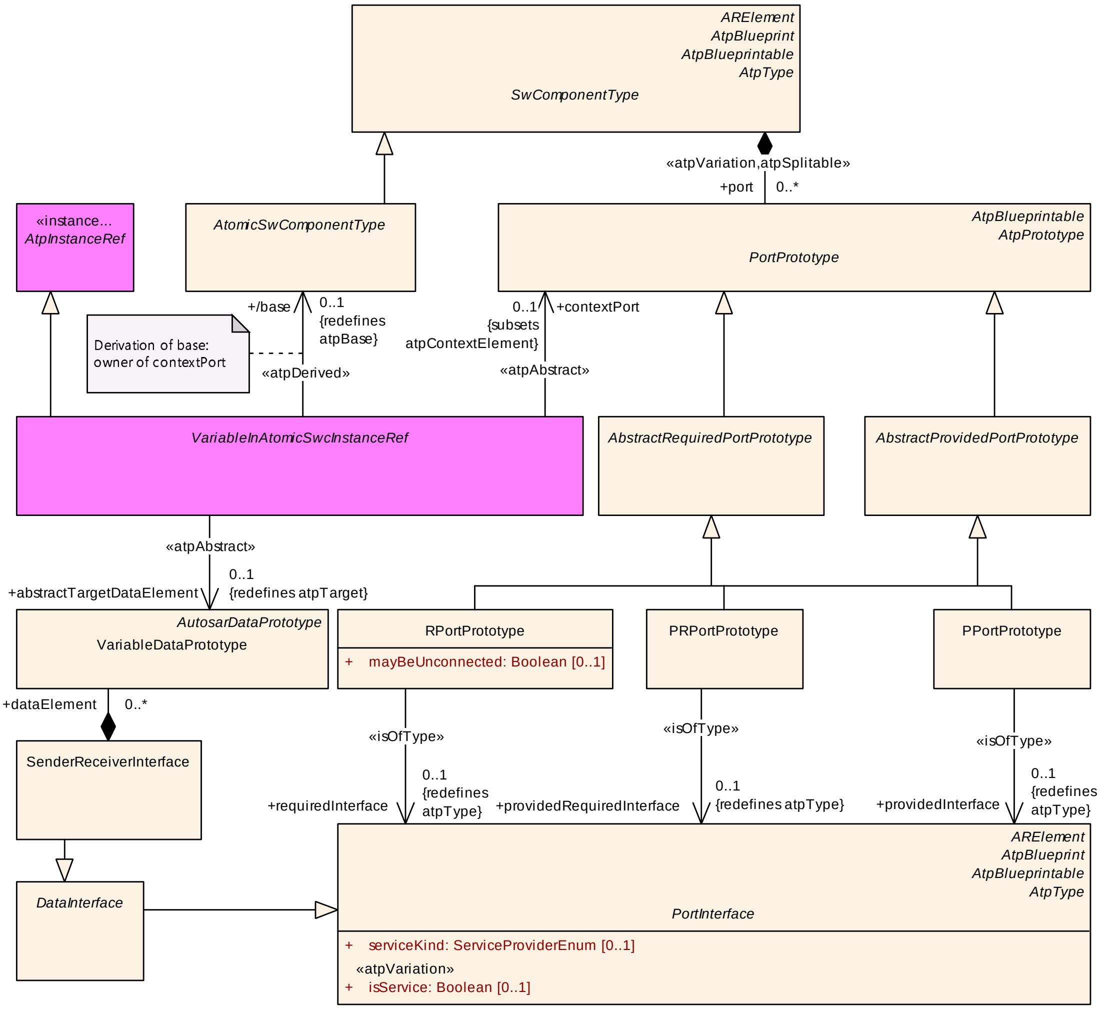


Figure D.1: Abstract modeling of references to VariableDataPrototype in the context of a AtomicSwComponentType

Class	***VariableInAtomicSwcInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Subclasses	RVariableInAtomicSwcInstanceRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
abstractTarget DataElement	VariableDataPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=30
base	AtomicSwComponent Type	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextPort	PortPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=20

Table D.1: VariableInAtomicSwcInstanceRef

Please note the example of how the redefinition of the context association works, i.e. the association from VariableInAtomicSwcInstanceRef to PortPrototype in the role contextPort is **redefined** by the subclass RVariableInAtomicSwcInstanceRef by means of an association to RPortPrototype in the role contextRPort.

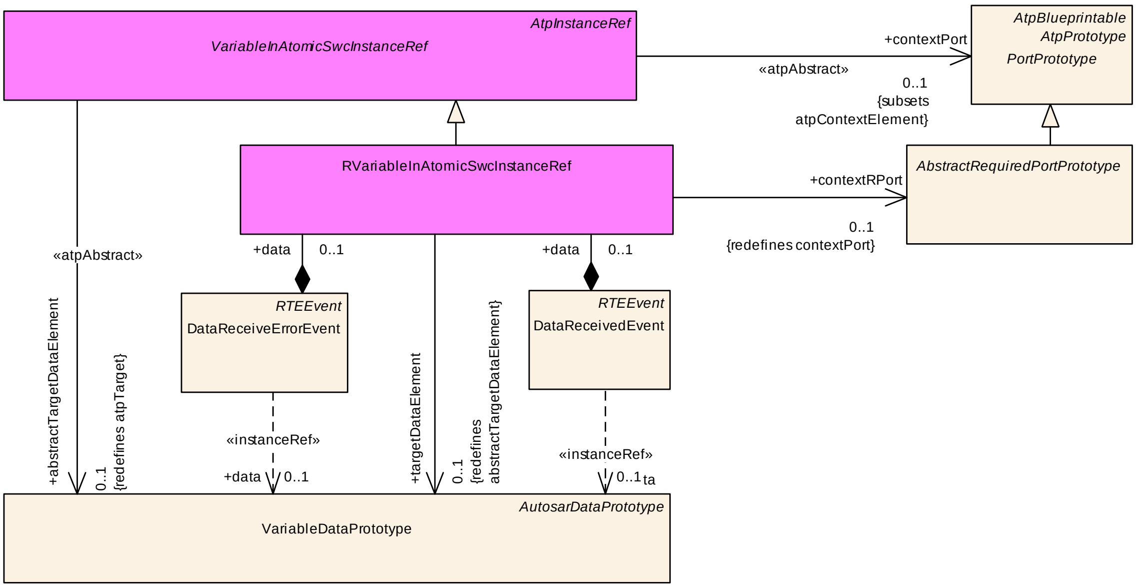


Figure D.2: Concrete modeling of references to VariableDataPrototype in the context of an RPortPrototype

The effect of this modeling is that the general relationship to PortPrototype is already established by VariableInAtomicSwcInstanceRef on an abstract level but actually it never makes the generated XML Schema because it is **redefined** by a subclass. In other words, the redefinition replaces the original association as far as the generation algorithm for the XML Schema is concerned.

For clarification, the interpretation of the values of `xml.sequenceOffset` in this particular case is that in the generated XML Schema the contextRPort comes first, followed by targetDataElement which concludes the definition of the `InstanceRef` in the XML Schema.

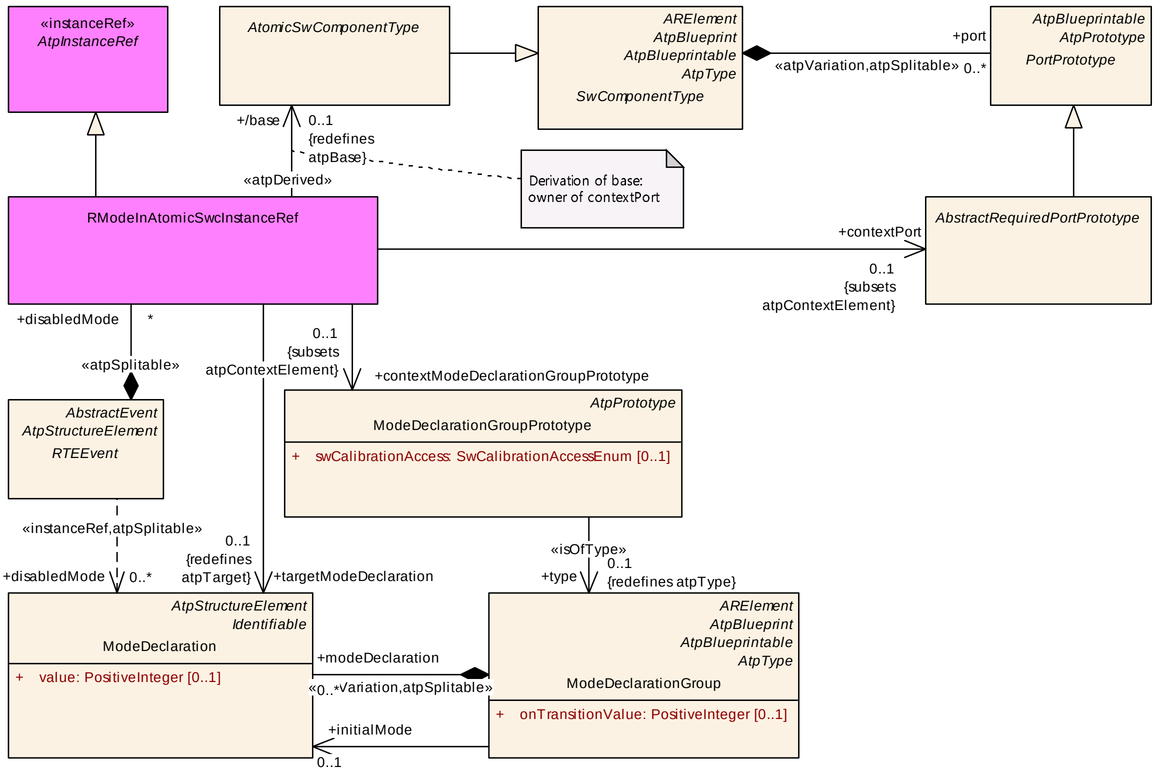


Figure D.3: Modeling of references to ModeDeclarationGroupPrototype in the context of an RPortPrototype

Class	**RVariableInAtomicSwcInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *VariableInAtomicSwcInstanceRef*			
Aggregated by	DataReceivedEvent.data, DataReceiveErrorEvent.data			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextRPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetData Element	VariableDataPrototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.2: RVariableInAtomicSwcInstanceRef

Class	**RModeInAtomicSwcInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	*RTEEvent*.disabledMode, SwcModeSwitchEvent.mode			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtomicSwComponent Type	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10

▽

△

Class	**RModeInAtomicSwcInstanceRef**			
contextMode Declaration GroupPrototype	ModeDeclarationGroup Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30
contextPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetMode Declaration	ModeDeclaration	0..1	ref	**Tags:**xml.sequenceOffset=40

Table D.3: RModeInAtomicSwcInstanceRef

Class	**InnerPortGroupInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	PortGroup.innerGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSw ComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
context (ordered)	SwComponent Prototype	*	ref	**Tags:**xml.sequenceOffset=20
target	PortGroup	0..1	ref	Links a PortGroup in a composition to another PortGroup, that is defined in a component which is part of this CompositionSwComponentType. There shall be at most one innerGroup per contained SwComponentPrototype. **Tags:**xml.sequenceOffset=30

Table D.4: InnerPortGroupInCompositionInstanceRef

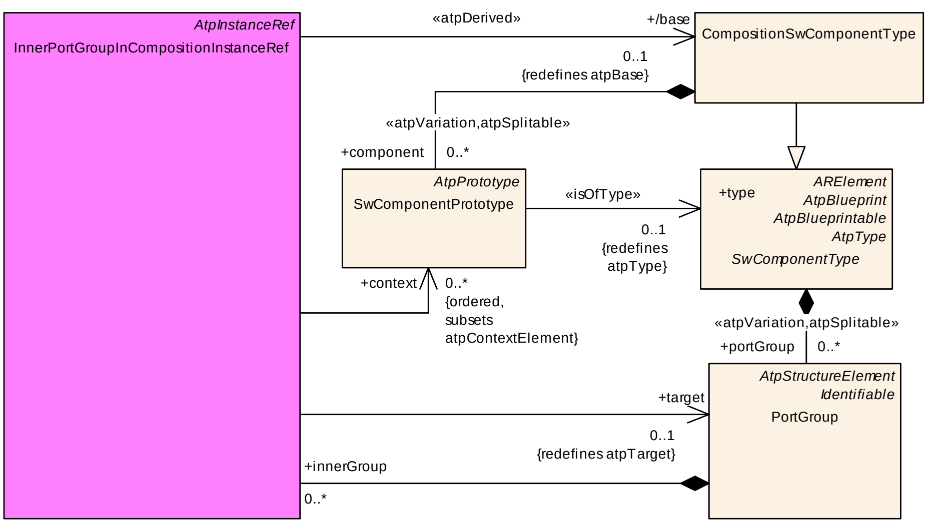


Figure D.4: Modeling of references to `PortGroup` in the context of a `CompositionSwComponentType`

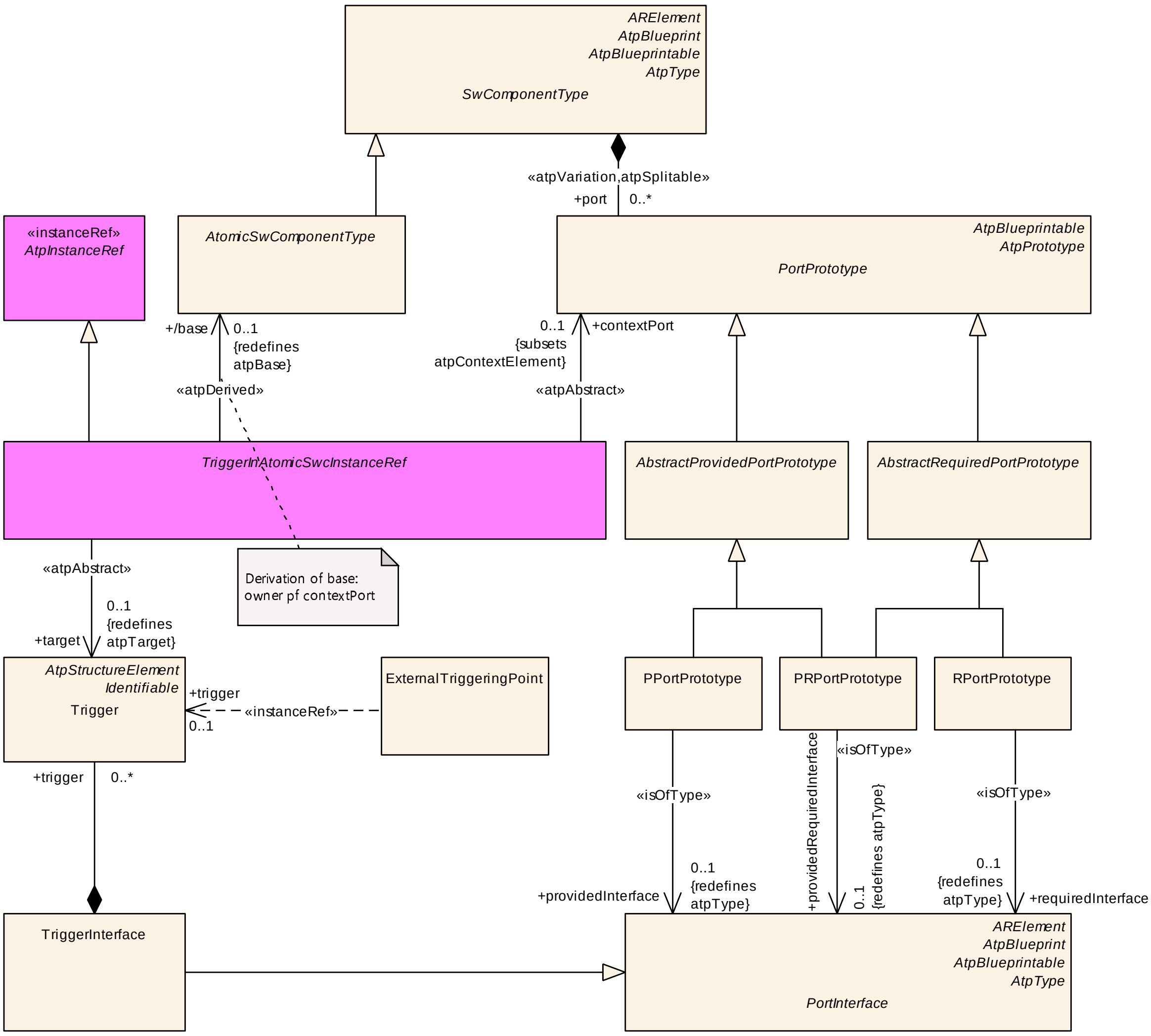


Figure D.5: Abstract modeling of references to Trigger in the context of a AtomicSwComponentType

Class	***TriggerInAtomicSwcInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Subclasses	PTriggerInAtomicSwcTypeInstanceRef, RTriggerInAtomicSwcInstanceRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtomicSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextPort	PortPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=20
target	Trigger	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=30

Table D.5: TriggerInAtomicSwcInstanceRef

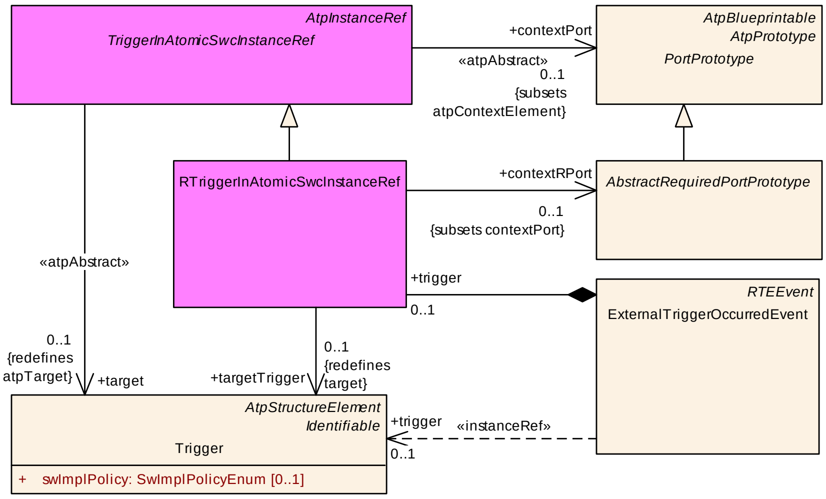


Figure D.6: Concrete modeling of references to Trigger in the context of an RPortPrototype

Class	RTriggerInAtomicSwcInstanceRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *TriggerInAtomicSwcInstanceRef*			
Aggregated by	ExternalTriggerOccurredEvent.trigger, TransformerHardErrorEvent.requiredTrigger			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
contextRPort	AbstractRequiredPortPrototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetTrigger	Trigger	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.6: RTriggerInAtomicSwcInstanceRef

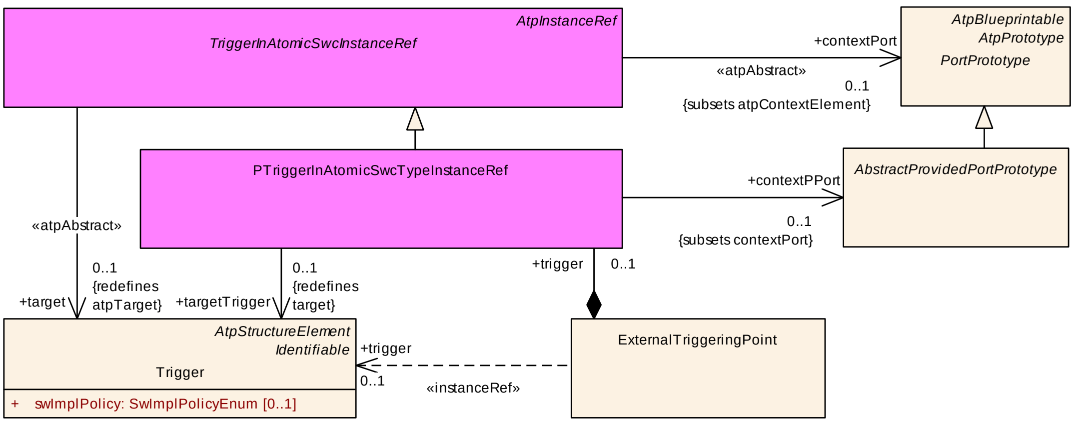


Figure D.7: Concrete modeling of references to Trigger in the context of a PPortPrototype

Class	**PTriggerInAtomicSwcTypeInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *TriggerInAtomicSwcInstanceRef*			
Aggregated by	ExternalTriggeringPoint.trigger, SwcBswSynchronizedTrigger.swcTrigger			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextPPort	AbstractProvidedPortPrototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetTrigger	Trigger	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.7: PTriggerInAtomicSwcTypeInstanceRef

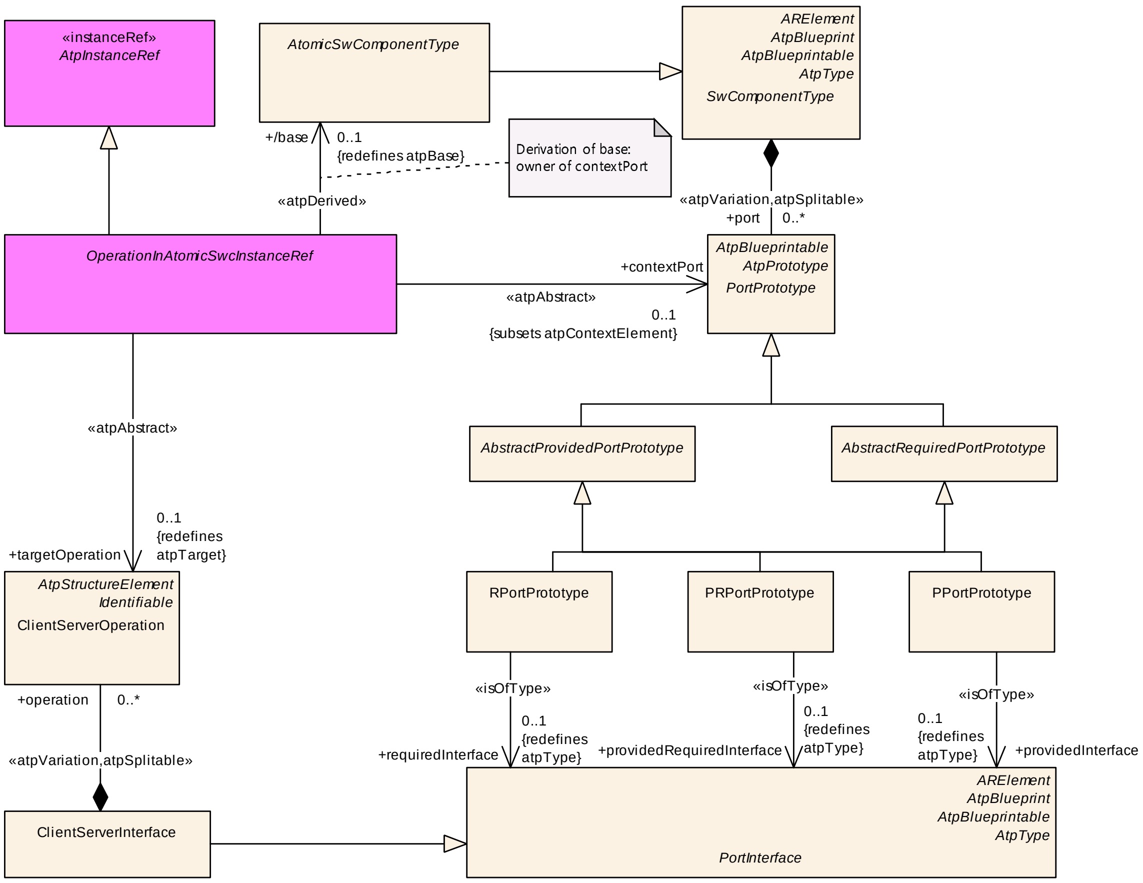


Figure D.8: Abstract modeling of references to `ClientServerOperation` in the context of a `AtomicSwComponentType`

Class	***OperationInAtomicSwcInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Subclasses	POperationInAtomicSwcInstanceRef, ROperationInAtomicSwcInstanceRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtomicSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10

△

Class	***OperationInAtomicSwcInstanceRef*** (abstract)			
contextPort	PortPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=20
targetOperation	ClientServerOperation	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=30

Table D.8: OperationInAtomicSwcInstanceRef

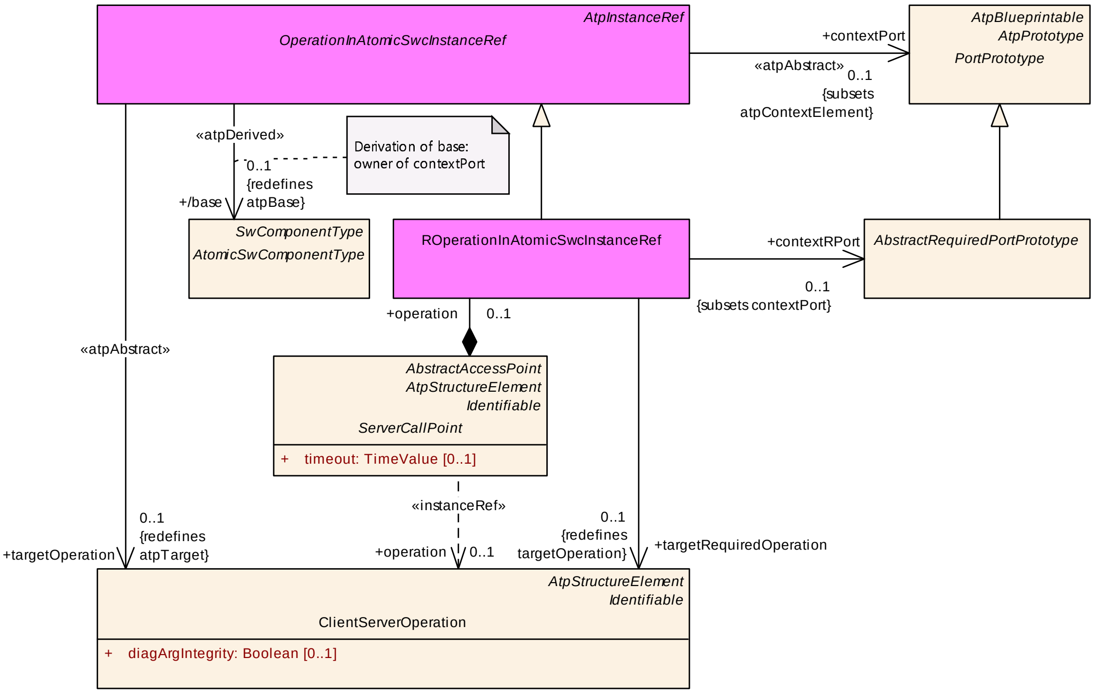


Figure D.9: Concrete modeling of references to `ClientServerOperation` in the context of an `RPortPrototype`

Class	**ROperationInAtomicSwcInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *OperationInAtomicSwcInstanceRef*			
Aggregated by	*ServerCallPoint*.operation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextRPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetRequired Operation	ClientServerOperation	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.9: ROperationInAtomicSwcInstanceRef

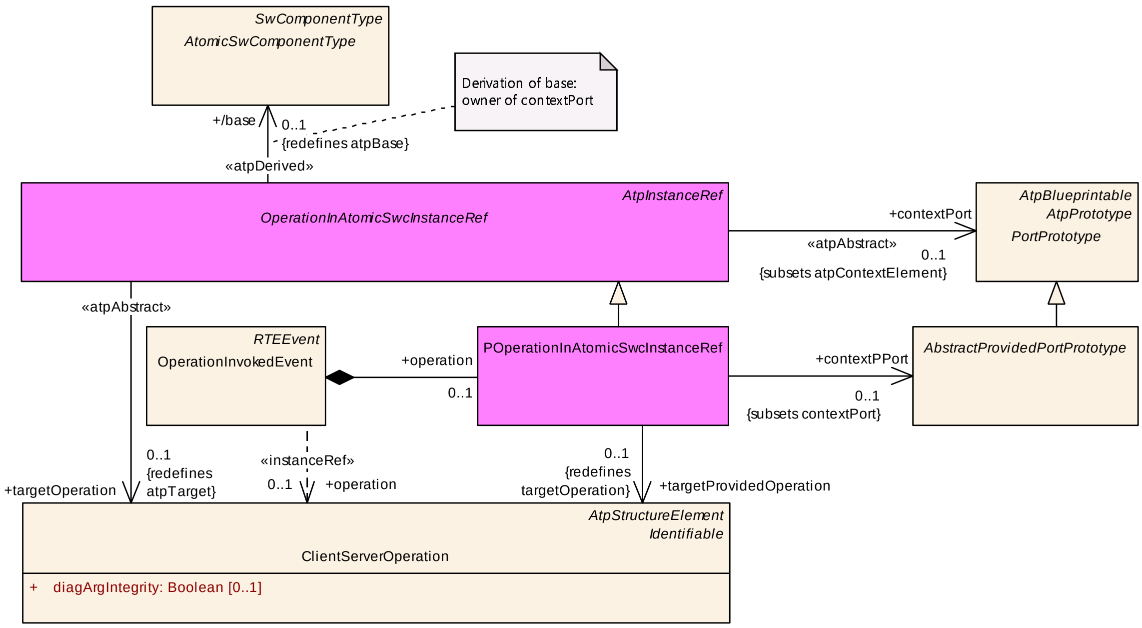


Figure D.10: Concrete modeling of references to ClientServerOperation in the context of a PPortPrototype

Class	**POperationInAtomicSwcInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *OperationInAtomicSwcInstanceRef*			
Aggregated by	OperationInvokedEvent.operation, TransformerHardErrorEvent.operation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextPPort	AbstractProvidedPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetProvided Operation	ClientServerOperation	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.10: POperationInAtomicSwcInstanceRef

Class	**RModeGroupInAtomicSWCInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *ModeGroupInAtomicSwcInstanceRef*			
Aggregated by	ModeAccessPoint.modeGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextRPort	AbstractRequiredPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetMode Group	ModeDeclarationGroup Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.11: RModeGroupInAtomicSWCInstanceRef

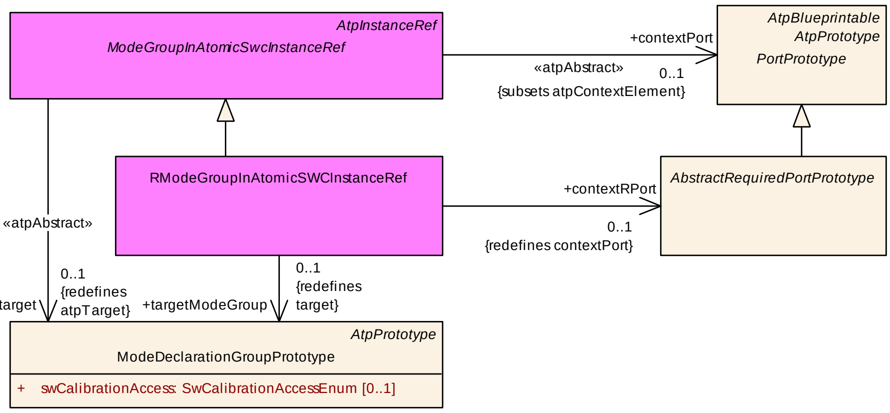


Figure D.11: Concrete modeling of references to ModeDeclarationGroupPrototype in the context of an RPortPrototype

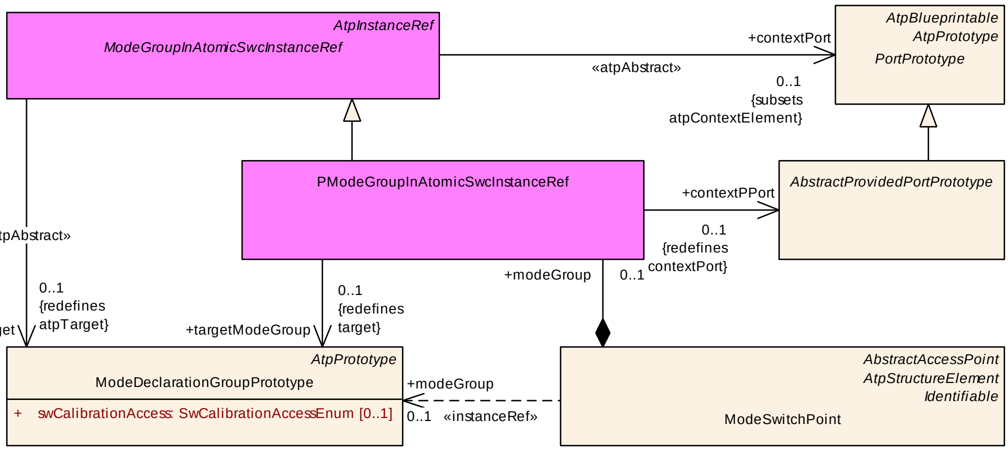


Figure D.12: Concrete modeling of references to ModeDeclarationGroupPrototype in the context of a PPortPrototype

Class	**PModeGroupInAtomicSwcInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *ModeGroupInAtomicSwcInstanceRef*			
Aggregated by	ModeAccessPoint.modeGroup, ModeSwitchPoint.modeGroup, SwcBswSynchronizedModeGroupPrototype.swcModeGroup, SwcModeManagerErrorEvent.modeGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
contextPPort	AbstractProvidedPortPrototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetModeGroup	ModeDeclarationGroupPrototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.12: PModeGroupInAtomicSwcInstanceRef

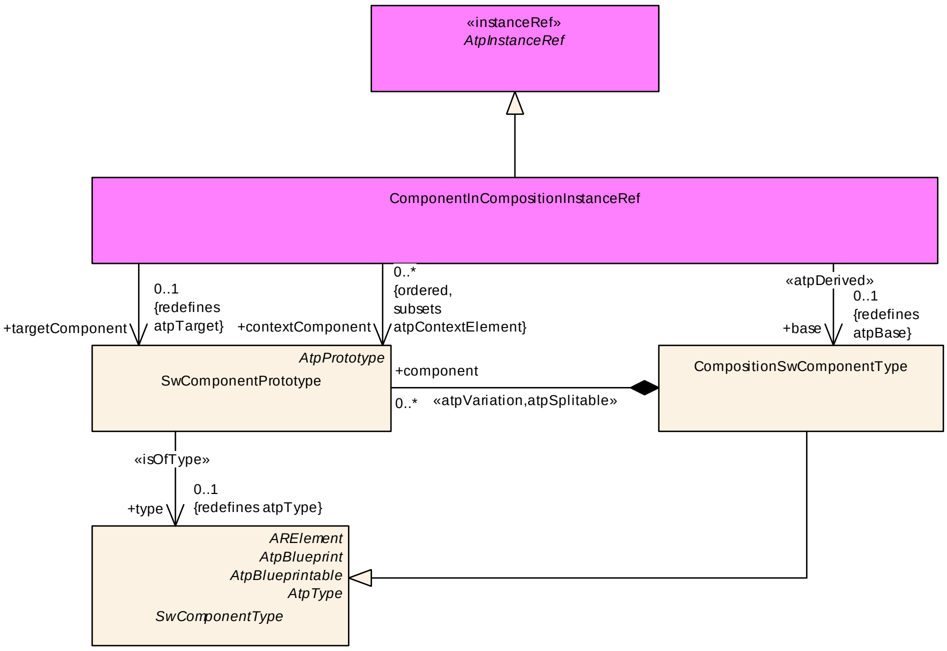


Figure D.13: Concrete modeling of references to `SwComponentPrototype` in the context of a `CompositionSwComponentType`

Class	**ComponentInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note	The ComponentInCompositionInstanceRef points to a concrete SwComponentPrototype within a CompositionSwComponentType.			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	DiagnosticJ1939SwMapping.swComponentPrototype, EOCEventRef.component, EOCExecutableEntityRef.component, ExecutionTimeConstraint.component, *TDEventSwc*.component, *TDEventVfb*.component			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextComponent (ordered)	SwComponentPrototype	*	ref	The context for the scope of this timing event. **Tags:**xml.sequenceOffset=20
targetComponent	SwComponentPrototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.13: ComponentInCompositionInstanceRef

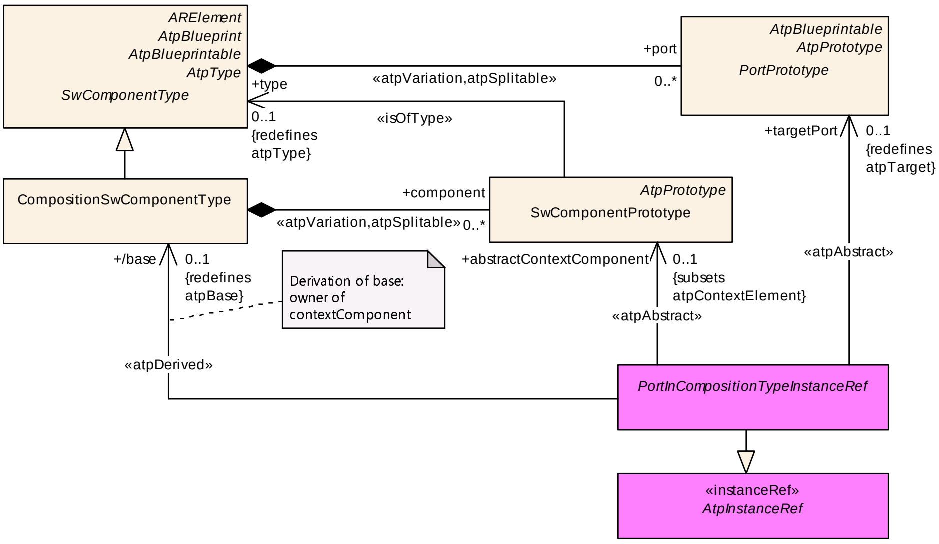


Figure D.14: Abstract modeling of references to PortPrototype in the context of a CompositionSwComponentType

Class	***PortInCompositionTypeInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Subclasses	PPortInCompositionInstanceRef, RPortInCompositionInstanceRef			
Aggregated by	DelegationSwConnector.innerPort			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
abstractContext Component	SwComponent Prototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=20
base	CompositionSw ComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
targetPort	PortPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=30

Table D.14: PortInCompositionTypeInstanceRef

Class	**PPortInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*, *PortInCompositionTypeInstanceRef*			
Aggregated by	AssemblySwConnector.provider, DelegationSwConnector.innerPort			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
context Component	SwComponent Prototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetPPort	AbstractProvidedPort Prototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.15: PPortInCompositionInstanceRef

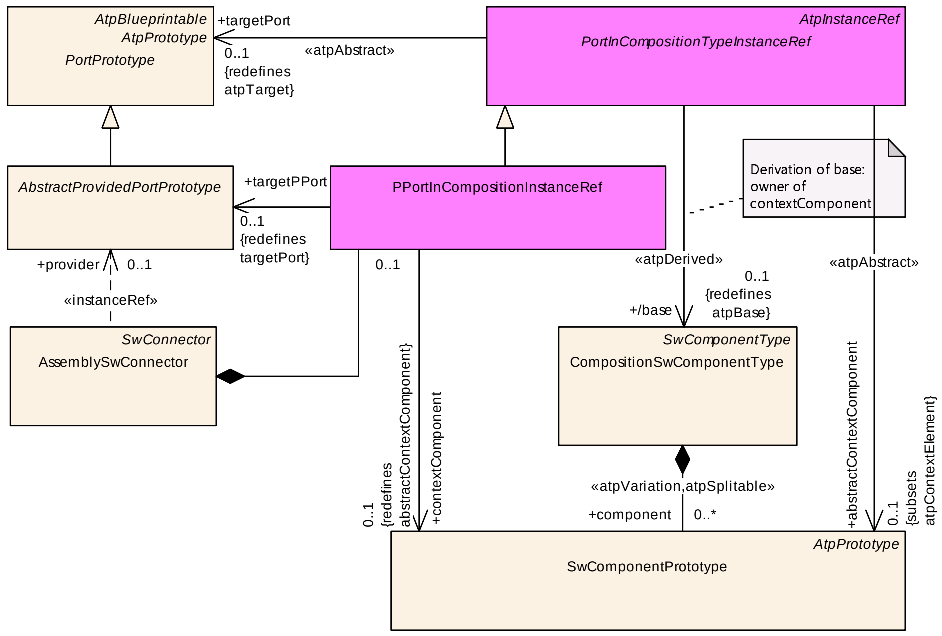


Figure D.15: Concrete modeling of references to PPortPrototype in the context of a CompositionSwComponentType

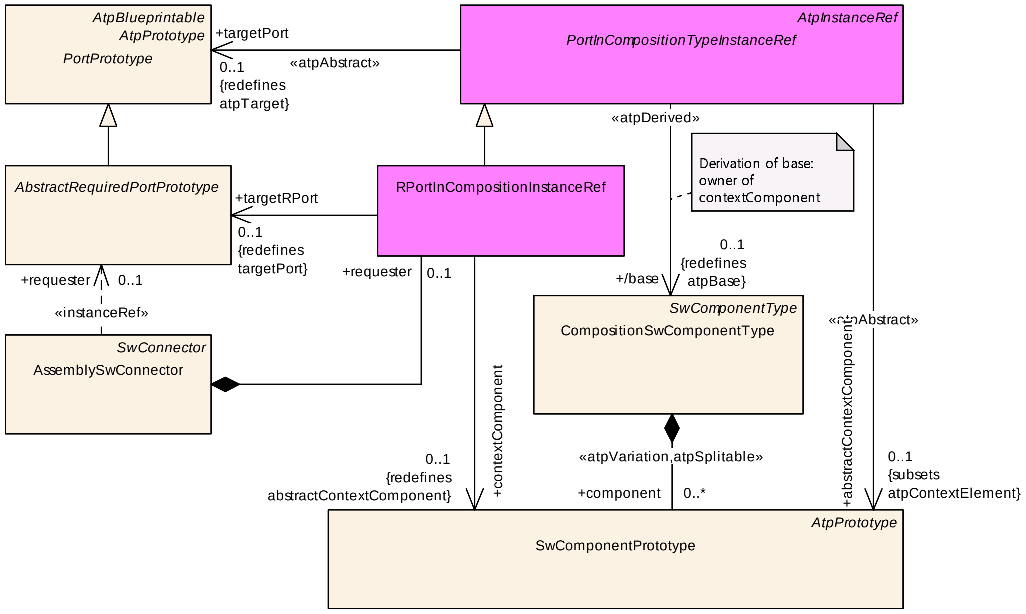


Figure D.16: Concrete modeling of references to RPortPrototype in the context of a CompositionSwComponentType

Class	**RPortInCompositionInstanceRef**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs
Note	
Base	*ARObject*, *AtpInstanceRef*, *PortInCompositionTypeInstanceRef*

▽

△

Class	**RPortInCompositionInstanceRef**			
Aggregated by	AssemblySwConnector.requester, DelegationSwConnector.innerPort, SecurityEventReportToSecurityEventDefinitionMapping.reportedSecurityEvent			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
contextComponent	SwComponentPrototype	0..1	ref	**Tags:**xml.sequenceOffset=20
targetRPort	AbstractRequiredPortPrototype	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.16: RPortInCompositionInstanceRef

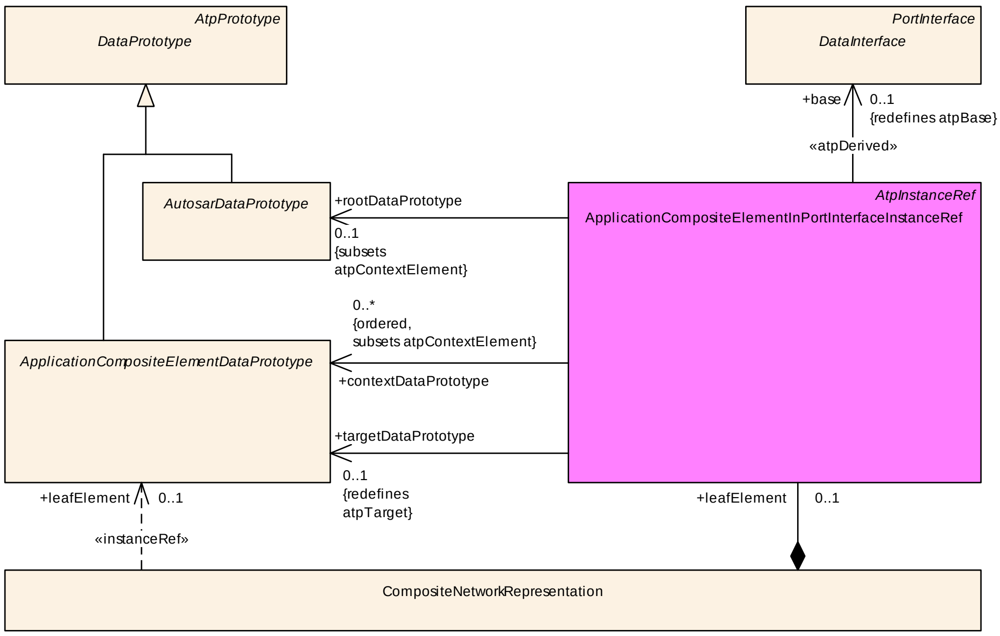


Figure D.17: Modeling of references to ApplicationCompositeElementDataPrototype for the purpose of defining a network representation

D.2.2 Definition of implicit Communication Behavior

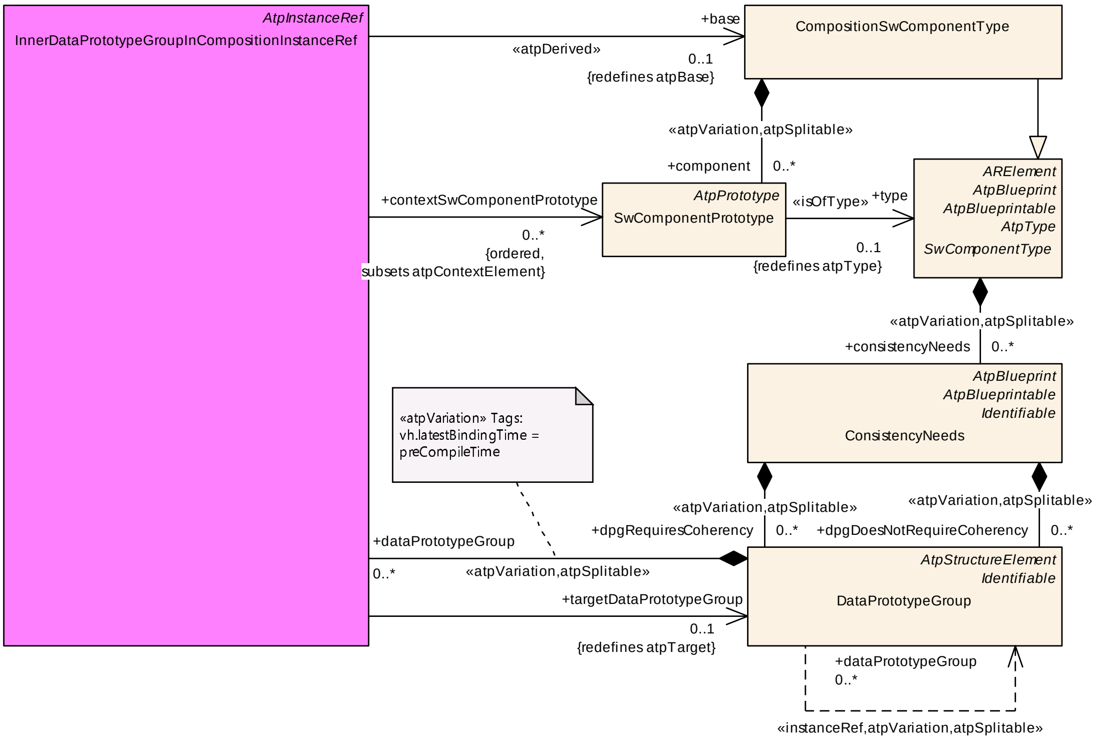


Figure D.18: Modeling of references to DataPrototypeGroup in the context of a CompositionSwComponentType

Class	**InnerDataPrototypeGroupInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior::InstanceRef			
Note	This meta-class represents the ability to define an InstanceRef to a nested DataPrototypeGroup			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	DataPrototypeGroup.dataPrototypeGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	This represents the base of the instanceRef. **Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextSwComponentPrototype (ordered)	SwComponentPrototype	*	ref	This represents the nested structure of SwComponentPrototypes. **Tags:**xml.sequenceOffset=20
targetDataPrototypeGroup	DataPrototypeGroup	0..1	ref	This represents the target of the InstanceRef **Tags:**xml.sequenceOffset=30

Table D.17: InnerDataPrototypeGroupInCompositionInstanceRef

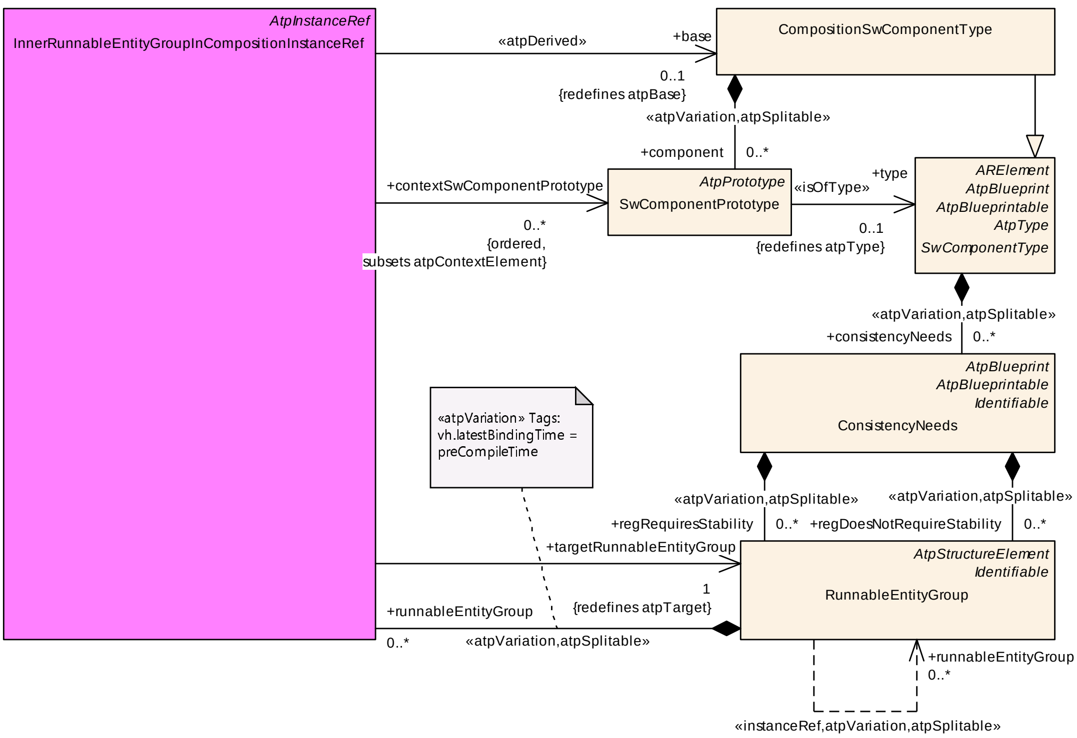


Figure D.19: Modeling of references to `RunnableEntityGroup` in the context of a `CompositionSwComponentType`

Class	**InnerRunnableEntityGroupInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior::InstanceRef			
Note	This meta-class represents the ability to define an InstanceRef to a nested RunnableEntityGroup.			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	RunnableEntityGroup.runnableEntityGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	This represents the base of the InstanceRef. **Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextSwComponentPrototype (ordered)	SwComponentPrototype	*	ref	This represents the nested structure of SwComponentPrototypes. **Tags:**xml.sequenceOffset=20
targetRunnableEntityGroup	RunnableEntityGroup	1	ref	This represents the target association of the InstanceRef. **Tags:**xml.sequenceOffset=30

Table D.18: InnerRunnableEntityGroupInCompositionInstanceRef

Class	**RunnableEntityInCompositionInstanceRef**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior::InstanceRef
Note	This meta-class represents the ability to define an InstanceRef to a RunnableEntity in the context of a CompositionSwComponentType.
Base	*ARObject*, *AtpInstanceRef*
Aggregated by	RunnableEntityGroup.runnableEntity

△

Class	**RunnableEntityInCompositionInstanceRef**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	This represents the base of the InstanceRef. **Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextSwComponentPrototype (ordered)	SwComponentPrototype	*	ref	This represents the nested structure of SwComponentPrototypes. **Tags:**xml.sequenceOffset=20
targetRunnableEntity	RunnableEntity	0..1	ref	This represents the target RunnableEntity. **Tags:**xml.sequenceOffset=30

Table D.19: RunnableEntityInCompositionInstanceRef

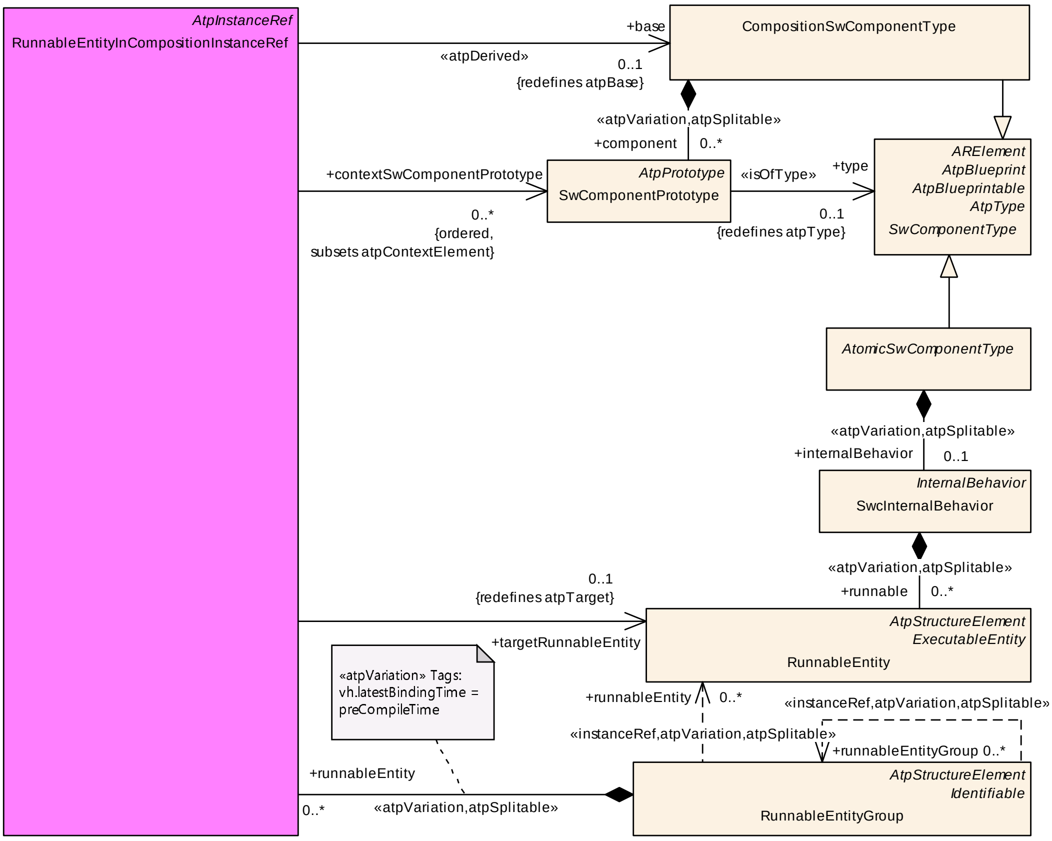


Figure D.20: Modeling of references to RunnableEntity in the context of a CompositionSwComponentType from the point of view of a RunnableEntityGroup

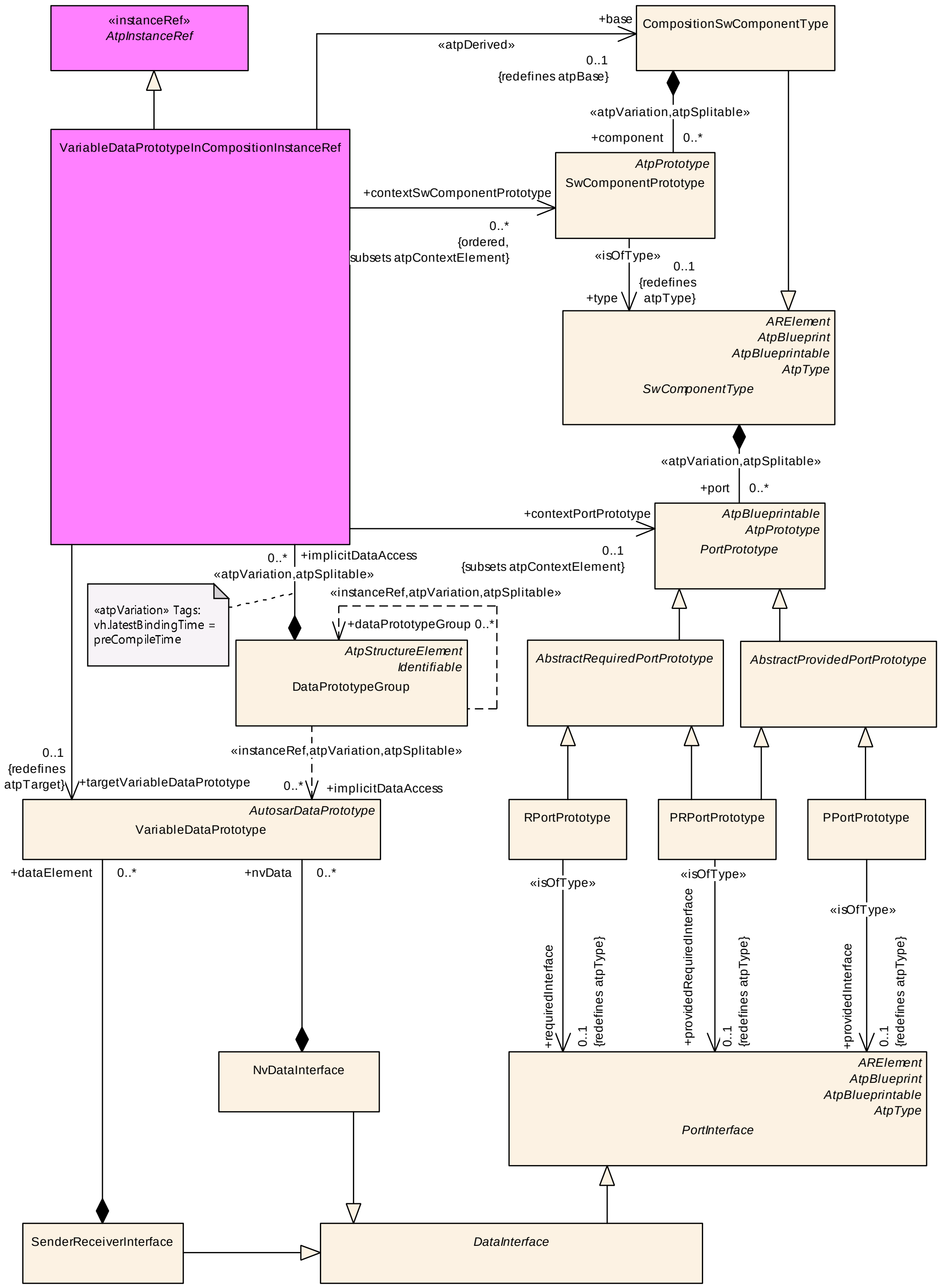


Figure D.21: Modeling of references to `VariableDataPrototype` in the context of a `CompositionSwComponentType` from the point of view of a `RunnableEntityGroup`

Class	**VariableDataPrototypeInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ImplicitCommunicationBehavior::InstanceRef			
Note	This meta-class represents the ability to define an InstanceRef to a VariableDataPrototype in the context of a CompositionSwComponentType.			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	DataPrototypeGroup.implicitDataAccess			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	This represents the base of the InstanceRef. **Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextPortPrototype	PortPrototype	0..1	ref	This represents a reference to a context PortPrototype. **Tags:**xml.sequenceOffset=30
contextSwComponentPrototype (ordered)	SwComponentPrototype	*	ref	This represents the nested structure of SwComponentPrototypes. **Tags:**xml.sequenceOffset=20
targetVariableDataPrototype	VariableDataPrototype	0..1	ref	This represents the target VariableDataPrototype. **Tags:**xml.sequenceOffset=40

Table D.20: VariableDataPrototypeInCompositionInstanceRef

Class	**InstanceEventInCompositionInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	*InstantiationRTEEventProps*.refinedEvent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	CompositionSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextComponentPrototype (ordered)	SwComponentPrototype	*	ref	**Tags:**xml.sequenceOffset=20
targetEvent	RTEEvent	0..1	ref	**Tags:**xml.sequenceOffset=30

Table D.21: InstanceEventInCompositionInstanceRef

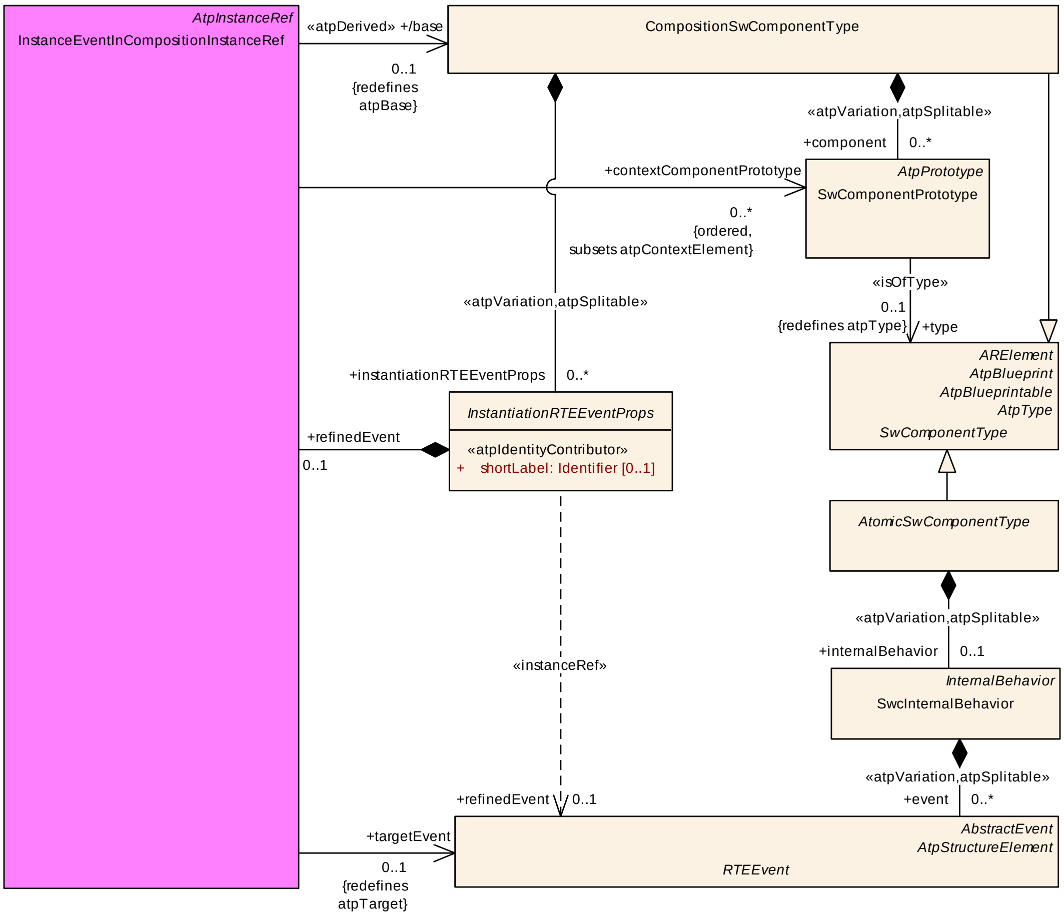


Figure D.22: Modeling of references to `RTEEvent` in the context of a `InstantiationRTEEventProps` from the point of view of a `CompositionSwComponentType`

D.2.3 Internal Behavior

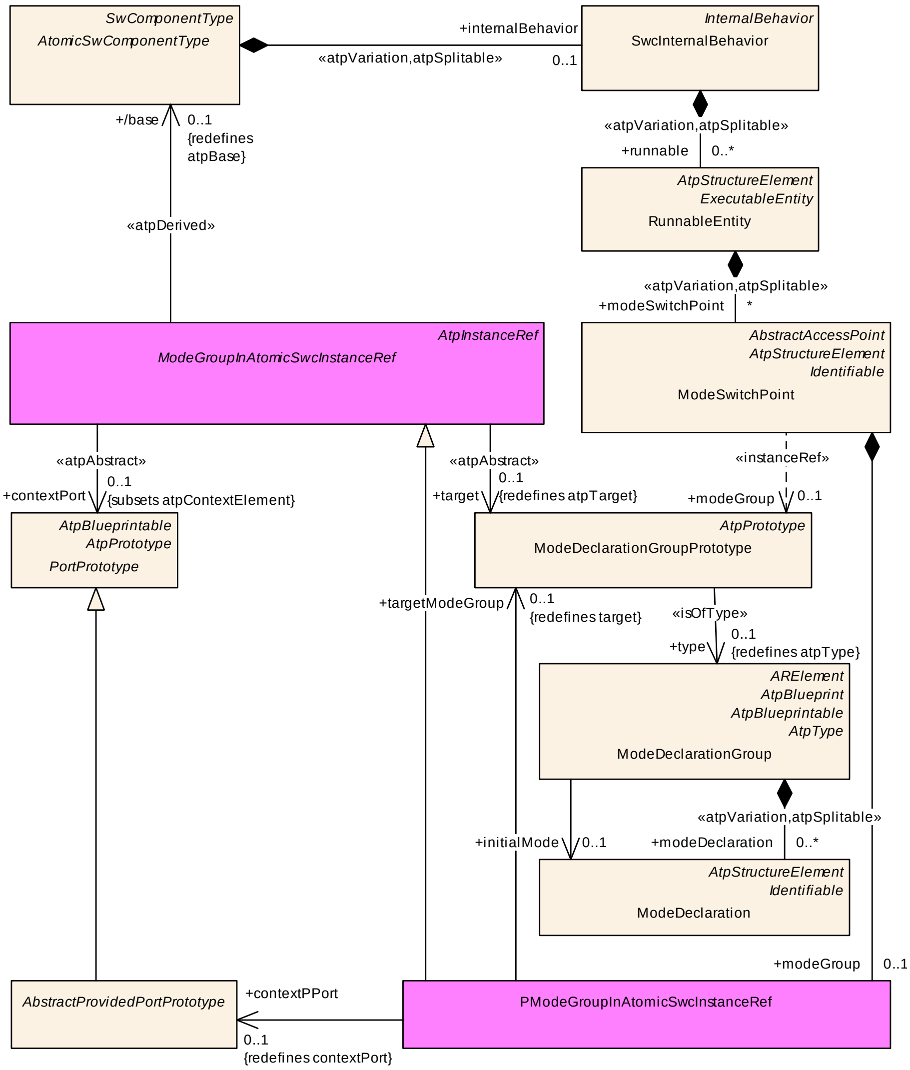


Figure D.23: Modeling of references to provided ModeDeclarationGroupPrototype in the context of an AtomicSwComponentType

Class	***ModeGroupInAtomicSwcInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Subclasses	PModeGroupInAtomicSwcInstanceRef, RModeGroupInAtomicSWCInstanceRef			
Aggregated by	ModeAccessPoint.modeGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtomicSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextPort	PortPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=20
target	ModeDeclarationGroupPrototype	0..1	ref	**Stereotypes:** atpAbstract **Tags:**xml.sequenceOffset=30

Table D.22: ModeGroupInAtomicSwcInstanceRef

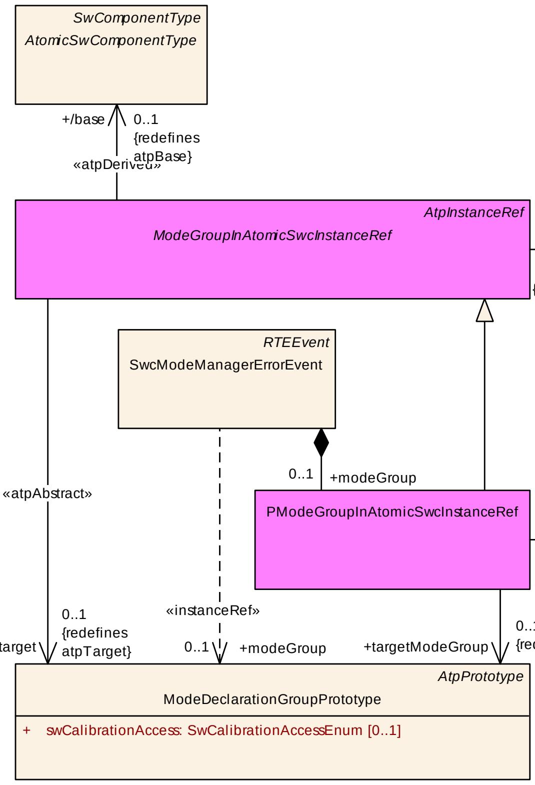


Figure D.24: Modeling of references to provided `ModeDeclarationGroupPrototype` to be used by `SwcModeManagerErrorEvent`

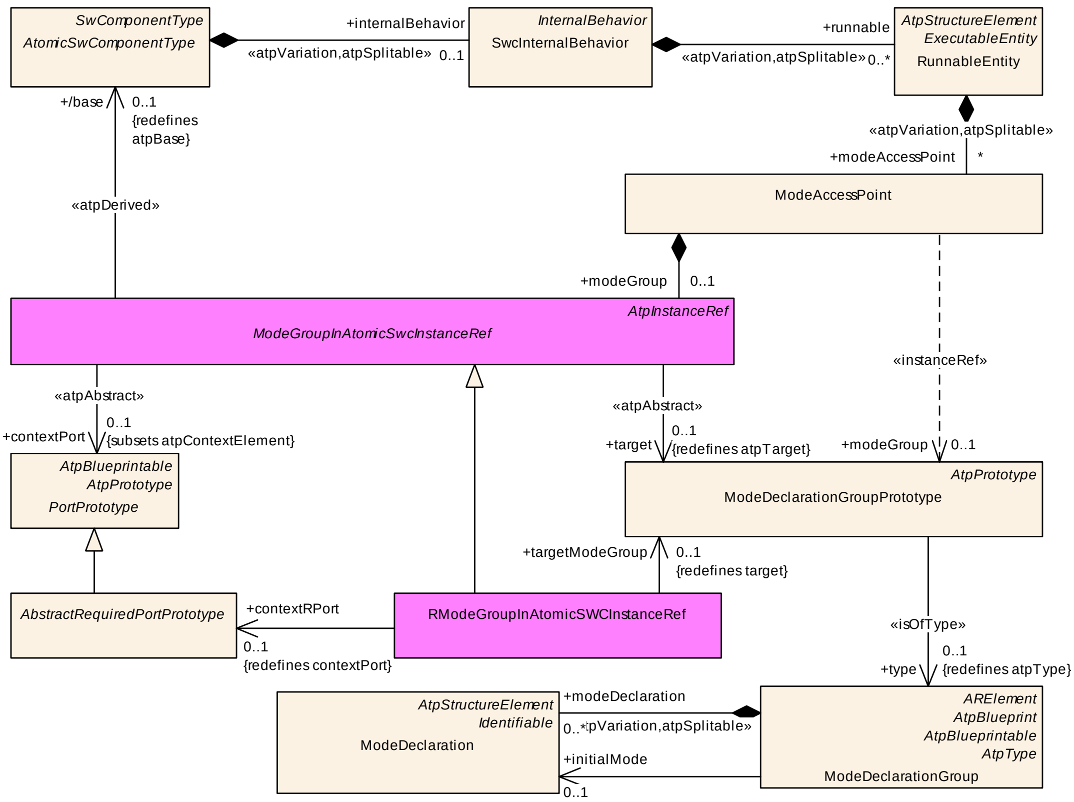


Figure D.25: Modeling of references to required `ModeDeclarationGroupPrototype` in the context of an `AtomicSwComponentType`

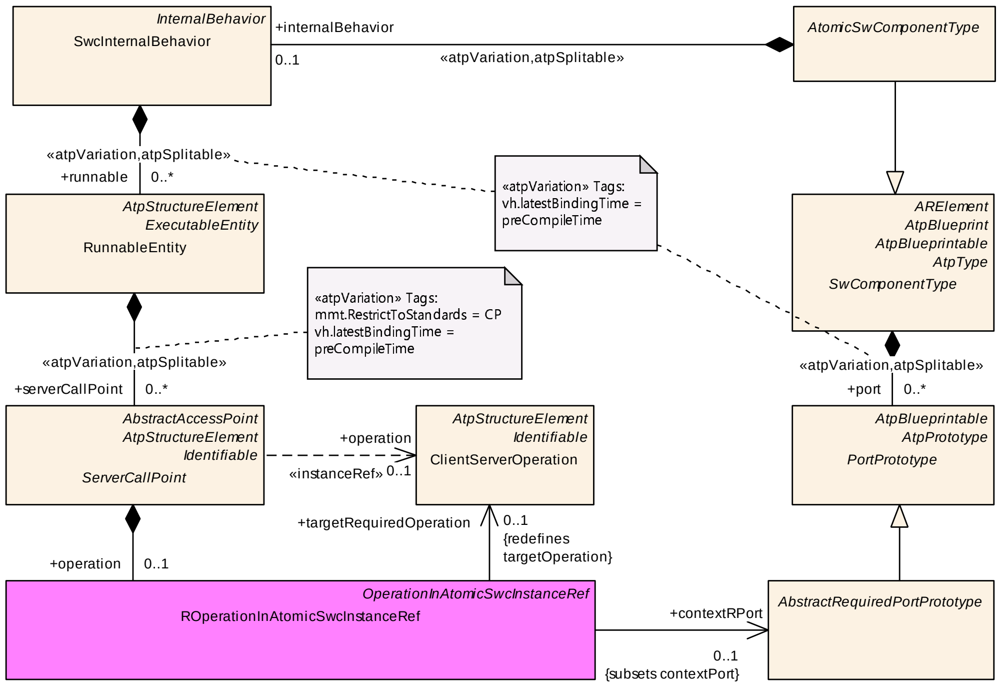


Figure D.26: Modeling of references to required `ClientServerOperation` in the context of a `SwComponentType`

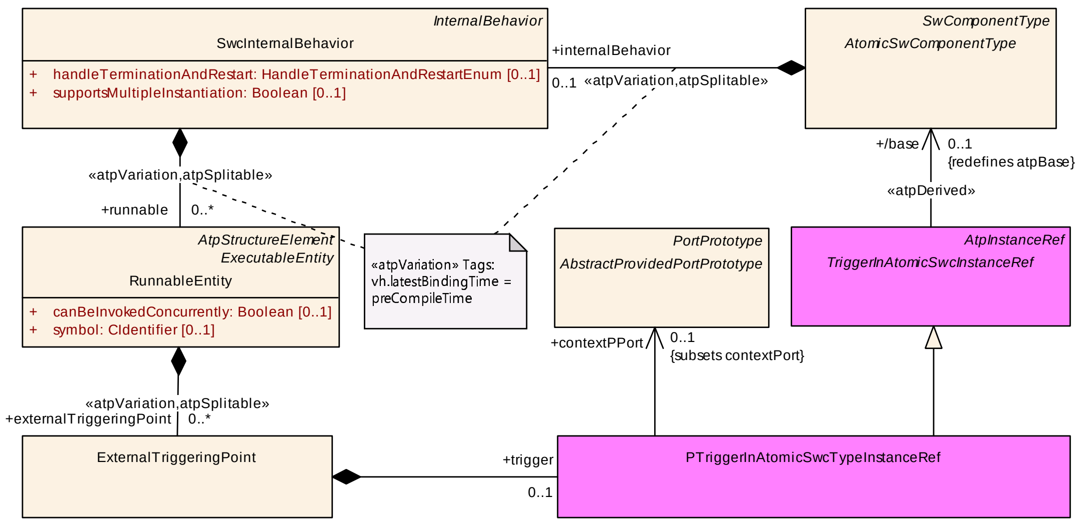


Figure D.27: Modeling of references to a `Trigger` in the context of a `SwComponentType`

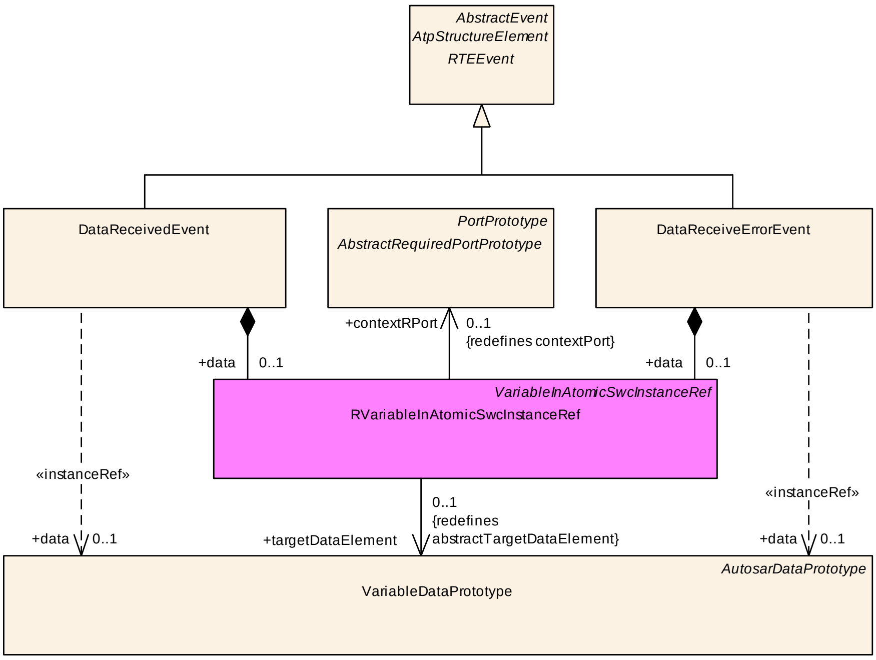


Figure D.28: Modeling of references to a VariableDataPrototype used in the context of an RTEEvent

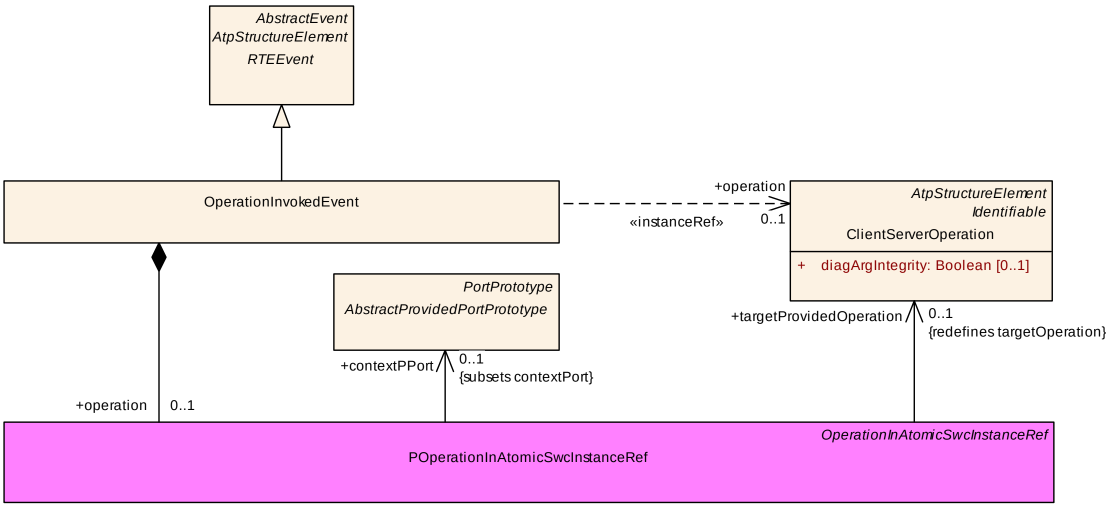


Figure D.29: Modeling of references to a ClientServerOperation used in the context of an RTEEvent

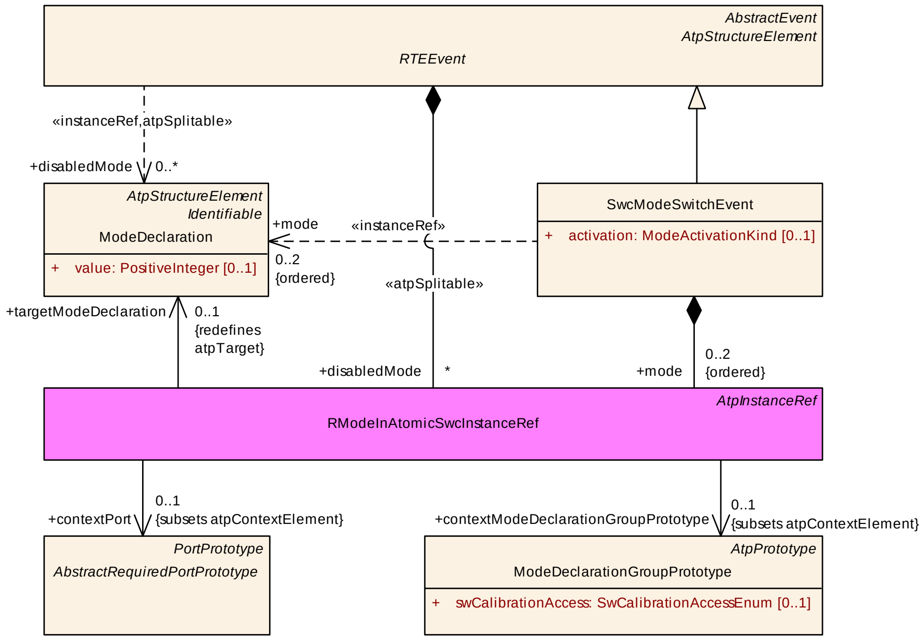


Figure D.30: Modeling of references to a ModeDeclaration used in the context of an RTEEvent

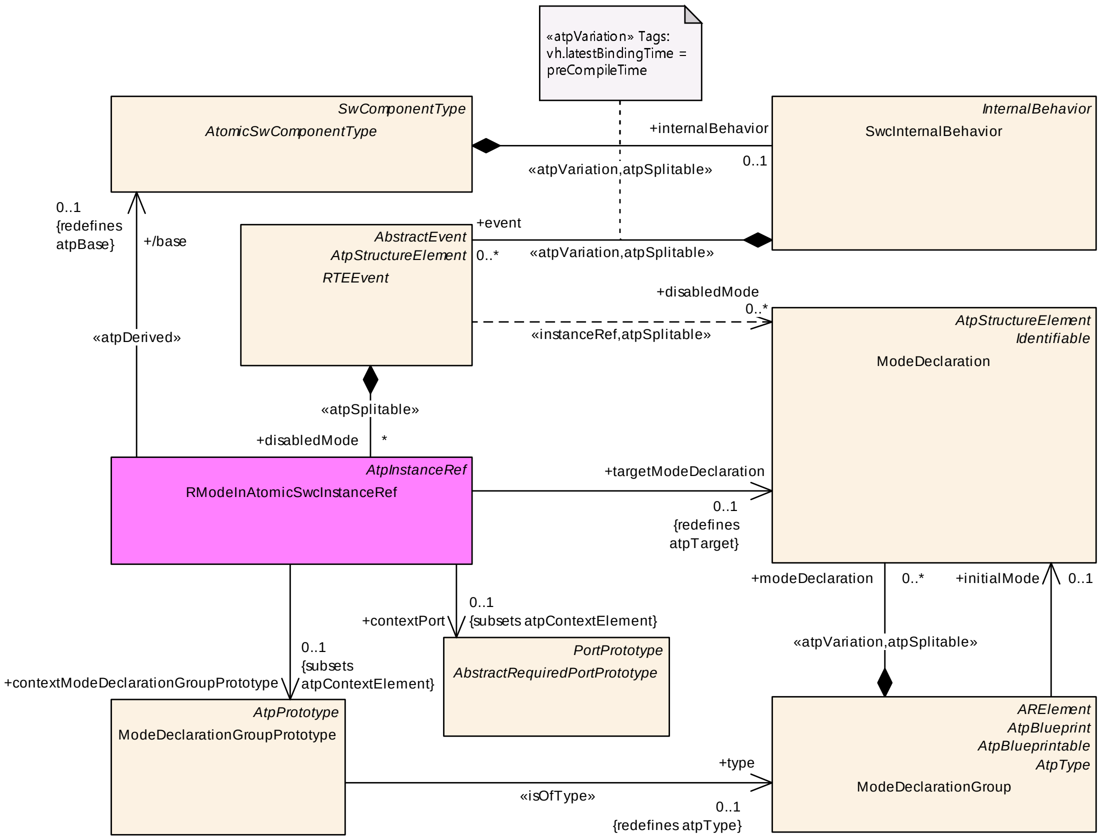


Figure D.31: Modeling of mode disabling

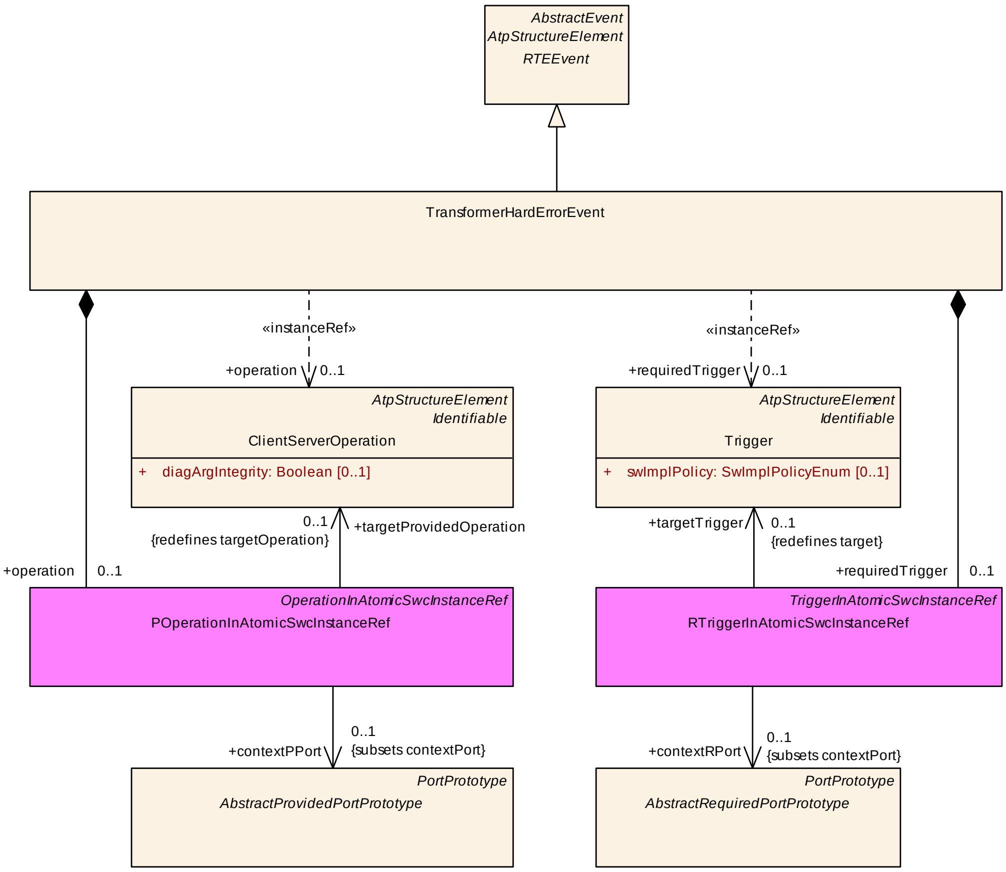


Figure D.32: Modeling of transformer error response

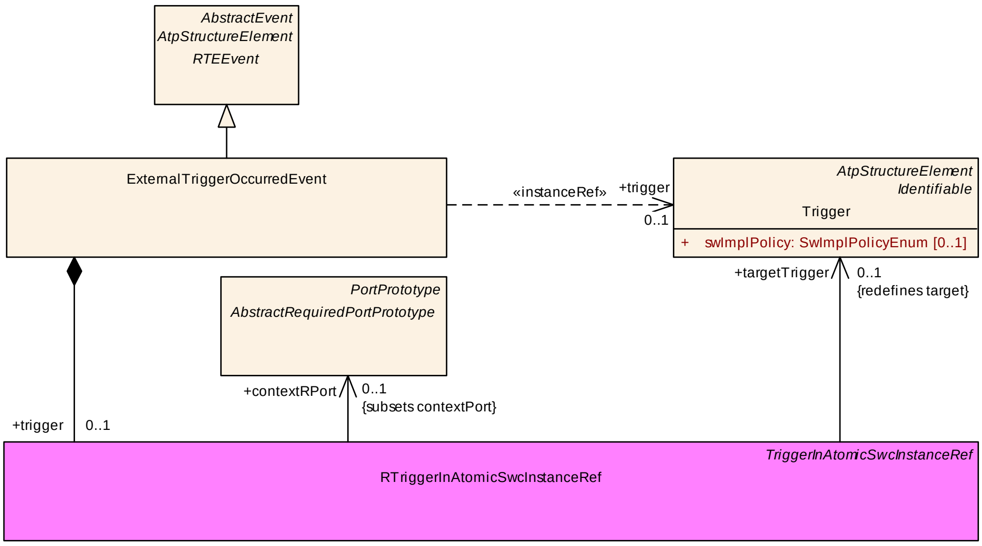


Figure D.33: Modeling of `RTriggerInAtomicSwcInstanceRef`

E Examples

E.1 Examples for the Definition of variable-size Arrays

This chapter contains some examples for the usage of variable-size arrays and its four defined use cases: Linear, square, rectangular and fully flexible.

All of these examples can be realized only using `ImplementationDataType`s or, as shown here, starting with the definition of `ApplicationDataType`s with corresponding `ImplementationDataType`s.

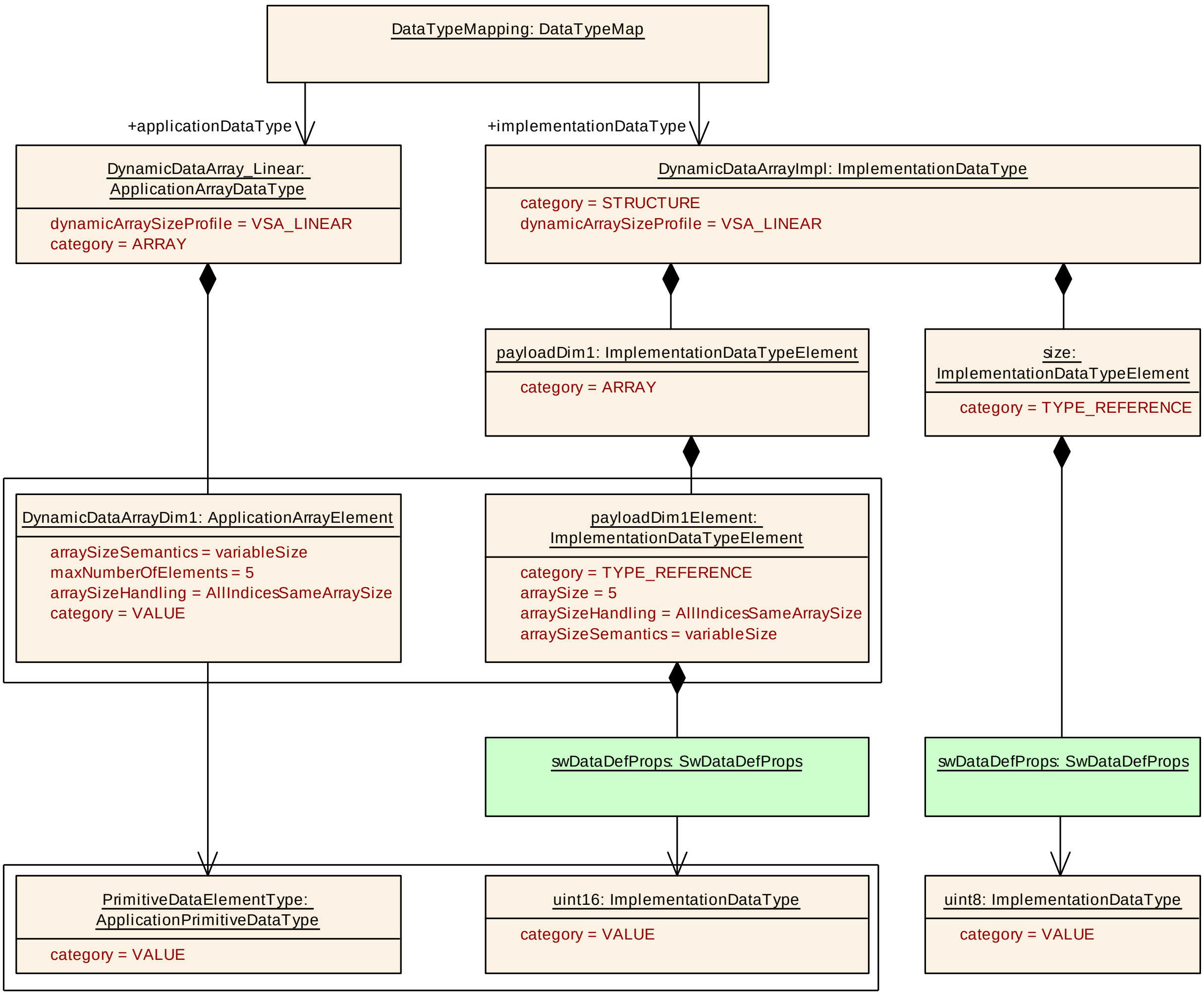


Figure E.1: Example of a linear variable size array

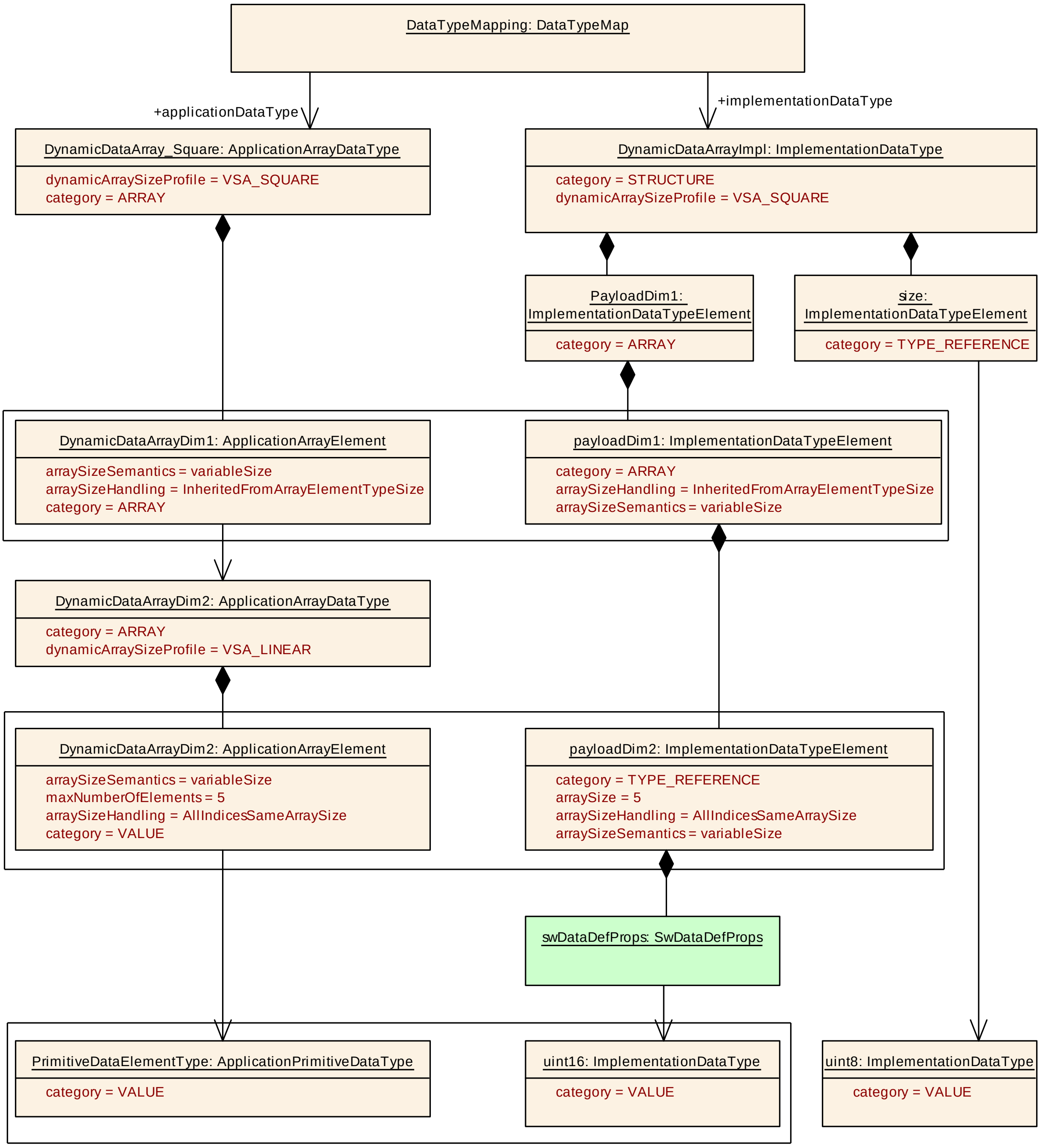


Figure E.2: Example of a square variable size array

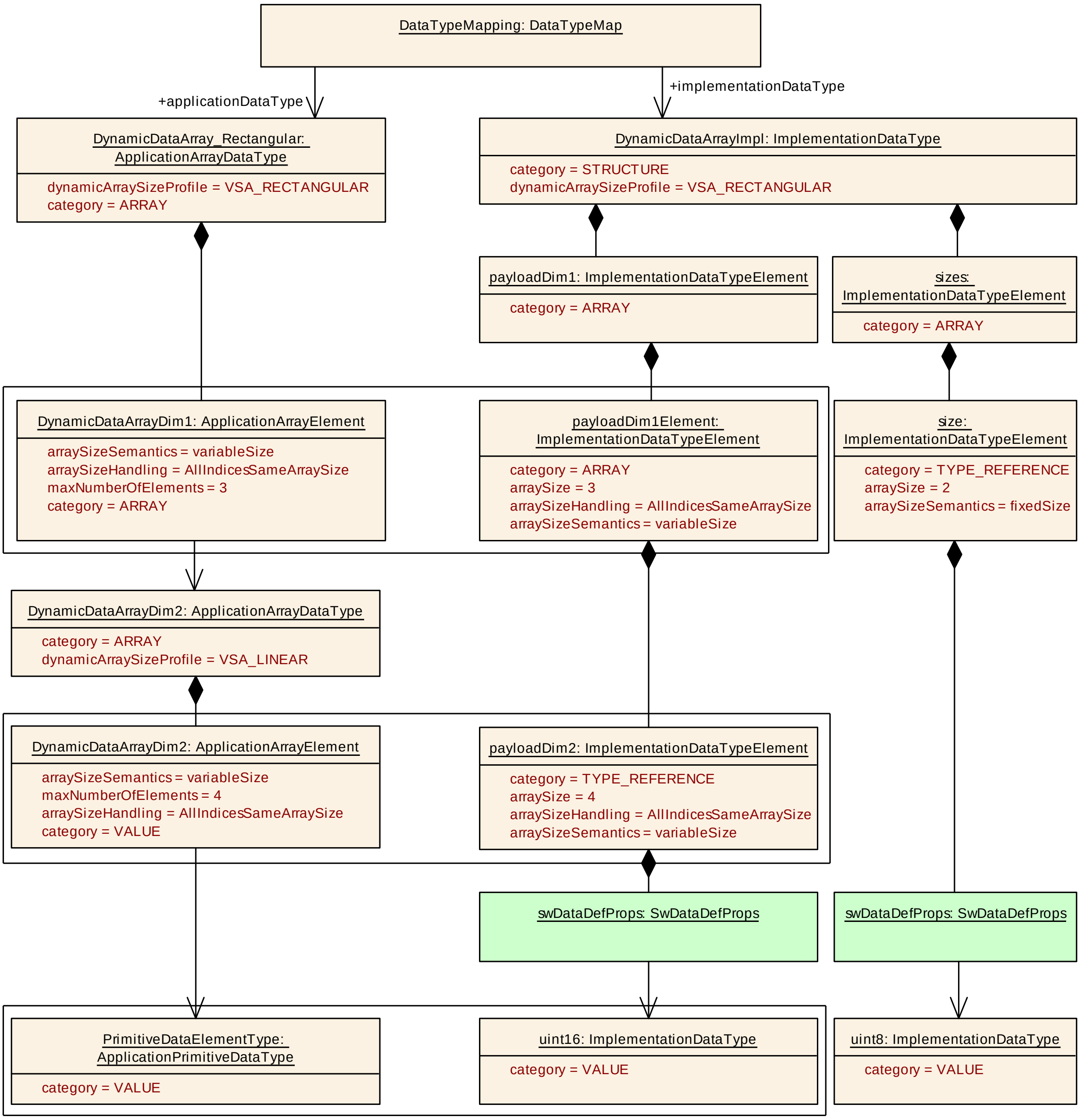


Figure E.3: Example of a rectangular variable size array

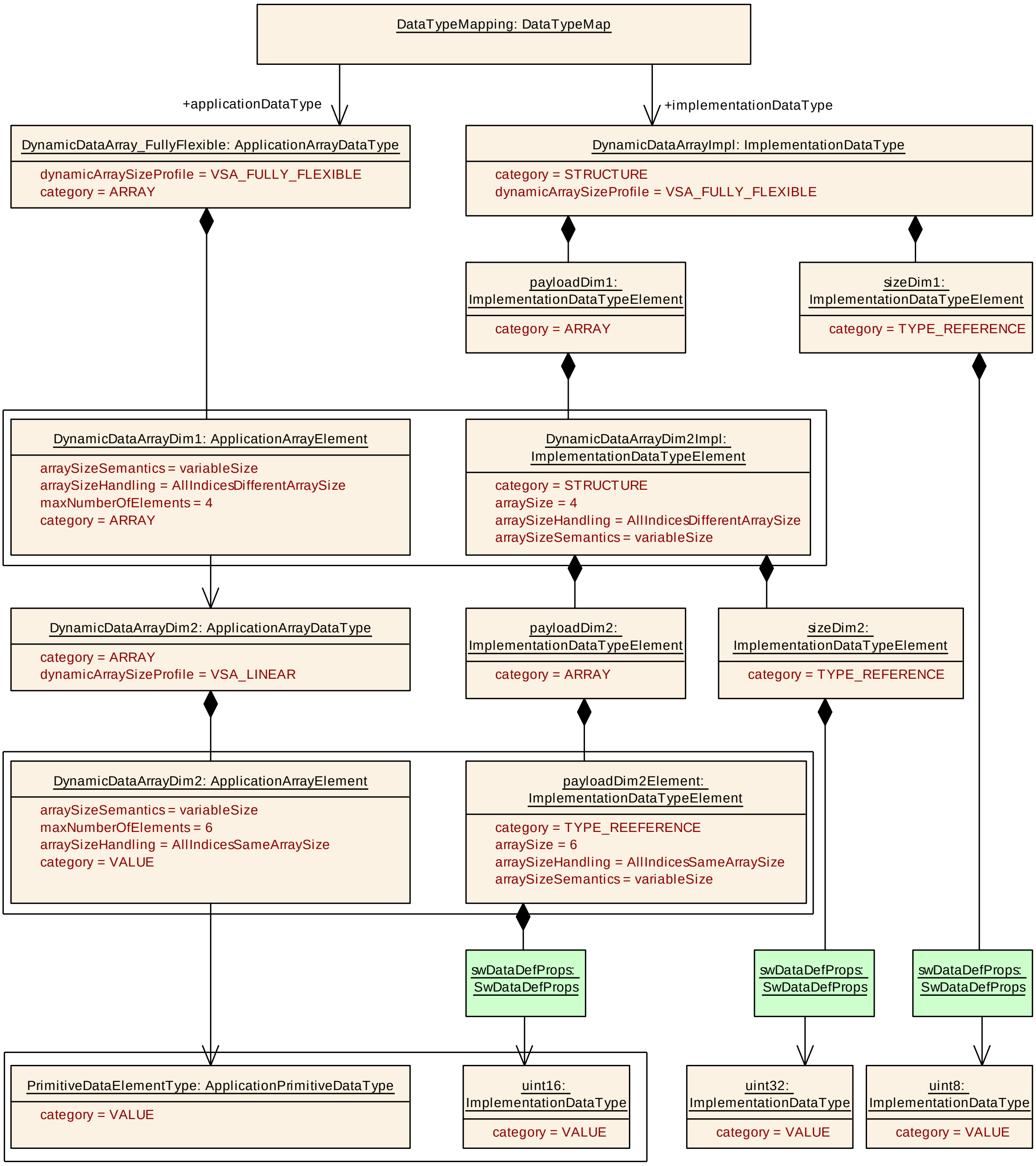


Figure E.4: Example of a fully flexible variable size array

F Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	***ARElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, ApplicationPartition, *AutosarDataType*, *BaseType*, BlueprintMappingSet, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, BuildActionManifest, CalibrationParameterValueSet, ClientIdDefinitionSet, ClientServerInterfaceToBswModuleEntryBlueprintMapping, Collection, CompuMethod, ConsistencyNeedsBlueprintSet, ConstantSpecification, ConstantSpecificationMappingSet, CpSoftwareCluster, CpSoftwareClusterBinaryManifestDescriptor, CpSoftwareClusterMappingSet, CpSoftwareClusterResourcePool, CryptoEllipticCurveProps, CryptoServiceCertificate, CryptoServiceKey, CryptoServicePrimitive, CryptoServiceQueue, CryptoSignatureScheme, DataConstr, DataExchangePoint, DataTransformationSet, DataTypeMappingSet, *DiagnosticCommonElement*, DiagnosticConnection, DiagnosticContributionSet, DltContext, DltEcu, Documentation, E2EProfileCompatibilityProps, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleConfigurationValues, EcucModuleDef, EcucValueCollection, EndToEndProtectionSet, EthIpProps, EthTcpIpIcmpProps, EthTcpIpProps, EvaluatedVariantSet, FMFeature, FMFeatureMap, FMFeatureModel, FMFeatureSelectionSet, FlatMap, GeneralPurposeConnection, HwCategory, HwElement, HwType, IPSecConfigProps, IPv6ExtHeaderFilterSet, *IdsCommonElement*, IdsDesign, *Implementation*, InterpolationRoutineMappingSet, J1939ControllerApplication, KeywordSet, LifeCycleInfoSet, LifeCycleStateDefinitionGroup, LogAndTraceMessageCollectionSet, MacSecGlobalKayProps, MacSecParticipantSet, McFunction, McGroup, ModeDeclarationGroup, ModeDeclarationMappingSet, OsTaskProxy, PhysicalDimension, PhysicalDimensionMappingSet, *PortInterface*, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuildVariantCriterionValueSet, PredefinedVariant, RapidPrototypingScenario, SdgDef, SignalServiceTranslationPropsSet, SomeipSdClientEventGroupTimingConfig, SomeipSdClientServiceInstanceConfig, SomeipSdServerEventGroupTimingConfig, SomeipSdServerServiceInstanceConfig, SwAddrMethod, SwAxisType, SwComponentMappingConstraints, *SwComponentType*, SwRecordLayout, SwSystemconst, SwSystemconstantValueSet, SwcBswMapping, System, SystemSignal, SystemSignalGroup, TDCpSoftwareClusterMappingSet, TcpOptionFilterSet, *TimingExtension*, TlsConnectionGroup, TlvDataIdDefinitionSet, TransformationPropsSet, Unit, UnitGroup, ViewMapSet			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table F.1: ARElement

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ARPackage.arPackage, AUTOSAR.arPackage			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	ARPackage			
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element.shortName, element.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=referenceBase.shortLabel xml.sequenceOffset=10

Table F.2: ARPackage

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:**xml.globalElement=true			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:**xml.sequenceOffset=10
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arPackage.shortName, arPackage.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
fileInfoComment	FileInfoComment	0..1	aggr	This represents a possibility to provide a structured comment in an AUTOSAR file. **Stereotypes:** atpStructuredComment **Tags:** xml.roleElement=true xml.sequenceOffset=-10 xml.typeElement=false
introduction	DocumentationBlock	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to represent disclaimers and legal notes. **Tags:**xml.sequenceOffset=20

Table F.3: AUTOSAR

Class	<<atpMixedString>> ***AbstractNumericalVariationPoint*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariationPoints			
Note	This is an abstract NumericalValueVariationPoint. It is introduced to support the case that additional attributes are required for particular purposes.			
Base	*ARObject*, *AttributeValueVariationPoint*, *FormulaExpression*, *SwSystemconstDependentFormula*			
Subclasses	LimitValueVariationPoint, NumericalValueVariationPoint			
Aggregated by	VariationPointProxy.valueAccess			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table F.4: AbstractNumericalVariationPoint

Class	**AdminData**			
Package	M2::MSR::AsamHdo::AdminData			
Note	AdminData represents the ability to express administrative information and custom extensions for an element. This administration information is to be treated as meta-data such as revision id or state of the file. There are basically the following kinds of meta-data • The language and/or used languages. • Revision information covering e.g. revision number, state, release date, changes. Note that this information can be given in general as well as related to a particular company. • Document meta-data specific for a company Beside that a custom extension of model-data is possible by • Special data			
Base	*ARObject*			
Aggregated by	AUTOSAR.adminData, *Describable*.adminData, *Identifiable*.adminData			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
docRevision (ordered)	DocRevision	*	aggr	This allows to denote information about the current revision of the object. Note that information about previous revisions can also be logged here. The entries shall be sorted descendant by date in order to reflect the history. Therefore the most recent entry representing the current version is denoted first. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=50 xml.typeElement=false xml.typeWrapperElement=false
language	LEnum	0..1	attr	This attribute specifies the master language of the document or the document fragment. The master language is the one in which the document is maintained and from which the other languages are derived from. In particular in case of inconsistencies, the information in the master language is priority. **Tags:**xml.sequenceOffset=20

▽

△

Class	AdminData			
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=sdg xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=60 xml.typeElement=false xml.typeWrapperElement=false
usedLanguages	MultiLanguagePlainText	0..1	aggr	This property specifies the languages which are provided in the document. Therefore it should only be specified in the top level admin data. For each language provided in the document there is one entry in MultilanguagePlain Text. The content of each entry can be used for illustration of the language. The used language itself depends on the language attribute in the entry. **Tags:**xml.sequenceOffset=30

Table F.5: AdminData

Class	**AnyInstanceRef**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AnyInstanceRef			
Note	Describes a reference to any instance in an AUTOSAR model. This is the most generic form of an instance ref. Refer to the superclass notes for more details.			
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	Collection.collectedInstance, Collection.sourceInstance, DocumentationContext.feature, EcucInstance ReferenceValue.value, FlatInstanceDescriptor.ecuExtractReference, FlatInstanceDescriptor.upstream Reference, RptContainer.byPassPoint, RptHook.rptArHook, ViewMap.firstElementInstance, ViewMap. secondElementInstance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	AtpClassifier	1	ref	This is the base from which navigation path begins. **Stereotypes:** atpDerived
contextElement (ordered)	AtpFeature	*	ref	This is one step in the navigation path specified by the instance ref.
target	AtpFeature	1	ref	This is the target of the instance ref.

Table F.6: AnyInstanceRef

Class	**ApplicationCompositeElementInPortInterfaceInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface::InstanceRefs			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	ApplicationCompositeDataTypeSubElementRef.applicationCompositeElement, CompositeNetwork Representation.leafElement			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
base	DataInterface	0..1	ref	This represents the SenderReceiverInterface that acts as the base in this InstanceRef definition **Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10

▽

△

Class	ApplicationCompositeElementInPortInterfaceInstanceRef			
contextData Prototype (ordered)	ApplicationComposite ElementDataPrototype	*	ref	This represents a context ApplicationCompositeData Prototype **Tags:**xml.sequenceOffset=20
rootData Prototype	AutosarDataPrototype	0..1	ref	This refers to the dataPrototype which is typed by the ApplicationDatatype in which which the target can be found. **Tags:**xml.sequenceOffset=15
targetData Prototype	ApplicationComposite ElementDataPrototype	0..1	ref	This represents the referenced ApplicationComposite DataPrototype. **Tags:**xml.sequenceOffset=30

Table F.7: ApplicationCompositeElementInPortInterfaceInstanceRef

Class	***AtpInstanceRef*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::AbstractStructure			
Note	An M0 instance of a classifier may be represented as a tree rooted at that instance, where under each node come the sub-trees representing the instances which act as features under that node. An instance ref specifies a navigation path from any M0 tree-instance of the base (which is a classifier) to a leaf (which is an instance of the target).			
Base	*ARObject*			
Subclasses	AnyInstanceRef, ApplicationCompositeElementInPortInterfaceInstanceRef, ComponentInComposition InstanceRef, ComponentInSystemInstanceRef, *DataPrototypeInPortInterfaceInstanceRef*, Data PrototypeInSystemInstanceRef, InnerDataPrototypeGroupInCompositionInstanceRef, InnerPortGroupIn CompositionInstanceRef, InnerRunnableEntityGroupInCompositionInstanceRef, InstanceEventIn CompositionInstanceRef, ModeDeclarationGroupPrototypeInSystemInstanceRef, *ModeGroupInAtomic SwcInstanceRef*, ModeInBswModuleDescriptionInstanceRef, ModeInSwcInstanceRef, Operation ArgumentInComponentInstanceRef, *OperationInAtomicSwcInstanceRef*, OperationInSystemInstance Ref, PModeInSystemInstanceRef, ParameterDataPrototypeInSystemInstanceRef, ParameterInAtomicS WCTypeInstanceRef, PortGroupInSystemInstanceRef, *PortInCompositionTypeInstanceRef*, RModeIn AtomicSwcInstanceRef, RteEventInCompositionInstanceRef, RteEventInEcuInstanceRef, RteEventIn SystemInstanceRef, RunnableEntityInCompositionInstanceRef, SwcServiceDependencyInSystem InstanceRef, *TriggerInAtomicSwcInstanceRef*, TriggerInSystemInstanceRef, VariableAccessInEcu InstanceRef, VariableDataPrototypeInCompositionInstanceRef, VariableDataPrototypeInSystemInstance Ref, VariableInAtomicSWCTypeInstanceRef, *VariableInAtomicSwcInstanceRef*, VariableInComponent InstanceRef			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
atpBase	AtpClassifier	1	ref	This is the base from which the navigaion path starts. **Stereotypes:** atpAbstract; atpDerived
atpContext Element (ordered)	AtpPrototype	*	ref	This is one particular step in the navigation path. **Stereotypes:** atpAbstract
atpTarget	AtpFeature	1	ref	This is the target of the instance ref. In other words it is the terminal of the navigation path. **Stereotypes:** atpAbstract

Table F.8: AtpInstanceRef

Enumeration	BindingTimeEnum
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumerator specifies the applicable binding times for the pre build variation points.
Aggregated by	*AttributeValueVariationPoint*.bindingTime, ConditionByFormula.bindingTime, FMFeature.maximumIntendedBindingTime, FMFeature.minimumIntendedBindingTime, FMFeatureSelection.maximumSelectedBindingTime, FMFeatureSelection.minimumSelectedBindingTime
Literal	***Description***
codeGenerationTime	• Coding by hand, based on requirements document. • Tool based code generation, e.g. from a model. • The model may contain variants. • Only code for the selected variant(s) is actually generated. **Tags:**atp.EnumerationLiteralIndex=0
linkTime	Configure what is included in object code, and what is omitted Based on which variant(s) are selected E.g. for modules that are delivered as object code (as opposed to those that are delivered as source code) **Tags:**atp.EnumerationLiteralIndex=1
preCompileTime	This is typically the C-Preprocessor. Exclude parts of the code from the compilation process, e.g., because they are not required for the selected variant, because they are incompatible with the selected variant, because they require resources that are not present in the selected variant. Object code is only generated for the selected variant(s). The code that is excluded at this stage code will not be available at later stages. **Tags:**atp.EnumerationLiteralIndex=2
systemDesignTime	• Designing the VFB. • Software Component types (PortInterfaces). • SWC Prototypes and the Connections between SWCprototypes. • Designing the Topology • ECUs and interconnecting Networks • Designing the Communication Matrix and Data Mapping **Tags:**atp.EnumerationLiteralIndex=3

Table F.9: BindingTimeEnum

Class	BswCalledEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity which is designed to be called from another BSW module or cluster.			
Base	*ARObject*, *BswModuleEntity*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	BswInternalBehavior.entity			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table F.10: BswCalledEntity

Class	BswImplementation
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation
Note	Contains the implementation specific information in addition to the generic specification (BswModuleDescription and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:**atp.recommendedPackage=BswImplementations

▽

△

Class	**BswImplementation**			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arRelease Version	RevisionLabelString	0..1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBehavior	0..1	ref	The behavior of this implementation. This relation is made as an association because • it follows the pattern of the SWCT • since ARElement cannot be splitted, but we want supply the implementation later, the Bsw Implementation is not aggregated in BswBehavior
preconfigured Configuration	EcucModule ConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:**xml.roleWrapperElement=true
recommended Configuration	EcucModule ConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiInfix	Identifier	0..1	attr	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <Module Name>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecific ModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this Bsw Implementation if it represents a cluster of modules • one or no EcucModuleDefs used in this Bsw Implementation if it represents a library **Tags:**xml.roleWrapperElement=true

Table F.11: BswImplementation

Class	**BswModuleDescription**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswOverview			
Note	Root element for the description of a single BSW module or BSW cluster. In case it describes a BSW module, the short name of this element equals the name of the BSW module. **Tags:**atp.recommendedPackage=BswModuleDescriptions			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bswModule Dependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDependency.shortName, bsw ModuleDependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModule Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, bswModule Documentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6
expectedEntry	BswModuleEntry	*	ref	Indicates an entry which is required by this module. Replacement of outgoingCallback / requiredEntry. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=expectedEntry.bswModuleEntry, expected Entry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
implemented Entry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions, interrupt routines, and callbacks. Replacement of providedEntry / expectedCallback. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implementedEntry.bswModuleEntry, implementedEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a Bsw ModuleDescription can be distributed over several physical files. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=internalBehavior.shortName xml.sequenceOffset=65
moduleId	PositiveInteger	0..1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be optionally chosen. **Tags:**xml.sequenceOffset=5

▽

△

Class	**BswModuleDescription**			
providedClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module provides a client server entry which can be called from another partition or core.This entry is declared locally to this context and will be connected to the requiredClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedClientServerEntry.shortName, providedClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
providedData	VariableDataPrototype	*	aggr	Specifies a data prototype provided by this module in order to be read from another partition or core.The providedData is declared locally to this context and will be connected to the requiredData of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedData.shortName, provided Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
providedMode Group	ModeDeclarationGroup Prototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the required ModeGroups of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with modes provided via ports by an associated ServiceSwComponentType, EcuAbstraction SwComponentType or ComplexDeviceDriverSw ComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedModeGroup.shortName, provided ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
releasedTrigger	Trigger	*	aggr	A Trigger released by this module or cluster. It can be connected to the requiredTriggers of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with Triggers provided via ports by an associated ServiceSwComponentType, Ecu AbstractionSwComponentType or ComplexDeviceDriver SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=releasedTrigger.shortName, released Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=35

▽

△

Class	BswModuleDescription			
requiredClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another partition or core.This entry is declared locally to this context and will be connected to the providedClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredClientServerEntry.shortName, requiredClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
requiredData	VariableDataPrototype	*	aggr	Specifies a data prototype required by this module in oder to be provided from another partition or core.The required Data is declared locally to this context and will be connected to the providedData of another or the same module via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredData.shortName, required Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
requiredMode Group	ModeDeclarationGroup Prototype	*	aggr	Specifies that this module or cluster depends on a certain mode group. The requiredModeGroup is local to this context and will be connected to the providedModeGroup of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredModeGroup.shortName, required ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
requiredTrigger	Trigger	*	aggr	Specifies that this module or cluster reacts upon an external trigger.This requiredTrigger is declared locally to this context and will be connected to the providedTrigger of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredTrigger.shortName, required Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table F.12: BswModuleDescription

Class	BswModuleEntry
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	This class represents a single API entry (C-function prototype) into the BSW module or cluster. The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation. **Tags:**atp.recommendedPackage=BswModuleEntrys

▽

△

Class	BswModuleEntry			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=argument.shortName, argument.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45
bswEntryKind	BswEntryKindEnum	0..1	attr	This describes whether the entry is concrete or abstract. If the attribute is missing the entry is considered as concrete. **Tags:**xml.sequenceOffset=40
callType	BswCallType	0..1	attr	The type of call associated with this service. **Tags:**xml.sequenceOffset=25
execution Context	BswExecutionContext	0..1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:**xml.sequenceOffset=30
function Prototype Emitter	NameToken	0..1	attr	This attribute is used to control the generation of function prototypes. If set to "RTE", the RTE generates the function prototypes in the Module Interlink Header File.
isReentrant	Boolean	0..1	attr	Reentrancy from the viewpoint of function callers: • true: Enables the service to be invoked again, before the service has finished. • false: It is prohibited to invoke the service again before is has finished. **Tags:**xml.sequenceOffset=15
isSynchronous	Boolean	0..1	attr	Synchronicity from the viewpoint of function callers: • true: This calls a synchronous service, i.e. the service is completed when the call returns. • false: The service (on semantical level) may not be complete when the call returns. **Tags:**xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:**xml.sequenceOffset=40
role	Identifier	0..1	attr	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:**xml.sequenceOffset=10
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:**xml.sequenceOffset=5

▽

△

Class	BswModuleEntry			
swServiceImpl Policy	SwServiceImplPolicy Enum	0..1	attr	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. **Tags:**xml.sequenceOffset=35

Table F.13: BswModuleEntry

Class	**BswSchedulableEntity**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity, which is designed for control by the BSW Scheduler. It may for example implement a so-called "main" function.			
Base	*ARObject*, *BswModuleEntity*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	BswInternalBehavior.entity			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table F.14: BswSchedulableEntity

Class	**BswServiceDependency**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specialization of ServiceDependency in the context of an BswInternalBehavior. It allows to associate BswModuleEntries and data defined for a BSW module or cluster to a given ServiceNeeds element.			
Base	*ARObject*, *ServiceDependency*			
Aggregated by	BswInternalBehavior.serviceDependency			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
assignedData	RoleBasedData Assignment	*	aggr	Defines the role of an associated data object (owned by this module or cluster) in the context of the ServiceNeeds element. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedData, assignedData.variation Point.shortLabel vh.latestBindingTime=preCompileTime
assignedEntry Role	RoleBasedBswModule EntryAssignment	*	aggr	Defines the role of an associated BswModuleEntry in the context of the ServiceNeeds element. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedEntryRole, assignedEntry Role.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
ident	BswService DependencyIdent	0..1	aggr	This adds the ability to become referrable to BswService Dependency. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-100
serviceNeeds	ServiceNeeds	0..1	aggr	The associated ServiceNeeds.

Table F.15: BswServiceDependency

Class	**CompuConstFormulaContent**			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the constant value of the computation method is represented by a variation point. This difference is due to compatibility with ASAM HDO.			
Base	*ARObject*, *CompuConstContent*			
Aggregated by	CompuConst.compuConstContentType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
vf	Numerical	1	attr	Value calculated via a system constant. This element is included in every case where parameters should be generated from numerical values during compile time (not runtime!). Thus for example, the influence of the cylinder number on conversion formulae can be introduced in a repeatable manner. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=30

Table F.16: CompuConstFormulaContent

Class	**ConsumedEventGroup**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	This element represents an event-group to which the service consumer wants to subscribe.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ConsumedServiceInstance.consumedEventGroup			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
application Endpoint	ApplicationEndpoint	0..1	ref	Defines the application endpoint where the events of the event group are received in case of multicast reception. **Tags:**atp.Status=obsolete
autoRequire	Boolean	0..1	attr	Defines that this ConsumedEventGroup shall be requested (subscribed) as soon as the corresponding ConsumedServiceInstance is requested. This could be at ECU start, if ConsumedServiceInstance.autoRequire is set to TRUE or as soon as the ConsumedServiceInstance is requested by the application, if ConsumedService Instance.autoRequire is set to FALSE.
eventGroup Identifier	PositiveInteger	0..1	attr	EventGroup ID. Shall be unique within one system to allow service discovery.
eventMulticast Address	ApplicationEndpoint	*	ref	This reference defines the multicast address or a multicast address resource where the events of the event group are received. If the multicast address is determined via configuration and not at runtime via service discovery this reference points to the multicast address over which the events will be received. If the multicast address is determined at runtime via service discovery this reference shall be used to define the necessary local multicast address resources, i.e. RAM space in the TcpIp module in which the multicast address is stored at runtime. Please note that in this case the referenced address may be defined as ANY UDP port and ANY IP address since the multicast address will be received at runtime. If several multicast addresses are considered to be used the ConsumedEventGroup shall ▽

▽

△

Class	ConsumedEventGroup			
				△ point to different ApplicationEndpoint objects to reserve the necessary resources in the configuration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=eventMulticastAddress.applicationEndpoint, eventMulticastAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
pduActivation RoutingGroup	PduActivationRouting Group	*	aggr	The ServiceDiscovery module is able to activate and deactivate the PDU routing for receiving events.
priority	PositiveInteger	0..1	attr	Defines the frame priority where values from 0 (best effort) to 7 (highest) are allowed.
routingGroup	SoAdRoutingGroup	*	ref	The ServiceDiscovery module is able to activate and deactivate the PDU routing for receiving events. **Tags:**atp.Status=obsolete
sdClientConfig	SdClientConfig	0..1	aggr	The readiness to receive events is defined by the Service Discovery of the ConsumedEventGroup. The Event Handler shall know about this announcement to decide about the submission of events. Therefore the Event Handler may be configured with Service-Discovery Client attributes. **Tags:**atp.Status=obsolete
sdClientTimer Config	SomeipSdClientEvent GroupTimingConfig	0..1	ref	Client Timing configuration settings that are EventGroup specific. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sdClientTimerConfig.someipSdClientEvent GroupTimingConfig, sdClientTimerConfig.variation Point.shortLabel vh.latestBindingTime=postBuild

Table F.17: ConsumedEventGroup

Class	ConsumedServiceInstance			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Service instances that are consumed by the ECU that is connected via the ApplicationEndpoint to a CommunicationConnector.			
Base	*ARObject*, *AbstractServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationEndpoint.consumedServiceInstance, ServiceInstanceCollectionSet.serviceInstance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
autoRequire	Boolean	0..1	attr	Defines that this ConsumedServiceInstance shall be required (searched for) by the service discovery at ECU start.
blocklisted Version	SomeipServiceVersion	*	aggr	Collection of blocklisted versions **Tags:**atp.Status=draft
consumedEvent Group	ConsumedEventGroup	*	aggr	Selection of event-groups the consumer wants to subscribe for. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=consumedEventGroup.shortName, consumedEventGroup.variationPoint.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	**ConsumedServiceInstance**			
eventMulticast Subscription Address	ApplicationEndpoint	0..1	ref	Multicast Address that is used by the client to subscribe to the server: This enables the multicast subscription feature. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=eventMulticastSubscription Address.applicationEndpoint, eventMulticastSubscription Address.variationPoint.shortLabel vh.latestBindingTime=postBuild
instance Identifier	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID.
localUnicast Address	ApplicationEndpoint	0..2	ref	The local address over which the CSI is consumed (udp, tcp or both). **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=localUnicastAddress.applicationEndpoint, localUnicastAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
minorVersion	AnyVersionString	0..1	attr	Minor Version of the ServiceInterface. Value can be set to a number that represents the Minor Version of the searched service or to ANY.
providedService Instance	ProvidedService Instance	0..1	ref	Reference to a providedServiceInstance to get the instanceIdentifier information from the ProvidedService Instance. **Tags:**atp.Status=obsolete
remoteUnicast Address	ApplicationEndpoint	0..2	ref	This reference defines the remote address where the service provider is located. This reference shall ONLY be used if the remote address is determined from the configuration and not at runtime from the Service Discovery. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=remoteUnicastAddress.applicationEndpoint, remoteUnicastAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
sdClientConfig	SdClientConfig	0..1	aggr	Service Discovery Client configuration. **Tags:**atp.Status=obsolete
sdClientTimer Config	SomeipSdClientService InstanceConfig	0..1	ref	Client specific configuration settings relevant for the SOME/IP service discovery. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sdClientTimerConfig.someipSdClientService InstanceConfig, sdClientTimerConfig.variationPoint.short Label vh.latestBindingTime=postBuild
serviceIdentifier	PositiveInteger	0..1	attr	This attribute represents the ability to describe the SOME/IP service ID that is searched.
versionDriven FindBehavior	ServiceVersion AcceptanceKindEnum	0..1	attr	Defines the service discovery find behavior. **Tags:**atp.Status=draft

Table F.18: ConsumedServiceInstance

Class	***DataMapping*** (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of port elements (data elements and parameters) to frames and signals.			
Base	*ARObject*			
Subclasses	ClientServerToSignalMapping, SenderReceiverCompositeElementToSignalMapping, SenderReceiverToSignalGroupMapping, SenderReceiverToSignalMapping, TriggerToSignalMapping			
Aggregated by	SystemMapping.dataMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
introduction	DocumentationBlock	0..1	aggr	This represents introductory documentation about the data mapping.

Table F.19: DataMapping

Class	***Describable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	This meta-class represents the ability to add a descriptive documentation to non identifiable elements.			
Base	*ARObject*			
Subclasses	CyclicTiming, EventControlledTiming, HwElementConnector, HwPinConnector, HwPinGroupConnector, IPduTiming, Ipv4DhcpServerConfiguration, Ipv6DhcpServerConfiguration, PncMapping, SocketConnection, *TransformationComSpecProps*, *TransformationDescription*, *TransformationISignalProps*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
adminData	AdminData	0..1	aggr	This represents the administrative data for the describable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-20
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Describable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30

Table F.20: Describable

Class	**DiagnosticDataElement**
Package	M2::AUTOSARTemplates::DiagnosticExtract::CommonDiagnostics
Note	This meta-class represents the ability to describe a concrete piece of data to be taken into account for diagnostic purposes.
Base	*ARObject*, *DiagnosticServiceMappingDiagTarget*, *Identifiable*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	DiagnosticDataElement			
Aggregated by	*DiagnosticAbstractParameter*.dataElement			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
maxNumberOf Elements	PositiveInteger	0..1	attr	The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take.
scalingInfoSize	PositiveInteger	0..1	attr	Size in bytes of scaling information for the DiagnosticData Element if used with DiagnosticReadScalingDataBy Identifier
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties in order to support the definition of e.g. computation formulae and data constraints. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps

Table F.21: DiagnosticDataElement

Class	DltArgument			
Package	M2::AUTOSARTemplates::LogAndTraceExtract			
Note	This element defines an Argument in a DltMessage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	DltArgument.dltArgumentEntry, DltMessage.dltArgument			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dltArgument Entry	DltArgument	*	aggr	This aggregation is used to describe subElements of a Dlt Argument that defines a Structure.
length	PositiveInteger	0..1	attr	Describes the DltArgument length in case of Arrays and Strings in number of BaseTypes.
network Representation	SwDataDefProps	0..1	aggr	Definition of the networkRepresentation of the Dlt Argument. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=networkRepresentation
optional	Boolean	0..1	attr	This attribute defines whether the argument is optional or not. If set to true, the argument can be omitted from the payload of a DLT message.
predefinedText	Boolean	0..1	attr	This attribute defines whether the DltArgument is a predefinedText (Static Data).
variableLength	Boolean	0..1	attr	This attribute defines whether the length of the Dlt Argument is variable (determined at runtime) or not.

Table F.22: DltArgument

Class	<<atpMixed>> **DocumentationBlock**
Package	M2::MSR::Documentation::BlockElements
Note	This class represents a documentation block. It is made of basic text structure elements which can be displayed in a table cell.
Base	*ARObject*

▽

△

Class	<<atpMixed>> **DocumentationBlock**			
Aggregated by	AUTOSAR.introduction, BlueprintGenerator.introduction, *BlueprintPolicyModifiable*.blueprintDerivationGuide, ClientServerOperationBlueprintMapping.blueprintMappingGuide, *DataMapping*.introduction, DefItem.def, *Describable*.introduction, EcucAddInfoParamValue.value, EcuResourceEstimation.introduction, Entry.entryContents, FrameMapping.introduction, *GeneralAnnotation*.annotationText, *Identifiable*.introduction, IPduMapping.introduction, ISignalMapping.introduction, Item.itemContents, LabeledItem.itemContents, LifeCycleInfo.remark, *MappingConstraint*.introduction, MsrQueryP2.msrQueryResultP2, Note.noteText, PortDefinedArgumentBlueprint.blueprintMappingGuide, PrmChar.cond, PrmChar.remark, *ScheduleTableEntry*.introduction, *SignalPathConstraint*.introduction, StructuredReq.conflicts, StructuredReq.dependencies, StructuredReq.description, StructuredReq.rationale, StructuredReq.remark, StructuredReq.supportingMaterial, StructuredReq.useCase, SwAxisType.swGenericAxisDesc, TopicContent.blockLevelContent, TraceableText.text, VariationPoint.blueprintCondition			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
defList	DefList	0..1	aggr	This represents a definition list in the documentation block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=defList, defList.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=40
figure	MlFigure	0..1	aggr	This represents a figure in the documentation block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=figure, figure.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=70
formula	MlFormula	0..1	aggr	This is a formula in the definition block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=formula, formula.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=60
labeledList	LabeledList	0..1	aggr	This represents a labeled list. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=labeledList, labeledList.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=50
list	List	0..1	aggr	This represents numbered or unnumbered list. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=list, list.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=30
msrQueryP2	MsrQueryP2	0..1	aggr	This represents automatically contributed contents provided by an msrquery in the context of DocumentationBlock.
note	Note	0..1	aggr	This represents a note in the text flow. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=note, note.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=80

▽

△

Class	<<atpMixed>> **DocumentationBlock**			
p	MultiLanguage Paragraph	0..1	aggr	This is one particular paragraph. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=p, p.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10
structuredReq	StructuredReq	0..1	aggr	This aggregation supports structured requirements embedded in a documentation block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=structuredReq.shortName, structured Req.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=100
trace	TraceableText	0..1	aggr	This represents traceable text in the documentation block. This allows to specify requirements/constraints in any documentation block. The kind of the trace is specified in the category. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=trace.shortName, trace.variationPoint.short Label vh.latestBindingTime=postBuild xml.sequenceOffset=90
verbatim	MultiLanguageVerbatim	0..1	aggr	This represents one particular verbatim text. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=verbatim, verbatim.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=20

Table F.23: DocumentationBlock

Class	**EcuInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreTopology			
Note	ECUInstances are used to define the ECUs used in the topology. The type of the ECU is defined by a reference to an ECU specified with the ECU resource description. **Tags:**atp.recommendedPackage=EcuInstances			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
associatedCom IPduGroup	ISignalIPduGroup	*	ref	With this reference it is possible to identify which ISignal IPduGroups are applicable for which Communication Connector/ ECU. Only top level ISignalIPduGroups shall be referenced by an EcuInstance. If an ISignalIPduGroup contains other ISignalIPduGroups than these contained ISignalIPdu Groups shall not be referenced by the EcuInstance. Contained ISignalIPduGroups are associated to an Ecu Instance via the top level ISignalIPduGroup.

▽

△

Class	**EcuInstance**			
associated Consumed Provided ServiceInstance Group	ConsumedProvided ServiceInstanceGroup	*	ref	With this reference it is possible to identify which ConsumedProvidedServiceInstanceGroups are applicable for which ECUInstance. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=associatedConsumedProvidedService InstanceGroup.consumedProvidedServiceInstanceGroup, associatedConsumedProvidedServiceInstance Group.variationPoint.shortLabel vh.latestBindingTime=postBuild
associatedPdur IPduGroup	PdurIPduGroup	*	ref	With this reference it is possible to identify which PduR IPdu Groups are applicable for which Communication Connector/ ECU.
clientIdRange	ClientIdRange	0..1	aggr	Restriction of the Client Identifier for this Ecu to an allowed range of numerical values. The Client Identifier of the transaction handle is generated by the client RTE for inter-Ecu Client/Server communication.
com Configuration GwTimeBase	TimeValue	0..1	attr	The period between successive calls to Com_Main FunctionRouteSignals of the AUTOSAR COM module in seconds.
com ConfigurationRx TimeBase	TimeValue	0..1	attr	The period between successive calls to Com_Main FunctionRx of the AUTOSAR COM module in seconds.
com ConfigurationTx TimeBase	TimeValue	0..1	attr	The period between successive calls to Com_Main FunctionTx of the AUTOSAR COM module in seconds.
comEnable MDTForCyclic Transmission	Boolean	0..1	attr	Enables for the Com module of this EcuInstance the minimum delay time monitoring for cyclic and repeated transmissions (TransmissionModeTiming has cyclic Timing assigned or eventControlledTiming with numberOf Repetitions > 0).
commController	Communication Controller	1..*	aggr	CommunicationControllers of the ECU. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=commController.shortName, comm Controller.variationPoint.shortLabel vh.latestBindingTime=postBuild
connector	Communication Connector	*	aggr	All channels controlled by a single controller. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variation Point.shortLabel vh.latestBindingTime=postBuild
dltConfig	DltConfig	0..1	aggr	Describes the Dlt configuration on this EcuInstance.
doIpConfig	DoIpConfig	0..1	aggr	DoIp configuration on this EcuInstance. **Tags:**atp.Status=draft
ecuTaskProxy	OsTaskProxy	*	ref	Reference to OsTaskProxies assigned to the Ecu Instance. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=ecuTaskProxy
ethSwitchPort Group Derivation	Boolean	0..1	attr	Defines whether the derivation of SwitchPortGroups based on VLAN and/or CouplingPort.pncMapping shall be performed for this EcuInstance. If not defined the derivation shall not be done.
partition	EcuPartition	*	aggr	Optional definition of Partitions within an Ecu.

▽

△

Class	**EcuInstance**			
pncNmRequest	Boolean	0..1	attr	Defines if this EcuInstance shall request Nm on all its PhysicalChannels which have Nm variant set to FULL each time a PNC is requested.
pncPrepare SleepTimer	TimeValue	0..1	attr	Time in seconds the PNC state machine shall wait in PNC_PREPARE_SLEEP.
pnc Synchronous Wakeup	Boolean	0..1	attr	If this parameter is available and set to true then all available PNCs will be woken up as soon as a channel wakeup occurs. This is ensured by adding all PNCs to all channel wakeup sources during upstream mapping.
pnResetTime	TimeValue	0..1	attr	Specifies the runtime of the reset timer in seconds. This reset time is valid for the reset of PN requests in the EIRA and in the ERA.
sleepMode Supported	Boolean	1	attr	Specifies whether the ECU instance may be put to a "low power mode" • true: sleep mode is supported • false: sleep mode is not supported Note: This flag may only be set to "true" if the feature is supported by both hardware and basic software.
tcpIpIcmpProps	EthTcpIpIcmpProps	0..1	ref	EcuInstance specific ICMP (Internet Control Message Protocol) attributes
tcpIpProps	EthTcpIpProps	0..1	ref	EcuInstance specific TcpIp Stack attributes.
v2xSupported	V2xSupportEnum	0..1	attr	This attribute is used to control the existence of the V2X stack on the given EcuInstance.
wakeUpOver BusSupported	Boolean	1	attr	Driver support for wakeup over Bus.

Table F.24: EcuInstance

Class	**EndToEndProtectionISignalIPdu**			
Package	M2::AUTOSARTemplates::SystemTemplate::EndToEndProtection			
Note	It is possible to protect the inter-ECU data exchange of safety-related ISignalGroups at the level of COM IPdus using protection mechanisms provided by E2E Library. For each ISignalGroup to be protected, a separate EndToEndProtectionISignalIPdu element shall be created within the EndToEndProtectionSet. The EndToEndProtectionISignalIPdu element refers to the ISignalGroup that is to be protected and to the ISignalIPdu that transmits the protected ISignalGroup. The information how the referenced ISignalGroup shall be protected (through which E2E Profile and with which E2E settings) is defined in the EndToEnd Description element.			
Base	*ARObject*			
Aggregated by	EndToEndProtection.endToEndProtectionISignalIPdu			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataOffset	Integer	1	attr	This attribute defines the beginning offset (in bits) of the Array representation of the Signal Group (including CRC, counter and application signal group) in the IPdu. This attribute is mandatory and the dataOffset shall always be defined.
iSignalGroup	ISignalGroup	1	ref	Reference to the ISignalGroup that is to be protected.
iSignalIPdu	ISignalIPdu	1	ref	Reference to the ISignalIPdu that transmits the protected ISignalGroup.

Table F.25: EndToEndProtectionISignalIPdu

Class	**EndToEndTransformationDescription**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	EndToEndTransformationDescription holds these attributes which are profile specific and have the same value for all E2E transformers.			
Base	*ARObject*, *Describable*, *TransformationDescription*			
Aggregated by	TransformationTechnology.transformationDescription			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clearFromValidToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
counterOffset	PositiveInteger	0..1	attr	Offset of the counter in the Data[] array in bits.
crcOffset	PositiveInteger	0..1	attr	Offset of the CRC in the Data[] array in bits.
dataIdMode	DataIdModeEnum	0..1	attr	This attribute describes the inclusion mode that is used to include the implicit two-byte Data ID in the one-byte CRC.
dataIdNibbleOffset	PositiveInteger	0..1	attr	Offset of the Data ID nibble in the Data[] array in bits.
e2eProfileCompatibilityProps	E2EProfileCompatibilityProps	0..1	ref	Reference to additional settings for the E2E state machine.
maxDeltaCounter	PositiveInteger	0..1	attr	Maximum allowed difference between two counter values of two consecutively received valid messages. For example, if the receiver gets data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
maxErrorStateInit	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INIT.
maxErrorStateInvalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.
maxErrorStateValid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_VALID.
maxNoNewOrRepeatedData	PositiveInteger	0..1	attr	The maximum allowed amount of consecutive failed counter checks.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT.
minOkStateInvalid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.
minOkStateValid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID.
offset	PositiveInteger	0..1	attr	Offset of the E2E header in the Data[] array in bits.
profileBehavior	EndToEndProfileBehaviorEnum	0..1	attr	Behavior of the check functionality
profileName	NameToken	1	attr	Definition of the E2E profile.
syncCounterInit	PositiveInteger	0..1	attr	Number of checks required for validating the consistency of the counter that shall be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.

▽

△

Class	**EndToEndTransformationDescription**			
upperHeader BitsToShift	PositiveInteger	0..1	attr	This attribute describes the number of upper-header bits to be shifted. value = 0 or not present: shift of upper header is NOT performed. value > 0: the E2E Transformer on the protect-side, takes the first upperHeaderBitsToShift bits from the upper buffer (e.g. SOME/IP header part generated by SOME/IP transformer) and shifts them towards the lower bytes and bits within the Data[] for the length of the E2E header (e.g. 12 bytes in case of E2E Profile 4). This means the shift distance is fixed - it depends on the E2E header size - what is configured here is the number of bits that are to be shifted. This option is defined because the Some/IP header generated by SOME/IP transformer shall be, due to compatibility between non-protected and E2E-protected communication, at the same position, which is before E2E header.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table F.26: EndToEndTransformationDescription

Class	**FlatInstanceDescriptor**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	Represents exactly one node (e.g. a component instance or data element) of the instance tree of a software system. The purpose of this element is to map the various nested representations of this instance to a flat representation and assign a unique name (shortName) to it. Use cases: • Specify unique names of measurable data to be used by MCD tools • Specify unique names of calibration data to be used by MCD tool • Specify a unique name for an instance of a component prototype in the ECU extract of the system description Note that in addition it is possible to assign alias names via AliasNameAssignment.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	FlatMap.instance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	FlatInstanceDescriptor			
ecuExtract Reference	AtpFeature	0..1	iref	Refers to the instance in the ECU extract. This is valid only, if the FlatMap is used in the context of an ECU extract. The reference shall be such that it uniquely defines the object instance. For example, if a data prototype is declared as a role within an SwcInternalBehavior, it is not enough to state the SwcInternalBehavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying instance of the component prototype and the Atomic SoftwareComponentType, which is refered by the particular SwcInternalBehavior. **Tags:**xml.sequenceOffset=40 **InstanceRef implemented by:**AnyInstanceRef
role	Identifier	0..1	attr	The role denotes the particular role of the downstream memory location described by this FlatInstanceDescriptor. It applies to use case where one upstream object results in multiple downstream objects, e.g. ModeDeclaration GroupPrototypes which are measurable. In this case the RTE will provide locations for current mode, previous mode and next mode.
rtePluginProps	RtePluginProps	0..1	aggr	The properties of a communication graph with respect to the utilization of RTE Implementation Plug-in. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=rtePluginProps
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this FlatInstanceDescriptor. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=swDataDefProps
upstream Reference	AtpFeature	0..1	iref	Refers to the instance in the context of an "upstream" descriptions, wich could be the system or system extract description, the basic software module description or (if a flat map is used in preliminary context) a description of an atomic component or composition. This reference is optional in case the flat map is used in ECU context. The reference shall be such that it uniquely defines the object instance in the given context. For example, if a data prototype is declared as a role within an SwcInternal Behavior, it is not enough to state the SwcInternal Behavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying the instance of the component prototype that contains the particular instance of Swc InternalBehavior. **Tags:**xml.sequenceOffset=20 **InstanceRef implemented by:**AnyInstanceRef

Table F.27: FlatInstanceDescriptor

Class	***HwDescriptionEntity*** (abstract)			
Package	M2::AUTOSARTemplates::EcuResourceTemplate			
Note	This meta-class represents the ability to describe a hardware entity.			
Base	*ARObject*, *Referrable*			
Subclasses	HwElement, HwPin, HwPinGroup, HwType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***HwDescriptionEntity*** (abstract)			
hwAttribute Value	HwAttributeValue	*	aggr	This aggregation represents a particular hardware attribute value. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=hwAttributeValue, hwAttributeValue.variation Point.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=50
hwCategory	HwCategory	*	ref	One of the associations representing one particular category of the hardware entity. **Tags:**xml.sequenceOffset=30
hwType	HwType	0..1	ref	This association is used to assign an optional HwType which contains the common attribute values for all occurences of this HwDescriptionEntity. Note that Hw Types can not be redefined and therefore shall not have a hwType reference.

Table F.28: HwDescriptionEntity

Class	**HwElement**			
Package	M2::AUTOSARTemplates::EcuResourceTemplate			
Note	This represents the ability to describe Hardware Elements on an instance level. The particular types of hardware are distinguished by the category. This category determines the applicable attributes. The possible categories and attributes are defined in HwCategory. **Tags:**atp.recommendedPackage=HwElements			
Base	*ARElement*, *ARObject*, *CollectableElement*, *HwDescriptionEntity*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
hwElement Connection	HwElementConnector	*	aggr	This represents one particular connection between two hardware elements. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=hwElementConnection, hwElement Connection.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=110
hwPinGroup	HwPinGroup	*	aggr	This aggregation is used to describe the connection facilities of a hardware element. Note that hardware element has no pins but only pingroups. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=hwPinGroup.shortName, hwPin Group.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=90

▽

△

Class	HwElement			
nestedElement	HwElement	*	ref	This association is used to establish hierarchies of hw elements. Note that one particular HwElement can be target of this association only once. I.e. multiple instantiation of the same HwElement is not supported (at any hierarchy level). **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=nestedElement.hwElement, nested Element.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=70

Table F.29: HwElement

Class	**HwType**			
Package	M2::AUTOSARTemplates::EcuResourceTemplate::HwElementCategory			
Note	This represents the ability to describe Hardware types on an abstract level. The particular types of hardware are distinguished by the category. This category determines the applicable attributes. The possible categories and attributes are defined in HwCategory. **Tags:**atp.recommendedPackage=HwTypes			
Base	*ARElement*, *ARObject*, *CollectableElement*, *HwDescriptionEntity*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table F.30: HwType

Class	**ISignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different SignalIPdus to multiple receivers. To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping. ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping). In case of the SystemSignalGroup an ISignal shall be created for each SystemSignal contained in the SystemSignalGroup. **Tags:**atp.recommendedPackage=ISignals			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
data Transformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignal. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataTransformation.dataTransformation, dataTransformation.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

▽

△

Class	**ISignal**			
dataTypePolicy	DataTypePolicyEnum	1	attr	With the aggregation of SwDataDefProps an ISignal specifies how it is represented on the network. This representation follows a particular policy. Note that this causes some redundancy which is intended and can be used to support flexible development methodology as well as subsequent integrity checks. If the policy "networkRepresentationFromComSpec" is chosen the network representation from the ComSpec that is aggregated by the PortPrototype shall be used. If the "override" policy is chosen the requirements specified in the PortInterface and in the ComSpec are not fulfilled by the networkRepresentationProps. In case the System Description doesn't use a complete Software Component Description (VFB View) the "legacy" policy can be chosen.
initValue	ValueSpecification	0..1	aggr	Optional definition of a ISignal's initValue in case the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy system signals. This value can be used to configure the Signal's "Init Value". If a full DataMapping exist for the SystemSignal this information may be available from a configured Sender ComSpec and ReceiverComSpec. In this case the initvalues in SenderComSpec and/or ReceiverComSpec override this optional value specification. Further restrictions apply from the RTE specification.
iSignalProps	ISignalProps	0..1	aggr	Additional optional ISignal properties that may be stored in different files. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=iSignalProps
iSignalType	ISignalTypeEnum	0..1	attr	This attribute defines whether this iSignal is an array that results in a UINT8_N / UINT8_DYN ComSignalType in the COM configuration or a primitive type.
length	UnlimitedInteger	1	attr	Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals. The ISignal length of zero bits is allowed.
network Representation Props	SwDataDefProps	0..1	aggr	Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used. The attribute "dataTypePolicy" in the SystemTemplate element defines whether this network representation shall be ignored and the information shall be taken over from the network representation of the ComSpec. If "override" is chosen by the system integrator the network representation can violate against the requirements defined in the PortInterface and in the network representation of the ComSpec. In case that the System Description doesn't use a complete Software Component Description (VFB View) ▽

▽

△

Class	**ISignal**			
				△ this element is used to configure "ComSignalDataInvalid Value" and the Data Semantics. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=networkRepresentationProps
systemSignal	SystemSignal	1	ref	Reference to the System Signal that is supposed to be transmitted in the ISignal.
timeout Substitution Value	ValueSpecification	0..1	aggr	Defines and enables the ComTimeoutSubstituition for this ISignal.
transformation ISignalProps	TransformationISignal Props	*	aggr	A transformer chain consists of an ordered list of transformers. The ISignal specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignals are described in the TransformationTechnology class. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=transformationISignalProps

Table F.31: ISignal

Class	**ISignalGroup**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	SignalGroup of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal Group is sent in different SignalIPdus to multiple receivers. An ISignalGroup refers to a set of ISignals that shall always be kept together. A ISignalGroup represents a COM Signal Group. Therefore it is recommended to put the ISignalGroup in the same Package as ISignals (see atp.recommendedPackage) **Tags:**atp.recommendedPackage=ISignalGroup			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
comBased SignalGroup Transformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignalGroup based on the COMBasedTransformer approach. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=comBasedSignalGroupTransformation.data Transformation, comBasedSignalGroup Transformation.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
iSignal	ISignal	*	ref	Reference to a set of ISignals that shall always be kept together.
systemSignal Group	SystemSignalGroup	1	ref	Reference to the SystemSignalGroup that is defined on VFB level and that is supposed to be transmitted in the ISignalGroup.

▽

△

Class	ISignalGroup			
transformationISignalProps	TransformationISignalProps	*	aggr	A transformer chain consists of an ordered list of transformers. The ISignalGroup specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignalGroups are described in the TransformationTechnology class. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=transformationISignalProps

Table F.32: ISignalGroup

Class	ISignalIPdu			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Represents the IPdus handled by Com. The ISignalIPdu assembled and disassembled in AUTOSAR COM consists of one or more signals. In case no multiplexing is performed this IPdu is routed to/from the Interface Layer. A maximum of one dynamic length signal per IPdu is allowed. **Tags:**atp.recommendedPackage=Pdus			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Pdu*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
iPduTimingSpecification	IPduTiming	0..1	aggr	Timing specification for Com IPdus (Transmission Modes). This information is mandatory for the sender in a System Extract. This information may be omitted on receivers in a System Extract. atpVariation: The timing of a Pdu can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=iPduTimingSpecification, iPduTimingSpecification.variationPoint.shortLabel vh.latestBindingTime=postBuild
iSignalToPduMapping	ISignalToIPduMapping	*	aggr	Definition of SignalToIPduMappings included in the SignalIPdu. atpVariation: The content of a PDU can be variable. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=iSignalToPduMapping.shortName, iSignalToPduMapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
unusedBitPattern	Integer	1	attr	AUTOSAR COM and AUTOSAR IPDUM are filling not used areas of an IPDU with this bit-pattern. This attribute is mandatory to avoid undefined behavior. This byte-pattern will be repeated throughout the IPdu.

Table F.33: ISignalIPdu

Class	***Identifiable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTaskProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressableObject*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, BlockState, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, *CpSoftwareClusterResource*, CpSoftwareClusterResourceToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareClusterToResourceMapping, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventInSystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextProps, *ServerCallPoint*, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TDCpSoftwareClusterMapping, TDCpSoftwareClusterResourceMapping, TcpOptionFilterList, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40

▽

△

Class	*Identifiable* (abstract)			
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table F.34: Identifiable

Class	**InstantiationTimingEventProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	This meta-class represents the ability to refine a timing event for particular instances of a software component. This approach supports an instance specific timing.			
Base	*ARObject*, *InstantiationRTEEventProps*			
Aggregated by	CompositionSwComponentType.instantiationRTEEventProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
period	TimeValue	0..1	attr	This attribute represents the value of the refined activation period.

Table F.35: InstantiationTimingEventProps

Class	McDataInstance			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Describes the specific properties of one data instance in order to support measurement and/or calibration of this data instance. The most important attributes are: • Its shortName is copied from the ECU Flat map (if applicable) and will be used as identifier and for display by the MC system. • The category is copied from the corresponding data type (ApplicationDataType if defined, otherwise ImplementationDataType) as far as applicable. • The symbol is the one used in the programming language. It will be used to find out the actual memory address by the final generation tool with the help of linker generated information. It is assumed that in the M1 model this part and all the aggregated and referred elements (with the exception of the Flat Map and the references from ImplementationElementInParameterInstanceRef and McAccessDetails) are completely generated from "upstream" information. This means, that even if an element like e.g. a CompuMethod is only used via reference here, it will be copied into the M1 artifact which holds the complete McSupportData for a given Implementation.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	McDataInstance.subElement, McSupportData.mcParameterInstance, McSupportData.mcVariableInstance			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of number of elements.
displayIdentifier	McdIdentifier	0..1	attr	An optional attribute to be used to set the ASAM ASAP2 DISPLAY_IDENTIFIER attribute.
flatMapEntry	FlatInstanceDescriptor	0..1	ref	Reference to the corresponding entry in the ECU Flat Map. This allows to trace back to the original specification of the generated data instance. This link shall be added by the RTE generator mainly for documentation purposes. The reference is optional because • The McDataInstance may represent an array or struct in which only the subElements correspond to FlatMap entries. • The McDataInstance may represent a task local buffer for rapid prototyping access which is different from the "main instance" used for measurement access.
instanceInMemory	ImplementationElementInParameterInstanceRef	0..1	aggr	Reference to the corresponding data instance in the description of calibration data structures published by the RTE generator. This is used to support emulation methods inside the ECU, it is not required for A2L generation.
mcDataAccessDetails	McDataAccessDetails	0..1	aggr	Refers to "upstream" information on how the RTE uses this data instance. Use Case: Rapid Prototyping
mcDataAssignment	RoleBasedMcDataAssignment	*	aggr	An assignment between McDataInstances. This supports the indication of related McDataElement implementing the of "RP global buffer", "RP global measurement buffer", "RP enabler flag".
resultingProperties	SwDataDefProps	0..1	aggr	These are the generated properties resulting from decisions taken by the RTE generator for the actually implemented data instance. Only those properties are relevant here, which are needed for the measurement and calibration system. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=resultingProperties

▽

△

Class	**McDataInstance**			
resultingRptSwPrototypingAccess	RptSwPrototypingAccess	0..1	aggr	Describes the implemented accessibility of data and modes by the rapid prototyping tooling.
role	Identifier	0..1	attr	An optional attribute to be used for additional information on the role of this data instance, for example in the context of rapid prototyping.
rptImplPolicy	RptImplPolicy	0..1	aggr	Describes the implemented code preparation for rapid prototyping at data accesses for a hook based bypassing.
subElement (ordered)	McDataInstance	*	aggr	This relation indicates, that the target element is part of a "struct" which is given by the source element. This information will be used by the final generator to set up the correct addressing scheme. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=subElement.shortName, subElement.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbol	SymbolString	0..1	attr	This String is used to determine the memory address during final generation of the MC configuration data (e.g. "A2L" file) . It shall be the name of the element in the programming language such that it can be identified in linker generated information. In case the McDataInstance is part of composite data in the programming language, the symbol String may include parts denoting the element context, unless the context is given by the symbol attribute of an enclosing McDataInstance. This means in particular for the C language that the "." character shall be used as a separator between the name of a "struct" variable the name of one of its elements. The symbol can differ from the shortName in case of generated C data declarations. It is an optional attribute since it may be missing in case the instance represents an element (e.g. a single array element) which has no name in the linker map. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbol

Table F.36: McDataInstance

Class	**McSupportData**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Root element for all measurement and calibration support data related to one Implementation artifact on an ECU. There shall be one such element related to the RTE implementation (if it owns MC data) and a separate one for each module or component, which owns private MC data.			
Base	*ARObject*			
Aggregated by	*Implementation*.mcSupport			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	McSupportData			
emulation Support	McSwEmulationMethod Support	*	aggr	Describes the calibration method used by the RTE. This information is not needed for A2L generation, but to setup software emulation in the ECU. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=emulationSupport, emulation Support.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
mcParameter Instance	McDataInstance	*	aggr	A data instance to be used for calibration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=mcParameterInstance.shortName, mc ParameterInstance.variationPoint.shortLabel vh.latestBindingTime=postBuild
mcVariable Instance	McDataInstance	*	aggr	A data instance to be used for measurement. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=mcVariableInstance.shortName, mcVariable Instance.variationPoint.shortLabel vh.latestBindingTime=postBuild
measurable System ConstantValues	SwSystemconstant ValueSet	*	ref	Sets of system constant values to be transferred to the MCD system, because the system constants have been specified with "swCalibrationAccess" = readonly.
rptSupportData	RptSupportData	0..1	aggr	The rapid prototyping support data belonging to this implementation. The aggregtion is <<atpSplitable>> because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=rptSupportData

Table F.37: McSupportData

Class	***MultilanguageReferrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders). They also may have a longName. But they are not considered to contribute substantially to the overall structure of an AUTOSAR description. In particular it does not contain other Referrables.			
Base	*ARObject*, *Referrable*			
Subclasses	Caption, DefItem, DocumentationContext, *Identifiable*, SdgCaption, *TraceReferrable*, *Traceable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
longName	MultilanguageLong Name	0..1	aggr	This specifies the long name of the object. Long name is targeted to human readers and acts like a headline.

Table F.38: MultilanguageReferrable

Class	ParameterInAtomicSWCTypeInstanceRef
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements::InstanceRefs Usage
Note	This class implements an instance reference which can be applied for variables as well as for parameters.
Base	*ARObject*, *AtpInstanceRef*

▽

△

Class	**ParameterInAtomicSWCTypeInstanceRef**			
Aggregated by	AutosarParameterRef.autosarParameter			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
base	AtomicSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextDataPrototype (ordered)	ApplicationCompositeElementDataPrototype	*	ref	This ist the context in a compositeDataType. **Tags:**xml.sequenceOffset=40
portPrototype	PortPrototype	0..1	ref	This is the port providing the variable or the entry point to the variable structure. **Tags:**xml.sequenceOffset=20
rootParameterDataPrototype	DataPrototype	0..1	ref	This represents the entry point for references into a CompositeDataType. **Tags:**xml.sequenceOffset=30
targetDataPrototype	DataPrototype	0..1	ref	This is the target parameter element. Note that this must be nested in ParameterDataPrototype. The target must be one of ParameterDataPrototype, ApplicationCompositeElementDataPrototype. **Tags:**xml.sequenceOffset=50

Table F.39: ParameterInAtomicSWCTypeInstanceRef

Class	**PostBuildVariantCriterionValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to denote one set of postBuildVariantCriterionValues. **Tags:**atp.recommendedPackage=PostBuildVariantCriterionValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
postBuildVariantCriterionValue	PostBuildVariantCriterionValue	*	aggr	This is is one particular postbuild variant criterion/value pair being part of the PostBuildVariantSet.

Table F.40: PostBuildVariantCriterionValueSet

Class	**ProvidedServiceInstance**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Service instances that are provided by the ECU that is connected via the ApplicationEndpoint to a CommunicationConnector.			
Base	*ARObject*, *AbstractServiceInstance*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationEndpoint.providedServiceInstance, ServiceInstanceCollectionSet.serviceInstance			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
autoAvailable	Boolean	0..1	attr	Defines that this ProvidedServiceInstance shall be offered by the service discovery at ECU start.

▽

△

Class	**ProvidedServiceInstance**			
eventHandler	EventHandler	*	aggr	Collection of event groups provided by the Provided ServiceInstance **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=eventHandler.shortName, event Handler.variationPoint.shortLabel vh.latestBindingTime=postBuild
instance Identifier	PositiveInteger	0..1	attr	Instance identifier. Can be used for e.g. service discovery to identify the instance of the service.
loadBalancing Priority	PositiveInteger	0..1	attr	Defines the value to be used for load balancing priority in the service offer. Lower value means higher priority.
loadBalancing Weight	PositiveInteger	0..1	attr	Defines the value to be used for load balancing weight in the service offer. Higher value means higher probability to be chosen.
localUnicast Address	ApplicationEndpoint	0..2	ref	The local address over which the PSI is provided (udp, tcp or both). **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=localUnicastAddress.applicationEndpoint, localUnicastAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
minorVersion	PositiveInteger	0..1	attr	Minor Version of the Service that is provided by this ProvidedServiceInstance.
priority	PositiveInteger	0..1	attr	Defines the frame priority where values from 0 (best effort) to 7 (highest) are allowed.
remoteMulticast Subscription Address	ApplicationEndpoint	*	ref	This reference defines the remote multicast subscribed addresses of service consumers. This reference shall ONLY be used if the remote address of the clients is determined from the configuration and not at runtime. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=remoteMulticastSubscription Address.applicationEndpoint, remoteMulticast SubscriptionAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
remoteUnicast Address	ApplicationEndpoint	*	ref	This reference defines the remote addresses of service consumers. This reference shall ONLY be used if the remote address of the clients is determined from the configuration and not at runtime. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=remoteUnicastAddress.applicationEndpoint, remoteUnicastAddress.variationPoint.shortLabel vh.latestBindingTime=postBuild
sdServerConfig	SdServerConfig	0..1	aggr	Service Discovery Server configuration. **Tags:**atp.Status=obsolete
sdServerTimer Config	SomeipSdServer ServiceInstanceConfig	0..1	ref	Server specific configuration settings relevant for the SOME/IP service discovery. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sdServerTimerConfig.someipSdServer ServiceInstanceConfig, sdServerTimerConfig.variation Point.shortLabel vh.latestBindingTime=postBuild
serviceIdentifier	PositiveInteger	0..1	attr	This attribute represents the ability to describe the SOME/ IP service ID that is offered.

Table F.41: ProvidedServiceInstance

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table F.42: Referrable

Class	**RootSwCompositionPrototype**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within a given System. According to the use case of the System, this may for example be a more or less complete VFB description, the software of a System Extract or the software of a flat ECU Extract with only atomic SWCs. Therefore the RootSwComposition will only occasionally contain all atomic software components that are used in a complete VFB System. The OEM is primarily interested in the required functionality and the interfaces defining the integration of the Software Component into the System. The internal structure of such a component contains often substantial intellectual property of a supplier. Therefore a top-level software composition will often contain empty compositions which represent subsystems. The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including PortPrototypes, PortInterfaces, VariableDataPrototypes, SwcInternalBehavior etc.), and their ports are interconnected using SwConnectorPrototypes.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, System.rootSoftwareComposition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
calibration ParameterValue Set	CalibrationParameter ValueSet	*	ref	Used CalibrationParameterValueSet for instance specific initialization of calibration parameters. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=calibrationParameterValueSet
flatMap	FlatMap	0..1	ref	The FlatMap used in the scope of this RootSwCompositionPrototype. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=flatMap
software Composition	CompositionSw ComponentType	1	tref	We assume that there is exactly one top-level composition that includes all Component instances of the system. **Stereotypes:** isOfType

Table F.43: RootSwCompositionPrototype

Class	**Sdg**			
Package	M2::MSR::AsamHdo::SpecialData			
Note	Sdg (SpecialDataGroup) is a generic model which can be used to keep arbitrary information which is not explicitly modeled in the meta-model. Sdg can have various contents as defined by sdgContentsType. Special Data should only be used moderately since all elements should be defined in the meta-model. Thereby SDG should be considered as a temporary solution when no explicit model is available. If an sdg Caption is available, it is possible to establish a reference to the sdg structure.			
Base	*ARObject*			
Aggregated by	AdminData.sdg, BuildActionEnvironment.sdg, BuildActionInvocator.sdg, BuildActionIoElement.sdg, File InfoComment.sdg, RptHook.sdg, SdgContents.sdg, VariationPoint.sdg			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
gid	NameToken	1	attr	This attributes specifies an identifier. Gid comes from the SGML/XML-Term "Generic Identifier" which is the element name in XML. The role of this attribute is the same as the name of an XML - element. **Tags:**xml.attribute=true
sdgCaption	SdgCaption	0..1	aggr	This aggregation allows to assign the properties of Identifiable to the sdg. By this, a shortName etc. can be assigned to the Sdg. **Tags:**xml.sequenceOffset=20
sdgContents Type	SdgContents	0..1	aggr	This is the content of the Sdg. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=30 xml.typeElement=false xml.typeWrapperElement=false

Table F.44: Sdg

Class	**SenderReceiverToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a sender receiver communication data element to a signal.			
Base	*ARObject*, *DataMapping*			
Aggregated by	SystemMapping.dataMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1	iref	Reference to the data element. **InstanceRef implemented by:**VariableDataPrototypeIn SystemInstanceRef
senderToSignal TextTable Mapping	TextTableMapping	0..1	aggr	This mapping allows for the text-table translation between the sending DataPrototype that is defined in the Port Prototype and the physicalProps defined for the System Signal.
signalTo ReceiverText TableMapping	TextTableMapping	0..1	aggr	This mapping allows for the text-table translation between the physicalProps defined for the SystemSignal and a receiving DataPrototype that is defined in the Port Prototype.
systemSignal	SystemSignal	1	ref	Reference to the system signal used to carry the data element.

Table F.45: SenderReceiverToSignalMapping

Primitive	**String**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents a String in which white-space shall be normalized before processing. For example: in order to compare two Strings: • leading and trailing white-space needs to be removed • consecutive white-space (blank, cr, lf, tab) needs to be replaced by one blank. **Tags:** xml.xsd.customType=STRING xml.xsd.type=string

Table F.46: String

Class	**SwServiceArg**			
Package	M2::MSR::DataDictionary::ServiceProcessTask			
Note	Specifies the properties of a data object exchanged during the call of an SwService, e.g. an argument or a return value. The SwServiceArg can also be used in the argument list of a C-macro. For this purpose the category shall be set to "MACRO". A reference to implementationDataType can optional be added if the actual argument has an implementationDataType.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	BswModuleEntry.argument, BswModuleEntry.returnType			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
direction	ArgumentDirectionEnum	0..1	attr	Specifies the direction of the data transfer. The direction shall indicate the direction of the actual information that is being consumed by the caller and/or the callee, not the direction of formal arguments in C. The attribute is optional for backwards compatibility reasons. For example, if a pointer is used to pass a memory address for the expected result, the direction shall be "out". If a pointer is used to pass a memory address with content to be read by the callee, its direction shall be "in". **Tags:**xml.sequenceOffset=10
swArraysize	ValueList	0..1	aggr	This turns the argument of the service to an array. **Tags:**xml.sequenceOffset=20
swDataDefProps	SwDataDefProps	0..1	aggr	Data properties of this SwServiceArg. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps xml.sequenceOffset=30

Table F.47: SwServiceArg

Class	<<atpMixedString>> ***SwSystemconstDependentFormula*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class represents an expression depending on system constants.			
Base	*ARObject*, *FormulaExpression*			
Subclasses	*AttributeValueVariationPoint*, BlueprintFormula, ConditionByFormula, *FMFormulaByFeaturesAndSwSystemconsts*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	<<atpMixedString>> ***SwSystemconstDependentFormula*** (abstract)			
sysc	SwSystemconst	0..1	ref	This refers to a system constant. The internal (coded) value of the system constant shall be used. **Tags:**xml.sequenceOffset=50
syscString	SwSystemconst	0..1	ref	syscString indicates that the referenced system constant shall be evaluated as a string according to [TPS_SWCT_01431].

Table F.48: SwSystemconstDependentFormula

Class	**SwSystemconstantValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:**atp.recommendedPackage=SwSystemconstantValueSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
sw Systemconstant Value	SwSystemconstValue	*	aggr	This is one particular value of a system constant.

Table F.49: SwSystemconstantValueSet

Class	**System**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The top level element of the System Description. The System description defines five major elements: Topology, Software, Communication, Mapping and Mapping Constraints. The System element directly aggregates the elements describing the Software, Mapping and Mapping Constraints; it contains a reference to an ASAM FIBEX description specifying Communication and Topology. **Tags:**atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clientId DefinitionSet	ClientIdDefinitionSet	*	ref	Set of Client Identifiers that are used for inter-ECU client-server communication in the System.
containerIPdu HeaderByte Order	ByteOrderEnum	0..1	attr	Defines the byteOrder of the header in ContainerIPdus.
ecuExtract Version	RevisionLabelString	0..1	attr	Version number of the Ecu Extract.

▽

△

Class	**System**			
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=fibexElement.fibexElement, fibex Element.variationPoint.shortLabel vh.latestBindingTime=postBuild
interpolation Routine MappingSet	InterpolationRoutine MappingSet	*	ref	This reference identifies the InterpolationRoutineMapping Sets that are relevant in the context of the enclosing System.
j1939Shared AddressCluster	J1939SharedAddress Cluster	*	aggr	Collection of J1939Clusters that share a common address space for the routing of messages. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=j1939SharedAddressCluster.shortName, j1939SharedAddressCluster.variationPoint.shortLabel vh.latestBindingTime=postBuild
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints). In order to support OEM / Tier 1 interaction and shared development for one common System this aggregation is atpSplitable and atpVariation. The content of System Mapping can be provided by several parties using different names for the SystemMapping. This element is not required when the System description is used for a network-only use-case. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=mapping.shortName, mapping.variation Point.shortLabel vh.latestBindingTime=postBuild
pncVector Length	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rootSoftwareComposition.shortName, root SoftwareComposition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime

▽

△

Class	System			
swCluster	CpSoftwareCluster	*	ref	CP Software Clusters of this System **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swCluster.cpSoftwareCluster, sw Cluster.variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=systemDesignTime
system Documentation	Chapter	*	aggr	Possibility to provide additional documentation while defining the System. The System documentation can be composed of several chapters. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=systemDocumentation.shortName, system Documentation.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=-10
systemVersion	RevisionLabelString	1	attr	Version number of the System Description.

Table F.50: System

Class	**SystemSignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with exactly one system signal defined for each data element prototype sent and received by connected SW component instances. **Tags:**atp.recommendedPackage=SystemSignals			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dynamicLength	Boolean	1	attr	The length of dynamic length signals is variable in run-time. Only a maximum length of such a signal is specified in the configuration (attribute length in ISignal element).
physicalProps	SwDataDefProps	0..1	aggr	Specification of the physical representation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=physicalProps

Table F.51: SystemSignal

Class	**TransientFault**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The reported failure is classified as runtime error.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TracedFailure*			
Aggregated by	ErrorTracerNeeds.tracedFailure			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
possibleError Reaction	PossibleErrorReaction	*	aggr	Describes a possible error reactions for the transient fault handler.

Table F.52: TransientFault

Class	VariableInAtomicSWCTypeInstanceRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements::InstanceRefsUsage			
Note				
Base	*ARObject*, *AtpInstanceRef*			
Aggregated by	AutosarVariableRef.autosarVariable			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
base	AtomicSwComponentType	0..1	ref	**Stereotypes:** atpDerived **Tags:**xml.sequenceOffset=10
contextDataPrototype (ordered)	ApplicationCompositeElementDataPrototype	*	ref	This is the context in a compositeDataType. **Tags:**xml.sequenceOffset=40
portPrototype	PortPrototype	0..1	ref	This is the port providing the parameter or the entry point to the parameter structure. **Tags:**xml.sequenceOffset=20
rootVariableDataPrototype	VariableDataPrototype	0..1	ref	**Tags:**xml.sequenceOffset=30
targetDataPrototype	DataPrototype	0..1	ref	This is the target of the instance ref. Note that it shall be one of ApplicationCompositeElementDataPrototype of VariableDataPrototype. **Tags:**xml.sequenceOffset=50

Table F.53: VariableInAtomicSWCTypeInstanceRef

Class	VariationPoint			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to express a "structural variation point". The container of the variation point is part of the selected variant if swSyscond evaluates to true and each postBuildVariantCriterion is fulfilled.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
blueprintCondition	DocumentationBlock	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint. Note that variationPoints are not allowed within a blueprintCondition. **Tags:**xml.sequenceOffset=28
desc	MultiLanguageOverviewParagraph	0..1	aggr	This allows to describe shortly the purpose of the variation point. **Tags:**xml.sequenceOffset=20
formalBlueprintGenerator	BlueprintGenerator	0..1	aggr	This represents a description that documents how the variation point shall be resolved when deriving objects from the blueprint by using ARMQL. Note that variationPoints are not allowed within a formalBlueprintGenerator. **Tags:** atp.Status=draft xml.sequenceOffset=30
postBuildVariantCondition	PostBuildVariantCondition	*	aggr	This is the set of post build variant conditions which all shall be fulfilled in order to (postbuild) bind the variation point. **Tags:**xml.sequenceOffset=40

▽

△

Class	VariationPoint			
sdg	Sdg	0..1	aggr	An optional special data group is attached to every variation point. These data can be used by external software systems to attach application specific data. For example, a variant management system might add an identifier, an URL or a specific classifier. **Tags:**xml.sequenceOffset=50
shortLabel	Identifier	0..1	attr	This provides a name to the particular variation point to support the RTE generator. It is necessary for supporting splitable aggregations and if binding time is later than codeGenerationTime, as well as some RTE conditions. It needs to be unique with in the enclosing Identifiables with the same ShortName. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=10
swSyscond	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the Variation Point. Note that the multiplicity is 0..1 in order to support pure postBuild variants. **Tags:**xml.sequenceOffset=30

Table F.54: VariationPoint

G Upstream Mapping

G.1 Introduction

This chapter describes the mapping of the ECU Configuration parameters (M1 model) onto the meta-classes and attributes of the AUTOSAR upstream templates (in this case: Software Component Template).

The relationships between upstream templates and ECU Configuration are described in order to answer typical questions like:

- How shall a supplier use the information in a System Description in order to fulfill the needs defined by the systems engineer?
- How is a tool vendor supposed to generate an ECU Configuration Description out of ECU Extract of System Description?

Please note that the upstream mapping tables contain the following columns:

Column Name	Column Meaning
BSW Module	Name of BSW module
BSW Context	Reference to parameter container
BSW Parameter	Name of the BSW parameter
BSW Type	Type of parameter
BSW Description	Description from the configuration document
Template Description	Class note or attribute note of the M2 model element
M2 Parameter	Name of the upstream template model element
Mapping Rule	Textual description on how to transform between M2 and BSW domains
Mapping Type	One of: **local** no mapping needed since parameter local to BSW **partial** some data can be automatically mapped but not all **full** all data can be automatically mapped
Mapping Status	Indication of life-cycle status of the mapping
ECUC Parameter ID	ID of the parameter in the respective SWS document (may be empty if the mapping is owned by an enumeration literal)

Table G.1: Upstream mapping table columns

G.2 NvM

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockJobPriority		ECUC-INTEGER-PARAM-DEF

▽

△

BSW Description	
Defines the job priority for a NVRAM block (0 = Immediate priority).	
Template Description	
NvBlockNeeds.writingPriority: Requires the priority of writing this block in case of concurrent requests to write other blocks. **NvBlockNeeds.storeEmergency:** Defines whether or not the associated RAM Block shall be implicitly stored in case of ECU failure (e.g. loss of power) by the basic software. If the attribute storeEmergency is set to true the associated RAM Block shall be configured to have immediate priority.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.`writingPriority`, CommonStructure::ServiceNeeds::NvBlockNeeds.`storeEmergency`	
Mapping Rule	**Mapping Type**
It is the integrators job to secure the value-monotonic assignment of writingPriority to NvMBlockJobPriority. This means that the lowest assigned value of writingPriority=MEDIUM shall be greater than highest assigned value of writingPriority=HIGH etc.If NvBlockNeeds.storeEmergency is set to True then NvMBlockJobPriority shall be 0 (Immediate priority). If NvBlockNeeds.storeEmergency is set to False then the value of NvMBlockJobPriority depends on the value of NvBlockNeeds.writingPriority.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00477]

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter	**BSW Type**	
NvMBlockManagementType	ECUC-ENUMERATION-PARAM-DEF	
BSW Description		
Defines the block management type for the NVRAM block.[SWS_NvM_00137]		
Template Description		
Reliability against data loss on the non-volatile medium.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.`reliability`		
Mapping Rule	**Mapping Type**	
if (reliability == errorDetection | noProtection) && nDataSets==0 then NvmBlockManagementType = NVM_BLOCK_NATIVE. if reliability == errorCorrection then NvmBlockManagementType = NVM_BLOCK_REDUNDANT. [constr_1095] applies.	full	
Mapping Status	**ECUC Parameter ID**	
valid	[ECUC_NvM_00062]	

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter	**BSW Type**	
NvMBlockUseAutoValidation	ECUC-BOOLEAN-PARAM-DEF	
BSW Description		
Defines whether the RAM Block shall be auto validated during shutdown phase. true: if auto validation mechanism is used, false: otherwise		
Template Description		

▽

△

If set to true the RAM Block shall be auto validated during shutdown phase.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.useAutoValidationAtShutDown	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00557]

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMBlockUseCRCCompMechanism		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines whether the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job. true: if compare mechanism is used, false: otherwise			
Template Description			
If set to true the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job in order to skip unnecessary NVRAM writings.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.useCRCCompMechanism			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00556]

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMBlockUseCrc		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines CRC usage for the NVRAM block, i.e. memory space for CRC is reserved in RAM and NV memory. true: CRC will be used for this NVRAM block. false: CRC will not be used for this NVRAM block. Note: Configuring CRC for a block with immediate priority is not recommended, since the CRC calculation may extend over more than one NvM main function and this could increase the time of writing the immediate data significantly, thus defeating the purpose of immediate priority.			
Template Description			
Reliability against data loss on the non-volatile medium.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.reliability			
Mapping Rule			**Mapping Type**
reliability == errorCorrection | errorDetection means that NvmBlockUseCrc shall bet set to true, else NvmBlockUseCrc = false			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00036]

BSW Module	BSW Context		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMBlockUseSetRamBlockStatus		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines if NvMSetRamBlockStatusApi shall be used for this block or not. Note: If NvMSetRamBlockStatusApi is disabled this configuration parameter shall be ignored. true: calling of NvMSetRamBlockStatus for this RAM block shall set the status of the RAM block. false: calling of NvMSetRamBlockStatus for this RAM block shall be ignored.			
Template Description			
This attribute defines how the management of the RAM Block status is controlled.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.ramBlockStatusControl			
Mapping Rule			**Mapping Type**
If the value of NvBlockNeeds.ramBlockStatusControl is set to RamBlockStatusControlEnum.api the parameter shall be set to true. If the value of NvBlockNeeds.ramBlockStatusControl is set to RamBlockStatusControlEnum.nvRamManager it shall be set to false.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00552]

BSW Module	BSW Context		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMBlockWriteProt		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines an initial write protection of the NV block true: Initial block write protection is enabled. false: Initial block write protection is disabled.			
Template Description			
true: data of this NVRAM Block are write protected for normal operation (but protection can be disabled) false: no restriction			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.readonly			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00033]

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMCalcRamBlockCrc		ECUC-BOOLEAN-PARAM-DEF
BSW Description		

▽

△

Defines CRC (re)calculation for the permanent RAM block or NVRAM blocks which are configured to use explicit synchronization mechanism. true: CRC will be (re)calculated for this permanent RAM block. false: CRC will not be (re)calculated for this permanent RAM block.		
Template Description		
Defines if CRC (re)calculation for the permanent RAM Block is required.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.calcRamBlockCrc		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_NvM_00119]

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMNvBlockNum		ECUC-INTEGER-PARAM-DEF	
BSW Description			
Defines the number of multiple NV blocks in a contiguous area according to the given block management type. 1-255 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to SWS_NvM_00444. 1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 2 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT			
Template Description			
NvBlockNeeds.nDataSets: Number of data sets to be provided by the NVRAM manager for this block. This is the total number of ROM Blocks and RAM Blocks. **NvBlockNeeds.reliability:** Reliability against data loss on the non-volatile medium.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.nDataSets, CommonStructure::ServiceNeeds::NvBlockNeeds.reliability			
Mapping Rule			**Mapping Type**
if (nDataSets == 0 && reliability ==noProtection | errorDetection) then NvMNvBlockNum = 1. if (nDataSets >0 && reliability ==noProtection | errorDetection) then NvMNvBlockNum = nDataSets.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00480]

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMResistantToChangedSw		ECUC-BOOLEAN-PARAM-DEF
BSW Description		

▽

△

Defines whether a NVRAM block shall be treated resistant to configuration changes or not. If there is no default data available at configuration time then the application shall be responsible for providing the default initialization data. In this case the application has to use NvM_GetErrorStatus()to be able to distinguish between first initialization and corrupted data. true: NVRAM block is resistant to changed software. false: NVRAM block is not resistant to changed software.
Template Description
Defines whether an NVRAM Block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, please refer to the NVRAM specification.
M2 Parameter
CommonStructure::ServiceNeeds::NvBlockNeeds.resistantToChangedSw

Mapping Rule	**Mapping Type**
1:1 Mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00483]

BSW Module	**BSW Context**
NvM	NvM/NvMBlockDescriptor
BSW Parameter	**BSW Type**
NvMRomBlockNum	ECUC-INTEGER-PARAM-DEF
BSW Description	
Defines the number of multiple ROM blocks in a contiguous area according to the given block management type. 0-254 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to SWS_NvM_00444. 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT	
Template Description	
Number of ROM Blocks to be provided by the NVRAM manager for this block. Please note that these multiple ROM Blocks are given in a contiguous area.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.nRomBlocks	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00485]

BSW Module	**BSW Context**
NvM	NvM/NvMBlockDescriptor
BSW Parameter	**BSW Type**
NvMSelectBlockForFirstInitAll	ECUC-BOOLEAN-PARAM-DEF
BSW Description	
Defines whether a block will be processed or not by NvM_FirstInitAll. A block can be configured to be processed even if it doesn't have permanent RAM and/or explicit synchronization. TRUE: block will be processed by NvM_FirstInitAll FALSE: block will not be processed by NvM_FirstInitAll	
Template Description	
If this attribute is set to true the NvM shall process this block in the NvM_FirstInitAll() function.	
M2 Parameter	

▽

△

CommonStructure::ServiceNeeds::NvBlockNeeds.selectBlockForFirstInitAll	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00558]

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMSelectBlockForReadAll		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines whether a NVRAM block shall be processed during NvM_ReadAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_ReadAll false: NVRAM block shall not be processed by NvM_ReadAll			
Template Description			
Defines whether the associated RAM Block shall be implicitly restored during startup by the basic software.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.restoreAtStart			
Mapping Rule			**Mapping Type**
1:1 Mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00117]

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMSelectBlockForWriteAll		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines whether a NVRAM block shall be processed during NvM_WriteAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_WriteAll false: NVRAM block shall not be processed by NvM_WriteAll			
Template Description			
Defines whether or not the associated RAM Block shall be implicitly stored during shutdown by the basic software.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.storeAtShutdown			
Mapping Rule			**Mapping Type**
1:1 Mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00549]

BSW Module	BSW Context		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMStaticBlockIDCheck		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines if the Static Block ID check is enabled. false: Static Block ID check is disabled. true: Static Block ID check is enabled.			
Template Description			
Defines if the Static Block Id check shall be enabled.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.checkStaticBlockId			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00532]

BSW Module	BSW Context		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMWriteBlockOnce		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines write protection after first write. The NVRAM manager sets the write protection bit either after the NV block was written the first time or if the block was already written and it is detected as valid and consistent during a read for it. true: Defines write protection after first write is enabled. false: Defines write protection after first write is disabled.			
Template Description			
Defines write protection after first write: true: This block is prevented from being changed/erased or being replaced with the default ROM data after first initialization by the software-component. false: No such restriction.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.writeOnlyOnce			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_NvM_00072]

BSW Module	BSW Context		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMWriteVerification		ECUC-BOOLEAN-PARAM-DEF	
BSW Description			
Defines if Write Verification is enabled. false: Write verification is disabled. true: Write Verification is enabled.			
Template Description			

▽

△

Defines if Write Verification shall be enabled for this NVRAM Block.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.writeVerification	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_NvM_00534]

G.3 Com

BSW Module	**BSW Context**
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription
BSW Parameter	**BSW Type**
ComFilter	ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description	
This container contains the configuration parameters of the AUTOSAR COM module's Filters. Note: On sender side the container is used to specify the transmission mode conditions.	
Template Description	
Base class for data filters. The type of the filter is specified in attribute dataFilterType. Some of the filter types require additional arguments which are specified as attributes of this class.	
M2 Parameter	
CommonStructure::Filter::DataFilter	
Mapping Rule	**Mapping Type**
Create container on the receiver side if the NonqueuedReceiverComSpec contains a DataFilter. Create Container on the sender side if the TransmissionModeCondition element contains a reference to this signal.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00339]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter
BSW Parameter	**BSW Type**
ComFilterAlgorithm	ECUC-ENUMERATION-PARAM-DEF
BSW Description	
The range of values is specified in the [17] specification, chapter 2.2.2, Reception Filtering.	
Template Description	
This attribute specifies the type of the filter.	
M2 Parameter	
CommonStructure::Filter::DataFilter.dataFilterType	
Mapping Rule	**Mapping Type**
Mapping between DataFilterTypeEnum and ComFilterAlgorithm Enum is necessary.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00146]

BSW Module	BSW Context	
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter	
BSW Parameter		**BSW Type**
ComFilterMask		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Mask for old and new value.		
M2 Parameter		
CommonStructure::Filter::DataFilter.mask		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00235]

BSW Module	BSW Context	
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter	
BSW Parameter		**BSW Type**
ComFilterMax		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the upper boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.max		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00317]

BSW Module	BSW Context	
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter	
BSW Parameter		**BSW Type**
ComFilterMin		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the lower boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.min		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00318]

BSW Module	BSW Context		
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter		
BSW Parameter		**BSW Type**	
ComFilterOffset		ECUC-INTEGER-PARAM-DEF	
BSW Description			
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering. Range = 0..(ComFilterPeriod-1)			
Template Description			
Specifies the initial number of messages to occur before the first message is passed			
M2 Parameter			
CommonStructure::Filter::DataFilter.offset			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00313]

BSW Module	BSW Context		
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter		
BSW Parameter		**BSW Type**	
ComFilterPeriod		ECUC-INTEGER-PARAM-DEF	
BSW Description			
This parameter defines the period of the ComFilterAlgorithm ONE_EVERY_N.			
Template Description			
Specifies number of messages to occur before the message is passed again			
M2 Parameter			
CommonStructure::Filter::DataFilter.period			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00312]

BSW Module	BSW Context		
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription/ComFilter		
BSW Parameter		**BSW Type**	
ComFilterX		ECUC-INTEGER-PARAM-DEF	
BSW Description			
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.			
Template Description			
Value to compare with			
M2 Parameter			
CommonStructure::Filter::DataFilter.x			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00147]

BSW Module	BSW Context		
Com	Com/ComConfig/ComGwMapping/ComGwDestination/ComGwDestinationDescription		
BSW Parameter		**BSW Type**	
ComSignalInitValue		ECUC-STRING-PARAM-DEF	
BSW Description			
Initial value for this signal. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case of UINT8_DYN the initial size shall be 0. In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.			
Template Description			
Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.			
M2 Parameter			
SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal.initValue, SWComponentTemplate::Communication::NonqueuedSenderComSpec.initValue			
Mapping Rule			**Mapping Type**
It is possible to aggregate an initValue at the level of a ComSpec in the SWC Template. in case the System Description doesn't use a complete Software Component Description (VFB View) the init Value is defined in the System Template.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00170]

BSW Module	BSW Context		
Com	Com/ComConfig/ComSignal		
BSW Parameter		**BSW Type**	
ComDataInvalidAction		ECUC-ENUMERATION-PARAM-DEF	
BSW Description			
This parameter defines the action performed upon reception of an invalid signal. Relating to signal groups the action in case if one of the included signals is an invalid signal. If Replace is used the ComSignalInitValue will be used for the replacement.			
Template Description			
InvalidationPolicy: Specifies whether the component can actively invalidate a particular dataElement. If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate. **ISignalPort.handleInvalid:** This attribute defines how invalidation is applied to the ISignals received in the context of this ISignalPort.			
M2 Parameter			
SWComponentTemplate::PortInterface::InvalidationPolicy, SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignalPort.handleInvalid			
Mapping Rule			**Mapping Type**
If strategy HandleInvalidEnum.keep is defined then set ComDataInvalidAction to NOTIFY. If strategy HandleInvalidEnum.replace is defined then set ComDataInvalidAction to REPLACE. In all other cases the ComDataInvalidAction shall not be configured.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00314]

BSW Module	BSW Context		
Com	Com/ComConfig/ComSignal		
BSW Parameter		**BSW Type**	
ComFilter		ECUC-PARAM-CONF-CONTAINER-DEF	
BSW Description			
This container contains the configuration parameters of the AUTOSAR COM module's Filters. Note: On sender side the container is used to specify the transmission mode conditions.			
Template Description			
Base class for data filters. The type of the filter is specified in attribute dataFilterType. Some of the filter types require additional arguments which are specified as attributes of this class.			
M2 Parameter			
CommonStructure::Filter::DataFilter			
Mapping Rule			**Mapping Type**
Create container on the receiver side if the NonqueuedReceiverComSpec contains a DataFilter. Create Container on the sender side if the TransmissionModeCondition element contains a reference to this signal.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00339]

BSW Module	BSW Context		
Com	Com/ComConfig/ComSignal/ComFilter		
BSW Parameter		**BSW Type**	
ComFilterAlgorithm		ECUC-ENUMERATION-PARAM-DEF	
BSW Description			
The range of values is specified in the [17] specification, chapter 2.2.2, Reception Filtering.			
Template Description			
This attribute specifies the type of the filter.			
M2 Parameter			
CommonStructure::Filter::DataFilter.dataFilterType			
Mapping Rule			**Mapping Type**
Mapping between DataFilterTypeEnum and ComFilterAlgorithm Enum is necessary.			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Com_00146]

BSW Module	BSW Context		
Com	Com/ComConfig/ComSignal/ComFilter		
BSW Parameter		**BSW Type**	
ComFilterMask		ECUC-INTEGER-PARAM-DEF	
BSW Description			
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.			
Template Description			
Mask for old and new value.			
M2 Parameter			
CommonStructure::Filter::DataFilter.mask			
Mapping Rule			**Mapping Type**
1:1 mapping			full

▽

△

Mapping Status	ECUC Parameter ID
valid	[ECUC_Com_00235]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignal/ComFilter	
BSW Parameter		**BSW Type**
ComFilterMax		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the upper boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.max		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00317]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignal/ComFilter	
BSW Parameter		**BSW Type**
ComFilterMin		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the lower boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.min		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00318]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignal/ComFilter	
BSW Parameter		**BSW Type**
ComFilterOffset		ECUC-INTEGER-PARAM-DEF
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering. Range = 0..(ComFilterPeriod-1)		
Template Description		
Specifies the initial number of messages to occur before the first message is passed		
M2 Parameter		
CommonStructure::Filter::DataFilter.offset		

▽

△

Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00313]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignal/ComFilter
BSW Parameter	**BSW Type**
ComFilterPeriod	ECUC-INTEGER-PARAM-DEF
BSW Description	
This parameter defines the period of the ComFilterAlgorithm ONE_EVERY_N.	
Template Description	
Specifies number of messages to occur before the message is passed again	
M2 Parameter	
CommonStructure::Filter::DataFilter.period	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00312]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignal/ComFilter
BSW Parameter	**BSW Type**
ComFilterX	ECUC-INTEGER-PARAM-DEF
BSW Description	
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.	
Template Description	
Value to compare with	
M2 Parameter	
CommonStructure::Filter::DataFilter.x	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00147]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignal
BSW Parameter	**BSW Type**
ComSignalDataInvalidValue	ECUC-STRING-PARAM-DEF
BSW Description	

▽

△

Defines the data invalid value of the signal. In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.
Template Description
InvalidationPolicy: Specifies whether the component can actively invalidate a particular dataElement. If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate. **SwDataDefProps.invalidValue:** Optional value to express invalidity of the actual data element.
M2 Parameter
SWComponentTemplate::PortInterface::`InvalidationPolicy`, DataDictionary::DataDefProperties::SwDataDefProps.`invalidValue`

Mapping Rule	**Mapping Type**
ComSignalDataInvalidValue is only derived 1:1 from the SwDataDefProps.invalidValue if the InvalidationPolicy equals keep or replace. In all other cases of InvalidationPolicy the ComSignal DataInvalidValue shall not be configured.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00391]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignal

BSW Parameter	**BSW Type**
ComSignalInitValue	ECUC-STRING-PARAM-DEF

BSW Description
Initial value for this signal. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case of UINT8_DYN the initial size shall be 0. In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.
Template Description
Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.
M2 Parameter
SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal.`initValue`, SWComponent Template::Communication::NonqueuedSenderComSpec.`initValue`

Mapping Rule	**Mapping Type**
It is possible to aggregate an initValue at the level of a ComSpec in the SWC Template. in case the System Description doesn't use a complete Software Component Description (VFB View) the init Value is defined in the System Template.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00170]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignal	
BSW Parameter		**BSW Type**
ComTimeout		ECUC-FLOAT-PARAM-DEF
BSW Description		
Defines the length of the deadline monitoring timeout period in seconds. The period for the first timeout period can be configured separately by ECUC_Com_00183.		
Template Description		
ISignalPort.timeout: • ISignalPort with communicationDirection = in: Optional timeout value in seconds for the reception of the ISignal. The attribute value is used to configure the ComTimeout in the COM module. The RTE ignores this attribute. The timeout can also be specified with the NonqueuedReceiverCom Spec.aliveTimeout attribute. If a full DataMapping exists for the SystemSignal and the value is available in the configured ReceiverComSpec, then the timeout value in the ReceiverComSpec overrides this optional timeout specification during the creation of the Base Ecu Configuration of the COM module. • ISignalPort with communicationDirection = out: Optional timeout value in seconds for the transmission of the ISignal. The attribute value is used to configure the Com Timeout in the COM module. The RTE ignores this attribute. The timeout can also be specified with the enderCom Spec.transmissionAcknowledge.timeout attribute. If a full DataMapping exists for the SystemSignal and the value is available in the configured SenderComSpec, then the timeout value in the SenderComSpec overrides this optional timeout specification during the creation of the Base Ecu Configuration of the COM module. This attribute can be used in the following cases: • legacy signal where the System Description doesn't use a complete Software Component Description (VFB View) and where the DataMapping is missing. • bus monitoring use cases in which the DataMapping is ignored. **TransmissionAcknowledgementRequest.timeout:** Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again. **NonqueuedReceiverComSpec.aliveTimeout:** Specify the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been received according to the specified timing description. If the aliveTimeout attribute is 0 no timeout monitoring shall be performed.		
M2 Parameter		
SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignalPort.timeout, SWComponent Template::Communication::TransmissionAcknowledgementRequest.timeout, SWComponent Template::Communication::NonqueuedReceiverComSpec.aliveTimeout		
Mapping Rule		**Mapping Type**
TX Signals: If a full DataMapping exist for the SystemSignal this information may be available from a configured SenderComSpec.transmissionAcknowledge.timeout that specifies the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been transmitted according to the specified timing description. In this case the timeout value in SenderComSpec overrides the optional timeout specification in the System Template defined on the ISignalPort. RX Signals: If a full DataMapping exist for the SystemSignal this information may be available from a configured NonqueuedReceiverComSpec.aliveTimeout. In this case the timeout value in ReceiverComSpec overrides the optional timeout specification in the System Template defined on the ISignalPort. Please note that the SWS_RTE defines an algorithm to finally set the applicable timeout value.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00263]

<table>
<tr><th>BSW Module</th><th colspan="3">BSW Context</th></tr>
<tr><td>Com</td><td colspan="3">Com/ComConfig/ComSignal</td></tr>
<tr><th colspan="2">BSW Parameter</th><th colspan="2">BSW Type</th></tr>
<tr><td colspan="2">ComTimeoutSubstitutionValue</td><td colspan="2">ECUC-STRING-PARAM-DEF</td></tr>
<tr><th colspan="4">BSW Description</th></tr>
<tr><td colspan="4">The signal substitution value will be used in case of a timeout and ComRxDataTimeoutAction is set to SUBSTITUTE. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00.

In case ofUINT8_DYN the initial size shall be 0.

In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification.

In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification.

In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification.

In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.</td></tr>
<tr><th colspan="4">Template Description</th></tr>
<tr><td colspan="4">Defines and enables the ComTimeoutSubstituition for this ISignal.</td></tr>
<tr><th colspan="4">M2 Parameter</th></tr>
<tr><td colspan="4">SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal.timeoutSubstitutionValue</td></tr>
<tr><th colspan="3">Mapping Rule</th><th>Mapping Type</th></tr>
<tr><td colspan="3">The mapping of ComTimeoutSubstitutionValue depends on the setting in the ISignal.dataType Policy: - ISignal.dataTypePolicy = override or legacy: SystemTemplate::Fibex::FibexCore::Core Communication::ISignal.timeoutSubstitutionValue

- ISignal.dataTypePolicy = networkRepresentationFromComSpec: SWComponent Template::Communication::NonequeuedReceiverComSpec.timeoutSubstitutionValue

- ISignal.dataTypePolicy = transformingISignal this is not supported.</td><td>full</td></tr>
<tr><th colspan="3">Mapping Status</th><th>ECUC Parameter ID</th></tr>
<tr><td colspan="3">valid</td><td>[ECUC_Com_10006]</td></tr>
</table>

<table>
<tr><th>BSW Module</th><th colspan="2">BSW Context</th></tr>
<tr><td>Com</td><td colspan="2">Com/ComConfig/ComSignalGroup</td></tr>
<tr><th colspan="2">BSW Parameter</th><th>BSW Type</th></tr>
<tr><td colspan="2">ComDataInvalidAction</td><td>ECUC-ENUMERATION-PARAM-DEF</td></tr>
<tr><th colspan="3">BSW Description</th></tr>
<tr><td colspan="3">This parameter defines the action performed upon reception of an invalid signal. Relating to signal groups the action in case if one of the included signals is an invalid signal. If Replace is used the ComSignalInitValue will be used for the replacement.</td></tr>
<tr><th colspan="3">Template Description</th></tr>
<tr><td colspan="3">InvalidationPolicy:
Specifies whether the component can actively invalidate a particular dataElement.

If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate.

ISignalPort.handleInvalid:
This attribute defines how invalidation is applied to the ISignals received in the context of this ISignalPort.</td></tr>
<tr><th colspan="3">M2 Parameter</th></tr>
<tr><td colspan="3">SWComponentTemplate::PortInterface::InvalidationPolicy, SystemTemplate::Fibex::FibexCore::Core Communication::ISignalPort.handleInvalid</td></tr>
</table>

▽

△

Mapping Rule	Mapping Type
If strategy HandleInvalidEnum.keep is defined then set ComDataInvalidAction to NOTIFY. If strategy HandleInvalidEnum.replace is defined then set ComDataInvalidAction to REPLACE. In all other cases the ComDataInvalidAction shall not be configured.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00314]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal	
BSW Parameter		**BSW Type**
ComFilter		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
This container contains the configuration parameters of the AUTOSAR COM module's Filters. Note: On sender side the container is used to specify the transmission mode conditions.		
Template Description		
Base class for data filters. The type of the filter is specified in attribute dataFilterType. Some of the filter types require additional arguments which are specified as attributes of this class.		
M2 Parameter		
CommonStructure::Filter::DataFilter		
Mapping Rule		**Mapping Type**
Create container on the receiver side if the NonqueuedReceiverComSpec contains a DataFilter. Create Container on the sender side if the TransmissionModeCondition element contains a reference to this signal.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00339]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter	
BSW Parameter		**BSW Type**
ComFilterAlgorithm		ECUC-ENUMERATION-PARAM-DEF
BSW Description		
The range of values is specified in the [17] specification, chapter 2.2.2, Reception Filtering.		
Template Description		
This attribute specifies the type of the filter.		
M2 Parameter		
CommonStructure::Filter::DataFilter.dataFilterType		
Mapping Rule		**Mapping Type**
Mapping between DataFilterTypeEnum and ComFilterAlgorithm Enum is necessary.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00146]

BSW Module	BSW Context	
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter	
BSW Parameter		**BSW Type**

▽

△

ComFilterMask	ECUC-INTEGER-PARAM-DEF	
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Mask for old and new value.		
M2 Parameter		
CommonStructure::Filter::DataFilter.mask		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00235]

BSW Module	**BSW Context**	
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter	
BSW Parameter	**BSW Type**	
ComFilterMax	ECUC-INTEGER-PARAM-DEF	
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the upper boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.max		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00317]

BSW Module	**BSW Context**	
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter	
BSW Parameter	**BSW Type**	
ComFilterMin	ECUC-INTEGER-PARAM-DEF	
BSW Description		
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.		
Template Description		
Value to specify the lower boundary		
M2 Parameter		
CommonStructure::Filter::DataFilter.min		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00318]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter
BSW Parameter	**BSW Type**
ComFilterOffset	ECUC-INTEGER-PARAM-DEF
BSW Description	
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering. Range = 0..(ComFilterPeriod-1)	
Template Description	
Specifies the initial number of messages to occur before the first message is passed	
M2 Parameter	
CommonStructure::Filter::DataFilter.offset	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00313]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter
BSW Parameter	**BSW Type**
ComFilterPeriod	ECUC-INTEGER-PARAM-DEF
BSW Description	
This parameter defines the period of the ComFilterAlgorithm ONE_EVERY_N.	
Template Description	
Specifies number of messages to occur before the message is passed again	
M2 Parameter	
CommonStructure::Filter::DataFilter.period	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00312]

BSW Module	**BSW Context**
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal/ComFilter
BSW Parameter	**BSW Type**
ComFilterX	ECUC-INTEGER-PARAM-DEF
BSW Description	
The name of this attribute corresponds to the parameter name in the [17] specification of Reception Filtering.	
Template Description	
Value to compare with	
M2 Parameter	
CommonStructure::Filter::DataFilter.x	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00147]

<table>
<tr><th>BSW Module</th><th colspan="2">BSW Context</th></tr>
<tr><td>Com</td><td colspan="2">Com/ComConfig/ComSignalGroup/ComGroupSignal</td></tr>
<tr><th colspan="2">BSW Parameter</th><th>BSW Type</th></tr>
<tr><td colspan="2">ComSignalDataInvalidValue</td><td>ECUC-STRING-PARAM-DEF</td></tr>
<tr><th colspan="3">BSW Description</th></tr>
<tr><td colspan="3">Defines the data invalid value of the signal.
In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.</td></tr>
<tr><th colspan="3">Template Description</th></tr>
<tr><td colspan="3">InvalidationPolicy:
Specifies whether the component can actively invalidate a particular dataElement.
If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate.
SwDataDefProps.invalidValue:
Optional value to express invalidity of the actual data element.</td></tr>
<tr><th colspan="3">M2 Parameter</th></tr>
<tr><td colspan="3">SWComponentTemplate::PortInterface::InvalidationPolicy, DataDictionary::DataDefProperties::SwDataDefProps.invalidValue</td></tr>
<tr><th colspan="2">Mapping Rule</th><th>Mapping Type</th></tr>
<tr><td colspan="2">ComSignalDataInvalidValue is only derived 1:1 from the SwDataDefProps.invalidValue if the InvalidationPolicy equals keep or replace. In all other cases of InvalidationPolicy the ComSignal DataInvalidValue shall not be configured.</td><td>full</td></tr>
<tr><th colspan="2">Mapping Status</th><th>ECUC Parameter ID</th></tr>
<tr><td colspan="2">valid</td><td>[ECUC_Com_00391]</td></tr>
</table>

<table>
<tr><th>BSW Module</th><th colspan="2">BSW Context</th></tr>
<tr><td>Com</td><td colspan="2">Com/ComConfig/ComSignalGroup/ComGroupSignal</td></tr>
<tr><th colspan="2">BSW Parameter</th><th>BSW Type</th></tr>
<tr><td colspan="2">ComSignalInitValue</td><td>ECUC-STRING-PARAM-DEF</td></tr>
<tr><th colspan="3">BSW Description</th></tr>
<tr><td colspan="3">Initial value for this signal. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case of UINT8_DYN the initial size shall be 0.
In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.</td></tr>
<tr><th colspan="3">Template Description</th></tr>
<tr><td colspan="3">Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.</td></tr>
<tr><th colspan="3">M2 Parameter</th></tr>
<tr><td colspan="3">SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal.initValue, SWComponent Template::Communication::NonqueuedSenderComSpec.initValue</td></tr>
</table>

▽

△

Mapping Rule	Mapping Type
It is possible to aggregate an initValue at the level of a ComSpec in the SWC Template. in case the System Description doesn't use a complete Software Component Description (VFB View) the init Value is defined in the System Template.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_00170]

BSW Module	BSW Context
Com	Com/ComConfig/ComSignalGroup/ComGroupSignal
BSW Parameter	**BSW Type**
ComTimeoutSubstitutionValue	ECUC-STRING-PARAM-DEF
BSW Description	
The signal substitution value will be used in case of a timeout and ComRxDataTimeoutAction is set to SUBSTITUTE. In case of UINT8_N the default value is a string of length ComSignalLength with all bytes set to 0x00. In case ofUINT8_DYN the initial size shall be 0. In case the ComSignalType is UINT8, UINT16, UINT32, UINT64, SINT8, SINT16, SINT32, SINT64 the string shall be interpreted as defined in the chapter Integer Type in the AUTOSAR EcuC specification. In case the ComSignalType is FLOAT32, FLOAT64 the string shall be interpreted as defined in the chapter Float Type in the AUTOSAR EcuC specification. In case the ComSignalType is BOOLEAN the string shall be interpreted as defined in the chapter Boolean Type in the AUTOSAR EcuC specification. In case the ComSignal is a UINT8_N, UINT8_DYN the string shall be interpreted as a decimal representation of the characters separated by blanks, e.g. "97 98 100" means a string "abd", where the char "a" is in byte 0(lowest address), "b" is in byte 1, and "d" is in byte 2 and (highest address). For the ComSignalType UINT8_DYN the dynamic length shall be set to the number of configured characters. An empty string "" shall be interpreted as 0-sized dynamic signal.	
Template Description	
Defines and enables the ComTimeoutSubstituition for this ISignal.	
M2 Parameter	
SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal.timeoutSubstitutionValue	
Mapping Rule	**Mapping Type**
The mapping of ComTimeoutSubstitutionValue depends on the setting in the ISignal.dataType Policy: - ISignal.dataTypePolicy = override or legacy: SystemTemplate::Fibex::FibexCore::Core Communication::ISignal.timeoutSubstitutionValue - ISignal.dataTypePolicy = networkRepresentationFromComSpec: SWComponent Template::Communication::NonequeuedReceiverComSpec.timeoutSubstitutionValue - ISignal.dataTypePolicy = transformingISignal this is not supported.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Com_10006]

BSW Module	BSW Context
Com	Com/ComConfig/ComSignalGroup
BSW Parameter	**BSW Type**
ComTimeout	ECUC-FLOAT-PARAM-DEF
BSW Description	
Defines the length of the deadline monitoring timeout period in seconds. The period for the first timeout period can be configured separately by ECUC_Com_00183.	
Template Description	

▽

△

ISignalPort.timeout: • ISignalPort with communicationDirection = in: Optional timeout value in seconds for the reception of the ISignal. The attribute value is used to configure the ComTimeout in the COM module. The RTE ignores this attribute. The timeout can also be specified with the NonqueuedReceiverCom Spec.aliveTimeout attribute. If a full DataMapping exists for the SystemSignal and the value is available in the configured ReceiverComSpec, then the timeout value in the ReceiverComSpec overrides this optional timeout specification during the creation of the Base Ecu Configuration of the COM module. • ISignalPort with communicationDirection = out: Optional timeout value in seconds for the transmission of the ISignal. The attribute value is used to configure the Com Timeout in the COM module. The RTE ignores this attribute. The timeout can also be specified with the enderCom Spec.transmissionAcknowledge.timeout attribute. If a full DataMapping exists for the SystemSignal and the value is available in the configured SenderComSpec, then the timeout value in the SenderComSpec overrides this optional timeout specification during the creation of the Base Ecu Configuration of the COM module. This attribute can be used in the following cases: • legacy signal where the System Description doesn't use a complete Software Component Description (VFB View) and where the DataMapping is missing. • bus monitoring use cases in which the DataMapping is ignored. **TransmissionAcknowledgementRequest.timeout:** Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again. **NonqueuedReceiverComSpec.aliveTimeout:** Specify the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been received according to the specified timing description. If the aliveTimeout attribute is 0 no timeout monitoring shall be performed.		
M2 Parameter		
SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignalPort.timeout, SWComponent Template::Communication::TransmissionAcknowledgementRequest.timeout, SWComponent Template::Communication::NonqueuedReceiverComSpec.aliveTimeout		
Mapping Rule		**Mapping Type**
TX Signals: If a full DataMapping exist for the SystemSignal this information may be available from a configured SenderComSpec.transmissionAcknowledge.timeout that specifies the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been transmitted according to the specified timing description. In this case the timeout value in SenderComSpec overrides the optional timeout specification in the System Template defined on the ISignalPort. RX Signals: If a full DataMapping exist for the SystemSignal this information may be available from a configured NonqueuedReceiverComSpec.aliveTimeout. In this case the timeout value in ReceiverComSpec overrides the optional timeout specification in the System Template defined on the ISignalPort. Please note that the SWS_RTE defines an algorithm to finally set the applicable timeout value.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Com_00263]

G.4 WdgM

BSW Module	**BSW Context**	
WdgM	WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams	
BSW Parameter		**BSW Type**
WdgMFailedAliveSupervisionRefCycleTol		ECUC-INTEGER-PARAM-DEF
BSW Description		

▽

△

This parameter shall contain the acceptable amount of reference cycles with incorrect/failed alive supervisions for this Supervised Entity.		
Template Description		
Number of consecutive failed alive cycles for this SupervisedEntity which shall be tolerated until the supervision status of the SupervisedEntity is set to WDGM_ALIVE_EXPIRED (see SWS WdgM for more details). Note that this value has to be recalculated with respect to the WdgM's own cycle time for ECU configuration.		
M2 Parameter		
CommonStructure::ServiceNeeds::SupervisedEntityNeeds.toleratedFailedCycles		
Mapping Rule		**Mapping Type**
1:1		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_WdgM_-00327]

G.5 Dcm

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsd	
BSW Parameter		**BSW Type**
DcmDsdServiceRequestManufacturerNotification		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
The name of this container is used to define the name of the R-Port through which the DCM accesses the interface Service RequestNotification. The R-Port is named ServiceRequestManufacturerNotification_{Name} where {Name} is the name of the container DcmDsdServiceRequestManufacturerNotification. The lowerMultiplicity is 0: If container DcmDsdServiceRequestManufacturerNotification does not exist the Indication API is not available.		
Template Description		
This represents the ability to define whether the usage of PortInterface ServiceRequestNotification has the characteristics of being initiated by a manufacturer or by a supplier.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType		
Mapping Rule		**Mapping Type**
If DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to Diagnostic ServiceRequestCallbackTypeEnum.requestCallbackTypeManufacturer then DcmDsdService RequestManufacturerNotification shall exist and the value of DcmDsdServiceRequestManufacturer Notification.shortName shall be taken from the SwcServiceDependency.shortName that aggregates the DiagnosticCommunicationManagerNeeds.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00681]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsd	
BSW Parameter		**BSW Type**
DcmDsdServiceRequestSupplierNotification		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		

▽

△

The name of this container is used to define the name of the R-Port through which the DCM accesses the interface Service RequestNotification. The R-Port is named ServiceRequestSupplierNotification_<SWC> where <SWC> is the name of the container DcmDsdServiceRequestSupplierNotification. The lowerMultiplicity is 0: If the container DcmDsdRequestSupplierNotification does not exist the Indication API is not available.	
Template Description	
This represents the ability to define whether the usage of PortInterface ServiceRequestNotification has the characteristics of being initiated by a manufacturer or by a supplier.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticCommunicationManagerNeeds.`serviceRequestCallbackType`	
Mapping Rule	**Mapping Type**
If DiagnosticCommunicationManagerNeeds.serviceRequestCallbackType is set to Diagnostic ServiceRequestCallbackTypeEnum.requestCallbackTypeSupplier then DcmDsdServiceRequest SupplierNotification shall exist and the value of DcmDsdServiceRequestSupplierNotification.short Name shall be taken from the SwcServiceDependency.shortName that aggregates the Diagnostic CommunicationManagerNeeds.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00816]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData	
BSW Parameter		**BSW Type**
DcmDspDataResetToDefaultFnc		ECUC-FUNCTION-NAME-DEF
BSW Description		
Function name to request to application to reset an IOControl to default value. (ResetToDefault-function). This parameter is related to the interface Xxx_ResetToDefault.		
Template Description		
DiagnosticIoControlNeeds.resetToDefaultSupported: This represents a flag for the existence of the ResetToDefault operation in the service interface. **DiagnosticServiceSwMapping.mappedBswServiceDependency:** This is supposed to represent a reference to a BswServiceDependency. the latter is not derived from Referrable and therefore this detour needs to be implemented to still let BswServiceDependency become the target of a reference.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.`resetToDefaultSupported`, DiagnosticExtract::Diagnostic Mapping::ServiceMapping::DiagnosticServiceSwMapping.`mappedBswServiceDependency`		
Mapping Rule		**Mapping Type**
It could be possible to get the FNC name via BswServiceDependency		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00673]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData	
BSW Parameter		**BSW Type**
DcmDspDataShortTermAdjustmentFnc		ECUC-FUNCTION-NAME-DEF
BSW Description		
Function name to request to application to adjust the IO signal. (ShortTermAdjustment-function). This parameter is related to the interface Xxx_ShortTermAdjustment.		
Template Description		

▽

△

DiagnosticIoControlNeeds.shortTermAdjustmentSupported: This attribute determines, if the referenced port supports temporarily setting of I/O value to a specific value provided by the diagnostic tester. **DiagnosticServiceSwMapping.mappedBswServiceDependency:** This is supposed to represent a reference to a BswServiceDependency. the latter is not derived from Referrable and therefore this detour needs to be implemented to still let BswServiceDependency become the target of a reference.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.shortTermAdjustmentSupported, Diagnostic Extract::DiagnosticMapping::ServiceMapping::DiagnosticServiceSwMapping.mappedBswServiceDependency		
Mapping Rule		**Mapping Type**
It could be possible to get the FNC name via BswServiceDependency		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00675]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
BOOLEAN		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is boolean.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition. baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = BOOLEAN baseTypeSize = 1 maxNumberOfElements shall not exist array SizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		
Template Description		

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. DiagnosticValueNeeds.fixedLength: This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
FLOAT_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float array.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
SINT16		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is sint16.		

▽

△

Template Description	
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = 2C baseTypeSize = 16 maxNumberOfElements shall not exist arraySize Semantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType
BSW Parameter	**BSW Type**
SINT16_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is sint16 array.	
Template Description	
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = 2C baseTypeSize = 16 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType		
BSW Parameter		**BSW Type**	
SINT32		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Type of the data is sint32.			
Template Description			
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.			
M2 Parameter			
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength			
Mapping Rule			**Mapping Type**
baseTypeEncoding = 2C baseTypeSize = 32 maxNumberOfElements shall not exist arraySize Semantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.			full
Mapping Status			**ECUC Parameter ID**
valid			

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType		
BSW Parameter		**BSW Type**	
SINT32_N		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Type of the data is sint32 array.			
Template Description			
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.			
M2 Parameter			
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength			
Mapping Rule			**Mapping Type**

▽

baseTypeEncoding = 2C baseTypeSize = 32 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType
BSW Parameter	**BSW Type**
SINT8	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is sint8.	
Template Description	
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = 2C baseTypeSize = 8 maxNumberOfElements shall not exist arraySize Semantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType
BSW Parameter	**BSW Type**
SINT8_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is sint8 array.	
Template Description	

△

BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **BaseTypeDirectDefinition.baseTypeEncoding:** This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = 2C baseTypeSize = 8 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT16		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint16.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, UTF-16 baseTypeSize = 16 maxNumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT16_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint16 array.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, UTF-16 baseTypeSize = 16 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT32		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint32.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**

▽

△

baseTypeEncoding = NONE, UTF-32 baseTypeSize = 32 maxNumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT32_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint32 array.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, UTF-32 baseTypeSize = 32 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT8		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint8.		
Template Description		

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType

BSW Parameter	**BSW Type**
UINT8_DYN	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is uint8 array with dynamic length.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01002) arraySizeSemantics exists and is set to ArraySizeSemanticsEnum.variableSize (cf. TPS_DEXT_01002) Derivation from DiagnosticValueNeeds.fixedLength=0 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataType	
BSW Parameter		**BSW Type**
UINT8_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is uint8 array.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticDataElement.arraySizeSemantics:** This attribute controls the meaning of the value of the array size. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.arraySizeSemantics, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 maxNumberOfElements exists and value is greater than 0 (cf. TPS_DEXT_01001) arraySizeSemantics either does not exist or exists and is set to ArraySizeSemanticsEnum.fixedSize (cf. TPS_DEXT_01001) Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData	
BSW Parameter		**BSW Type**
DcmDspDataUsePort		ECUC-ENUMERATION-PARAM-DEF
BSW Description		
Defines which interface shall be used to access the data.		
Template Description		
This attribute controls whether interaction requires the software-component to react synchronously on a request or whether it processes the request in background but still the DCM has to issue the call again to eventually obtain the result of the request.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticValueNeeds.processingStyle		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00713]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort
BSW Parameter	**BSW Type**
USE_DATA_ASYNCH_CLIENT_SERVER	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
The DCM will access the Data using an R-Port requiring a asynchronous ClientServertInterface DataServices_{Data}. The R-Port is named DataServices_{Data} where {Data} is the name of the container DcmDspData.	
Template Description	
The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleAsynchronous	
Mapping Rule	**Mapping Type**
DiagnosticServiceSwMapping is having a SwcServiceDependency and ServiceNeeds::Diagnostic ProcessingStyleEnum is equal to processingStyleAsynchronous	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort
BSW Parameter	**BSW Type**
USE_DATA_ASYNCH_CLIENT_SERVER_ERROR	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
The Dcm will access the Data using an R-Port requiring a asynchronous ClientServertInterface DataServices_{Data}. The parameter ErrorCode can be returned to allow the application to trigger a negative response during the operation. The R-Port is named DataServices_{Data} where {Data} is the name of the container DcmDspData.	
Template Description	
The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request or handle error code.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleAsynchronousWithError	
Mapping Rule	**Mapping Type**
DiagnosticServiceSwMapping is having a SwcServiceDependency and ServiceNeeds::Diagnostic ProcessingStyleEnum is equal to processingStyleAsynchronousWithError	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort
BSW Parameter	**BSW Type**
USE_DATA_SYNCH_CLIENT_SERVER	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
The DCM will access the Data using an R-Port requiring a synchronous ClientServertInterface DataServices_{Data}. The R-Port is named DataServices_{Data} where {Data} is the name of the container DcmDspData.	
Template Description	
The software-component is supposed to react synchronously on the request.	
M2 Parameter	

▽

△

CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleSynchronous	
Mapping Rule	**Mapping Type**
DiagnosticServiceSwMapping is having a SwcServiceDependency and ServiceNeeds::Diagnostic ProcessingStyleEnum is equal to processingStyleSynchronous	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDiagnosisScaling/DcmDspAlternativeData Interface
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a VariableDataPrototype in a DataInterface. The CompuMethod of the data type of the referenced VariableDataPrototype will be applied to the data type of the Variable DataPrototype in the interface used by the Dcm.	
Template Description	
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.	
M2 Parameter	
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00995]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDiagnosisScaling/DcmDspAlternativeData Interface
BSW Parameter	**BSW Type**
DcmPortInterfaceMapping	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Optional reference to PortInterfaceMapping which defines the mapping rules. The PortInterfaceMapping is used to get the DataPrototypeMapping that describes a conversion between the data prototype referenced by DcmDataElement and the data prototype referenced from DcmDspExternalSRDataElementClass.	
Template Description	
Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).	
M2 Parameter	
SWComponentTemplate::PortInterface::PortInterfaceMapping	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00996]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDiagnosisScaling/DcmDspAlternativeDataType	
BSW Parameter		**BSW Type**
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidControl	
BSW Parameter		**BSW Type**
DcmDspDidFreezeCurrentState		ECUC-BOOLEAN-PARAM-DEF
BSW Description		
This indicates the presence of "FreezeCurrentState".		
Template Description		
DiagnosticIOControl.freezeCurrentState: Setting this attribute to true represents the ability of the Dcm to execute a freezeCurrentState. **DiagnosticIoControlNeeds.freezeCurrentStateSupported:** This attribute determines, if the referenced port supports temporary freezing of I/O value.		
M2 Parameter		
DiagnosticExtract::Dcm::DiagnosticService::IOControl::DiagnosticIOControl.freezeCurrentState, CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.freezeCurrentStateSupported		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00624]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidControl	
BSW Parameter		**BSW Type**
DcmDspDidResetToDefault		ECUC-BOOLEAN-PARAM-DEF
BSW Description		
This indicates the presence of "ResetToDefault".		
Template Description		

▽

△

DiagnosticIOControl.resetToDefault: Setting this attribute to true represents the ability of the Dcm to execute a resetToDefault. **DiagnosticIoControlNeeds.resetToDefaultSupported:** This represents a flag for the existence of the ResetToDefault operation in the service interface.
M2 Parameter
DiagnosticExtract::Dcm::DiagnosticService::IOControl::DiagnosticIOControl.resetToDefault, CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.resetToDefaultSupported

Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00623]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidControl

BSW Parameter	**BSW Type**
DcmDspDidShortTermAdjustment	ECUC-BOOLEAN-PARAM-DEF

BSW Description
This indicates the presence of "ShortTermAdjustment".
Template Description
DiagnosticIOControl.shortTermAdjustment: Setting this attribute to true represents the ability of the Dcm to execute a shortTermAdjustment. **DiagnosticIoControlNeeds.shortTermAdjustmentSupported:** This attribute determines, if the referenced port supports temporarily setting of I/O value to a specific value provided by the diagnostic tester.
M2 Parameter
DiagnosticExtract::Dcm::DiagnosticService::IOControl::DiagnosticIOControl.shortTermAdjustment, CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.shortTermAdjustmentSupported

Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00625]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspMemoryTransfer

BSW Parameter	**BSW Type**
DcmDspMemoryTransferUsePort	ECUC-BOOLEAN-PARAM-DEF

BSW Description
If this parameter is set to true, the Dcm uses a port requiring a PortInterface UploadDownload. If the parameter is false, the DCM uses the according C-API callouts.
Template Description
This meta-class represents the ability to specify needs regarding upload and download by means of diagnostic services.
M2 Parameter
CommonStructure::ServiceNeeds::DiagnosticUploadDownloadNeeds

Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01133]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData	
BSW Parameter		**BSW Type**
DcmDspPidDataByteSize		ECUC-INTEGER-PARAM-DEF
BSW Description		
Defines the array length in bytes or the the maximum array length for variable datalengths.		
Template Description		
BaseTypeDirectDefinition.baseTypeSize: Describes the length of the data type specified in the container in bits. **DiagnosticDataElement.maxNumberOfElements:** The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of how many elements the array can take.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, DiagnosticExtract::CommonDiagnostics::DiagnosticDataElement.maxNumberOfElements		
Mapping Rule		**Mapping Type**
S/R via array: DcmDspPidDataByteSize= maxNumberOfElements * (baseTypeSize / 8) C/S of FNC callback: DcmDspPidDataByteSize= maxNumberOfElements Note: 8 is the baseType Size of UINT8		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_01108]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData/DcmDspPidService01/DcmDspDiagnosisScaling/DcmDspAlternativeDataInterface	
BSW Parameter		**BSW Type**
DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a VariableDataPrototoype in a DataInterface. The CompuMethod of the data type of the referenced VariableDataPrototype will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.		
M2 Parameter		
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00995]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData/DcmDspPidService01/DcmDspDiagnosisScaling/DcmDspAlternativeDataInterface	
BSW Parameter		**BSW Type**
DcmPortInterfaceMapping		ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
Optional reference to PortInterfaceMapping which defines the mapping rules. The PortInterfaceMapping is used to get the DataPrototypeMapping that describes a conversion between the data prototype referenced by DcmDataElement and the data prototype referenced from DcmDspExternalSRDataElementClass.	
Template Description	
Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).	
M2 Parameter	
SWComponentTemplate::PortInterface::PortInterfaceMapping	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00996]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspPid/DcmDspPidData/DcmDspPidService01/DcmDspDiagnosisScaling/DcmDspAlternativeDataType	
BSW Parameter		**BSW Type**
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine	
BSW Parameter		**BSW Type**
DcmDspRequestRoutineResults		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
Provides the configuration of RequestResult subservice for RoutineControl service. Existence indicates that the RequestRoutineResults in the RoutineControl is supported.		
Template Description		
DiagnosticRoutine.requestResult: This represents the ability to request the result of a running routine. **DiagnosticRoutineNeeds.diagRoutineType:** This denotes the type of diagnostic routine which is implemented by the referenced server port.		
M2 Parameter		

▽

△

DiagnosticExtract::CommonDiagnostics::DiagnosticRoutine.requestResult, CommonStructure::ServiceNeeds::DiagnosticRoutineNeeds.diagRoutineType	
Mapping Rule	**Mapping Type**
1:1 mapping for DiagnosticRoutine.requestResult OR DiagnosticRoutineNeeds.diagRoutineTyoe == asynchronous	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01023]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData	
BSW Parameter		**BSW Type**
DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.		
Template Description		
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.		
M2 Parameter		
SWComponentTemplate::PortInterface::ArgumentDataPrototype		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_01056]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType	
BSW Parameter		**BSW Type**
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float array.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignalCompositePool/DcmDspRequestRoutineResultsInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData		
BSW Parameter		**BSW Type**	
DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.			
Template Description			
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
M2 Parameter			
SWComponentTemplate::PortInterface::ArgumentDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_01056]

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignalCompositePool/DcmDspRequestRoutineResultsInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType		
BSW Parameter		**BSW Type**	
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.			
Template Description			
A primitive data type defines a set of allowed values.			
M2 Parameter			
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignalCompositePool/DcmDspRequestRoutineResultsInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		

▽

△

Type of the data is float.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsIn/DcmDspRequestRoutineResultsInSignalCompositePool/DcmDspRequestRoutineResultsInSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float array.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData		
BSW Parameter		**BSW Type**	
DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.			
Template Description			
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
M2 Parameter			
SWComponentTemplate::PortInterface::ArgumentDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_01056]

BSW Module	BSW Context		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType		
BSW Parameter		**BSW Type**	
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.			
Template Description			
A primitive data type defines a set of allowed values.			
M2 Parameter			
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		

▽

△

Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float array.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignalCompositePool/DcmDspRequestRoutineResultsOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**

▽

△

DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.			
Template Description			
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
M2 Parameter			
SWComponentTemplate::PortInterface::`ArgumentDataPrototype`			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_01056]

BSW Module	**BSW Context**		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignalCompositePool/DcmDspRequestRoutineResultsOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType		
BSW Parameter		**BSW Type**	
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.			
Template Description			
A primitive data type defines a set of allowed values.			
M2 Parameter			
SWComponentTemplate::Datatype::Datatypes::`ApplicationPrimitiveDataType`			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_00998]

BSW Module	**BSW Context**		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignalCompositePool/DcmDspRequestRoutineResultsOutSignal/DcmDspRoutineSignalType		
BSW Parameter		**BSW Type**	
FLOAT		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Type of the data is float.			
Template Description			

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspRequestRoutineResults/DcmDspRequestRoutineResultsOut/DcmDspRequestRoutineResultsOutSignalCompositePool/DcmDspRequestRoutineResultsOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float array.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine
BSW Parameter	**BSW Type**
DcmDspStartRoutine	ECUC-PARAM-CONF-CONTAINER-DEF

▽

△

BSW Description	
Provides the configuration of Start subservice for RoutineControl service.	
Template Description	
DiagnosticRoutine.start: This represents the ability to start a routine **DiagnosticRoutineNeeds:** Specifies the general needs on the configuration of the Diagnostic Communication Manager (Dcm) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.	
M2 Parameter	
DiagnosticExtract::CommonDiagnostics::DiagnosticRoutine.start, CommonStructure::ServiceNeeds::DiagnosticRoutineNeeds	
Mapping Rule	**Mapping Type**
A routine always comes with a start routine, independently of whether the execution is done synchronously or asynchronously.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01021]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.	

▽

△

Template Description	
A primitive data type defines a set of allowed values.	
M2 Parameter	
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float array.		
Template Description		

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	**BSW Context**		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignalCompositePool/DcmDspStartRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData		
BSW Parameter		**BSW Type**	
DcmDataElement		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.			
Template Description			
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
M2 Parameter			
SWComponentTemplate::PortInterface::ArgumentDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dcm_01056]

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignalCompositePool/DcmDspStartRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType	
BSW Parameter		**BSW Type**
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		

▽

△

Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.
Template Description
A primitive data type defines a set of allowed values.
M2 Parameter
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType

Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignalCompositePool/DcmDspStartRoutineInSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineIn/DcmDspStartRoutineInSignalCompositePool/DcmDspStartRoutineInSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float array.
Template Description

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.

M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.	
Template Description	

▽

△

A primitive data type defines a set of allowed values.	
M2 Parameter	
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float array.	
Template Description	

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignalCompositePool/DcmDspStartRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignalCompositePool/DcmDspStartRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	

▽

△

Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00998]

BSW Module	**BSW Context**		
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignalCompositePool/DcmDspStartRoutineOutSignal/DcmDspRoutineSignalType		
BSW Parameter		**BSW Type**	
FLOAT		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Type of the data is float.			
Template Description			
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.			
M2 Parameter			
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength			
Mapping Rule			**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.			full
Mapping Status			**ECUC Parameter ID**
valid			

BSW Module	**BSW Context**	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStartRoutine/DcmDspStartRoutineOut/DcmDspStartRoutineOutSignalCompositePool/DcmDspStartRoutineOutSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float array.		

▽

△

Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine
BSW Parameter	**BSW Type**
DcmDspStopRoutine	ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description	
Provides the configuration of Stop subservice for RoutineControl service. Existence indicates that the StopRoutine in the RoutineControl is supported.	
Template Description	
DiagnosticRoutine.stop: This represents the ability to stop a running routine. **DiagnosticRoutineNeeds.diagRoutineType:** This denotes the type of diagnostic routine which is implemented by the referenced server port.	
M2 Parameter	
DiagnosticExtract::CommonDiagnostics::DiagnosticRoutine.stop, CommonStructure::ServiceNeeds::DiagnosticRoutineNeeds.diagRoutineType	
Mapping Rule	**Mapping Type**
1:1 mapping for DiagnosticRoutine.stop OR DiagnosticRoutineNeeds.diagRoutineType == asynchronous	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01022]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	

▽

△

Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.	
Template Description	
A primitive data type defines a set of allowed values.	
M2 Parameter	
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float.	
Template Description	

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignal/DcmDspRoutineSignalType

BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float array.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignalCompositePool/DcmDspStopRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData

BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignalCompositePool/DcmDspStopRoutineInSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType	
BSW Parameter		**BSW Type**
DcmApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dcm_00998]

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignalCompositePool/DcmDspStopRoutineInSignal/DcmDspRoutineSignalType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		
Template Description		

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineIn/DcmDspStopRoutineInSignalCompositePool/DcmDspStopRoutineInSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float array.	
Template Description	
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.	
M2 Parameter	
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength	
Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData
BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.	
Template Description	
A primitive data type defines a set of allowed values.	
M2 Parameter	
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float.	
Template Description	

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignal/DcmDspRoutineSignalType

BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float array.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignalCompositePool/DcmDspStopRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeArgumentData

BSW Parameter	**BSW Type**
DcmDataElement	ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ArgumentDataPrototype. The CompuMethod of the data type of the referenced ArgumentDataPrototype will be applied to the data type of the ArgumentDataPrototype in the interface used by the Dcm.	
Template Description	
An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.	
M2 Parameter	
SWComponentTemplate::PortInterface::ArgumentDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_01056]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignalCompositePool/DcmDspStopRoutineOutSignal/DcmDspArgumentScaling/DcmDspAlternativeDataType
BSW Parameter	**BSW Type**
DcmApplicationDataType	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dcm.	
Template Description	
A primitive data type defines a set of allowed values.	
M2 Parameter	
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dcm_00998]

BSW Module	BSW Context
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignalCompositePool/DcmDspStopRoutineOutSignal/DcmDspRoutineSignalType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float.	
Template Description	

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspRoutine/DcmDspStopRoutine/DcmDspStopRoutineOut/DcmDspStopRoutineOutSignalCompositePool/DcmDspStopRoutineOutSignal/DcmDspRoutineSignalType

BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float array.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspSecurity

BSW Parameter	**BSW Type**
DcmDspSecurityRow	ECUC-PARAM-CONF-CONTAINER-DEF

▽

△

<table>
<tr><th colspan="2">BSW Description</th></tr>
<tr><td colspan="2">Definition of a single Row of configuration for security level configuration (per security level) The name of this container is used to define the name of the R-Port through which the DCM accesses the interface SecurityAccess_{SecurityLevel}. The R-Port is named SecurityAccess_{SecurityLevel} where {SecurityLevel} is the name of the container DcmDspSecurityRow. If there is no reference, no check of security level shall be done.</td></tr>
<tr><th colspan="2">Template Description</th></tr>
<tr><td colspan="2">This meta-class represents the needs of a software-component to verify the access to security level via diagnostic services.</td></tr>
<tr><th colspan="2">M2 Parameter</th></tr>
<tr><td colspan="2">CommonStructure::ServiceNeeds::DiagnosticsCommunicationSecurityNeeds</td></tr>
<tr><th>Mapping Rule</th><th>Mapping Type</th></tr>
<tr><td>1:1 mapping</td><td>full</td></tr>
<tr><th>Mapping Status</th><th>ECUC Parameter ID</th></tr>
<tr><td>valid</td><td>[ECUC_Dcm_00759]</td></tr>
</table>

G.6 Dem

<table>
<tr><th>BSW Module</th><th colspan="2">BSW Context</th></tr>
<tr><td>Dem</td><td colspan="2">Dem/DemConfigSet</td></tr>
<tr><th colspan="2">BSW Parameter</th><th>BSW Type</th></tr>
<tr><td colspan="2">DemDebounceCounterBasedClass</td><td>ECUC-PARAM-CONF-CONTAINER-DEF</td></tr>
<tr><th colspan="3">BSW Description</th></tr>
<tr><td colspan="3">This container contains the configuration of Debounce Counter Based Class</td></tr>
<tr><th colspan="3">Template Description</th></tr>
<tr><td colspan="3">This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor.

This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounceCounterBased.</td></tr>
<tr><th colspan="3">M2 Parameter</th></tr>
<tr><td colspan="3">CommonStructure::ServiceNeeds::DiagEventDebounceCounterBased</td></tr>
<tr><th colspan="2">Mapping Rule</th><th>Mapping Type</th></tr>
<tr><td colspan="2">1:1 mapping</td><td>full</td></tr>
<tr><th colspan="2">Mapping Status</th><th>ECUC Parameter ID</th></tr>
<tr><td colspan="2">valid</td><td>[ECUC_Dem_00881]</td></tr>
</table>

<table>
<tr><th>BSW Module</th><th colspan="2">BSW Context</th></tr>
<tr><td>Dem</td><td colspan="2">Dem/DemConfigSet/DemDebounceCounterBasedClass</td></tr>
<tr><th colspan="2">BSW Parameter</th><th>BSW Type</th></tr>
<tr><td colspan="2">DemDebounceCounterJumpDown</td><td>ECUC-BOOLEAN-PARAM-DEF</td></tr>
<tr><th colspan="3">BSW Description</th></tr>
<tr><td colspan="3">Switch for the activation of Jump-Down.

true: Jump-Down activated false: Jump-Down deactivated</td></tr>
<tr><th colspan="3">Template Description</th></tr>
<tr><td colspan="3">This value activates or deactivates the counter jump-down behavior.</td></tr>
</table>

▽

△

M2 Parameter	
CommonStructure::ServiceNeeds::DiagEventDebounceCounterBased.counterJumpDown	
Mapping Rule	**Mapping Type**
Shall be taken from DiagnosticExtract::DiagnosticCommonProps.debounceAlgorithmProps.debounceAlgorithm.counterJumpDown. Applicable if DiagnosticExtract::DiagnosticCommonProps.debounceAlgorithmProps.debounceAlgorithm is modeled by means of a DiagEventDebounceCounterBased.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dem_00685]

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemDebounceCounterBasedClass	
BSW Parameter		**BSW Type**
DemDebounceCounterPassedThreshold		ECUC-INTEGER-PARAM-DEF
BSW Description		
Defines the value of the internal debounce counter, which indicates the passed status.		
Template Description		
This value defines the event-specific limit that indicates the "passed" counter status.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagEventDebounceCounterBased.counterPassedThreshold		
Mapping Rule		**Mapping Type**
Shall be taken from DiagnosticExtract::DiagnosticCommonProps.debounceAlgorithmProps.debounceAlgorithm.counterPassedThreshold. Applicable if DiagnosticExtract::DiagnosticCommonProps.debounceAlgorithmProps.debounceAlgorithm is modeled by means of a DiagEventDebounceCounterBased.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dem_00636]

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter	
BSW Parameter		**BSW Type**
DemDebounceAlgorithmClass		ECUC-CHOICE-CONTAINER-DEF
BSW Description		
Debounce algorithm class: counter based, time based, or monitor internal.		
Template Description		
This class represents the ability to specify the pre-debounce algorithm which is selected and/or required by the particular monitor. This class inherits from Identifiable in order to allow further documentation of the expected or implemented debouncing and to use the category for the identification of the expected / implemented debouncing.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagEventDebounceAlgorithm		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dem_00604]

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter/DemDebounceAlgorithmClass	
BSW Parameter		**BSW Type**
DemDebounceCounterBased		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
This container contains the configuration (parameters) for counter based debouncing.		
Template Description		
This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounceCounterBased.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagEventDebounceCounterBased		
Mapping Rule		**Mapping Type**
There are two ways to derive the existence of DemDebounceCounterBased: 1. DiagEventNeeds,diagEventDebounceAlgoritm exists and is modeled as a DiagEventDebounce CounterBased. 2. DiagnosticContributionSet.commonProperties.debounceAlgorithmProps.debounceAlgorithm exists and is modeled as a DiagEventDebounceCounterBased If both alternatives exist at the same time then the definition ot DiagnosticContributionSet.common Properties.debounceAlgorithmProps.debounceAlgorithm shall be handled with priority.		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dem_00711]

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter/DemDebounceAlgorithmClass	
BSW Parameter		**BSW Type**
DemDebounceMonitorInternal		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
This container contains the configuration (parameters) for monitor internal debouncing.		
Template Description		
"This meta-class represents the ability to indicate that no Dem pre-debounce algorithm shall be used for this diagnostic monitor. The SWC might implement an internal debouncing algorithm and report qualified (debounced) results to the Dem/ DM.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagEventDebounceMonitorInternal		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dem_00712]

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter/DemDebounceAlgorithmClass	
BSW Parameter		**BSW Type**
DemDebounceTimeBased		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
This container contains the configuration (parameters) for time based debouncing.		

▽

△

Template Description	
This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to DemDebounceTimeBase.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagEventDebounceTimeBased	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dem_00713]

BSW Module	BSW Context
Dem	Dem/DemConfigSet/DemEventParameter
BSW Parameter	**BSW Type**
DemFFPrestorageInNvm	ECUC-BOOLEAN-PARAM-DEF
BSW Description	
If the event uses a pre-stored freeze-frame this attribute indicates if the event requires the pre-stored data to be stored in non-volatile memory. TRUE = store the pre-stored data in non-volatile memory, FALSE = pre-stored data is not stored in non-volatile memory.	
Template Description	
DiagnosticEventNeeds.prestoredFreezeframeStoredInNvm: If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestoredFreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm). **DiagnosticEvent.prestoredFreezeframeStoredInNvm:** If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestoredFreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm)	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticEventNeeds.prestoredFreezeframeStoredInNvm, DiagnosticExtract::Dem::DiagnosticEvent::DiagnosticEvent.prestoredFreezeframeStoredInNvm	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dem_00948]

BSW Module	BSW Context
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalCSDataElementClass/DemDataElementDataType
BSW Parameter	**BSW Type**
FLOAT	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Type of the data is float.	
Template Description	

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.`baseTypeEncoding`, AsamHdo::BaseTypes::BaseTypeDirectDefinition.`baseTypeSize`, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.`fixedLength`

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalCSDataElementClass/DemDataElementDataType

BSW Parameter	**BSW Type**
FLOAT_N	ECUC-ENUMERATION-LITERAL-DEF

BSW Description
Type of the data is float array.
Template Description
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.
M2 Parameter
AsamHdo::BaseTypes::BaseTypeDirectDefinition.`baseTypeEncoding`, AsamHdo::BaseTypes::BaseTypeDirectDefinition.`baseTypeSize`, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.`fixedLength`

Mapping Rule	**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalCSDataElementClass

BSW Parameter	**BSW Type**
DemDataElementProvideMonitorData	ECUC-BOOLEAN-PARAM-DEF

BSW Description

▽

△

If the parameter is set to True, the generated function call to retrieve the data element has the monitorData0 as additional first parameter.	
Template Description	
This attribute defines whether additional monitor data shall be added to the reporting of events.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticEventNeeds.usesMonitorData	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dem_00951]

BSW Module	**BSW Context**	
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDataElementDataType	
BSW Parameter		**BSW Type**
FLOAT		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float.		
Template Description		
BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.		
M2 Parameter		
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength		
Mapping Rule		**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	**BSW Context**	
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDataElementDataType	
BSW Parameter		**BSW Type**
FLOAT_N		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Type of the data is float array.		
Template Description		

▽

△

BaseTypeDirectDefinition.baseTypeEncoding: This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **BaseTypeDirectDefinition.baseTypeSize:** Describes the length of the data type specified in the container in bits. **DiagnosticValueNeeds.fixedLength:** This attribute is applicable only if the DiagnosticValueNeeds is aggregated within a BswModuleDependency. This attribute controls whether the data length of the data is fixed.			
M2 Parameter			
AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeEncoding, AsamHdo::BaseTypes::BaseTypeDirectDefinition.baseTypeSize, CommonStructure::ServiceNeeds::DiagnosticValueNeeds.fixedLength			
Mapping Rule			**Mapping Type**
baseTypeEncoding = NONE, WINDOWS-1252, UTF-8, BCD-P, BCD-UP baseTypeSize = 8 max NumberOfElements shall not exist arraySizeSemantics shall not exist Derivation from DiagnosticValueNeeds.fixedLength=1 possible.			full
Mapping Status			**ECUC Parameter ID**
valid			

BSW Module	**BSW Context**		
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDiagnosisScaling/DemAlternativeDataInterface		
BSW Parameter		**BSW Type**	
DemDataElement		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Alternative Diagnosis Representation for the data defined by the means of a VariableDataPrototype in a DataInterface. The CompuMethod of the data type of the referenced VariableDataPrototype will be applied to the data type of the VariableDataPrototype in the interface used by the Dem.			
Template Description			
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.			
M2 Parameter			
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Dem_00845]

BSW Module	**BSW Context**		
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDiagnosisScaling/DemAlternativeDataInterface		
BSW Parameter		**BSW Type**	
DemPortInterfaceMapping		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Optional reference to PortInterfaceMapping which defines the mapping rules. The PortInterfaceMapping is used to get the DataPrototypeMapping that describes a conversion between the data prototype referenced by DemDataElement and the data prototype referenced from DcmDspExternalSRDataElementClass.			
Template Description			

▽

△

Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).	
M2 Parameter	
SWComponentTemplate::PortInterface::PortInterfaceMapping	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Dem_00846]

BSW Module	**BSW Context**	
Dem	Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDiagnosis Scaling/DemAlternativeDataType	
BSW Parameter		**BSW Type**
DemApplicationDataType		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Alternative Diagnosis Representation for the data defined by the means of a ApplicationDataType of category VALUE, BOOLEAN or ARRAY. The CompuMethod that applies to the referenced ApplicationDataType in case of category VALUE or BOOLEAN will be applied to the data type of the VariableDataPrototype in the interface used by the Dem.		
Template Description		
A primitive data type defines a set of allowed values.		
M2 Parameter		
SWComponentTemplate::Datatype::Datatypes::ApplicationPrimitiveDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Dem_00848]

G.7 BswM

BSW Module	**BSW Context**	
BswM	BswM/BswMConfig/BswMArbitration/BswMModeCondition/BswMConditionValue/BswMCompu ScaleModeValue	
BSW Parameter		**BSW Type**
BswMCompuMethodRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
This is a foreign reference to the CompuMethod used for mode requests.		
Template Description		
This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant.		
M2 Parameter		
AsamHdo::ComputationMethod::CompuMethod		

▽

△

Mapping Rule	Mapping Type
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_BswM_01040]

BSW Module	BSW Context	
BswM	BswM/BswMConfig/BswMArbitration/BswMModeCondition/BswMConditionValue/BswMMode Declaration	
BSW Parameter		**BSW Type**
BswMModeValueRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
This is a foreign reference to the Mode Declaration used for the mode requests corresponding to this condition.		
Template Description		
Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.		
M2 Parameter		
CommonStructure::ModeDeclaration::ModeDeclaration		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_BswM_00864]

BSW Module	BSW Context	
BswM	BswM/BswMConfig/BswMArbitration/BswMModeRequestPort/BswMModeInitValue/BswMCompu ScaleModeValue	
BSW Parameter		**BSW Type**
BswMCompuMethodRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
This is a foreign reference to the CompuMethod used for mode requests.		
Template Description		
This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant.		
M2 Parameter		
AsamHdo::ComputationMethod::CompuMethod		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_BswM_01040]

BSW Module	BSW Context	
BswM	BswM/BswMConfig/BswMArbitration/BswMModeRequestPort/BswMModeRequestSource/Bsw MBswModeNotification	
BSW Parameter		**BSW Type**
BswMBswModeDeclarationGroupPrototypeRef		ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
This is a foreign reference to the Mode Declaration Group Prototype.	
Template Description	
The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.	
M2 Parameter	
CommonStructure::ModeDeclaration::ModeDeclarationGroupPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_BswM_00927]

BSW Module	BSW Context
BswM	BswM/BswMConfig/BswMArbitration/BswMModeRequestPort/BswMModeRequestSource/BswMSwcModeNotification
BSW Parameter	**BSW Type**
BswMSwcModeNotificationModeDeclarationGroupPrototypeRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
This is a foreign reference to the ModeDeclarationGroupPrototype.	
Template Description	
The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.	
M2 Parameter	
CommonStructure::ModeDeclaration::ModeDeclarationGroupPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_BswM_00893]

BSW Module	BSW Context
BswM	BswM/BswMConfig/BswMArbitration/BswMModeRequestPort/BswMModeRequestSource/BswMSwcModeRequest
BSW Parameter	**BSW Type**
BswMSwcModeRequestVariableDataPrototypeRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
This is a reference to the VariableDataPrototype.	
Template Description	
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.	
M2 Parameter	
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_BswM_01046]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMDataTypeMappingSets		
BSW Parameter		**BSW Type**	
BswMDataTypeMappingSetRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to DataTypeMappingSet.			
Template Description			
This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups.			
M2 Parameter			
SWComponentTemplate::Datatype::Datatypes::DataTypeMappingSet			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_BswM_00937]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMModeControl/BswMAction/BswMAvailableActions/BswMRteMode Request		
BSW Parameter		**BSW Type**	
BswMRequestedModeRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
This is a foreign reference to the Mode Declaration used for the mode request			
Template Description			
Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
M2 Parameter			
CommonStructure::ModeDeclaration::ModeDeclaration			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_BswM_01024]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMModeControl/BswMAction/BswMAvailableActions/BswMRteSwitch		
BSW Parameter		**BSW Type**	
BswMSwitchedMode		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
This parameter contains the integer value that corresponds to a certain mode in a Mode Declaration Group.			
Template Description			
Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
M2 Parameter			
CommonStructure::ModeDeclaration::ModeDeclaration			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_BswM_00896]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMModeControl/BswMAction/BswMAvailableActions/BswMSchMSwitch		
BSW Parameter		**BSW Type**	
BswMSchMSwitchedMode		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
This parameter contains the integer value that corresponds to a certain mode in a Mode Declaration Group.			
Template Description			
Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
M2 Parameter			
CommonStructure::ModeDeclaration::ModeDeclaration			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_BswM_00901]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMModeControl/BswMRteModeRequestPort		
BSW Parameter		**BSW Type**	
BswMRteModeRequestPortInterfaceRef		ECUC-INSTANCE-REFERENCE-DEF	
BSW Description			
This is an instance reference to the variable data prototype used for the mode request.			
Template Description			
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.			
M2 Parameter			
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_BswM_01025]

BSW Module	BSW Context		
BswM	BswM/BswMConfig/BswMModeControl/BswMRteModeRequestPort		
BSW Parameter		**BSW Type**	
BswMRteModeRequestVariableDataPrototypeSRRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
This is a foreign reference to a VariableDataPrototype used for the mode request.			
Template Description			
A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.			
M2 Parameter			
SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full

▽

△

Mapping Status	ECUC Parameter ID
valid	[ECUC_BswM_01057]

BSW Module	BSW Context	
BswM	BswM/BswMConfig/BswMModeControl/BswMSwitchPort	
BSW Parameter		**BSW Type**
BswMModeSwitchInterfaceRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to the ModeSwitchInterface from which the BswM will generate a PPortPrototype.		
Template Description		
A mode switch interface declares a ModeDeclarationGroupPrototype to be sent and received.		
M2 Parameter		
SWComponentTemplate::PortInterface::ModeSwitchInterface		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_BswM_00951]

BSW Module	BSW Context	
BswM	BswM/BswMConfig/BswMModeControl/BswMSwitchPort	
BSW Parameter		**BSW Type**
BswMSchMModeDeclarationGroupRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to the ModeDeclarationGroup from which the BswM will generate a ModeDeclarationGroupPrototype.		
Template Description		
A collection of Mode Declarations. Also, the initial mode is explicitly identified.		
M2 Parameter		
CommonStructure::ModeDeclaration::ModeDeclarationGroup		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_BswM_01031]

G.8 MemMap

BSW Module	BSW Context	
MemMap	MemMap/MemMapAllocation/MemMapGenericMapping	
BSW Parameter		**BSW Type**
MemMapSwAddressMethodRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		

▽

△

<table>
<tr><td colspan="2">Reference to the SwAddrMethod which applies to the MemMapGenericMapping.</td></tr>
<tr><td colspan="2">Template Description</td></tr>
<tr><td colspan="2">Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components.</td></tr>
<tr><td colspan="2">M2 Parameter</td></tr>
<tr><td colspan="2">DataDictionary::AuxillaryObjects::SwAddrMethod</td></tr>
<tr><td>Mapping Rule</td><td>Mapping Type</td></tr>
<tr><td>1:1 mapping</td><td>full</td></tr>
<tr><td>Mapping Status</td><td>ECUC Parameter ID</td></tr>
<tr><td>valid</td><td>[ECUC_MemMap_-00013]</td></tr>
</table>

<table>
<tr><td>BSW Module</td><td colspan="2">BSW Context</td></tr>
<tr><td>MemMap</td><td colspan="2">MemMap/MemMapAllocation/MemMapSectionSpecificMapping</td></tr>
<tr><td colspan="2">BSW Parameter</td><td>BSW Type</td></tr>
<tr><td colspan="2">MemMapMemorySectionRef</td><td>ECUC-FOREIGN-REFERENCE-DEF</td></tr>
<tr><td colspan="3">BSW Description</td></tr>
<tr><td colspan="3">Reference to the MemorySection which applies to the MemMapSectionSpecificMapping.</td></tr>
<tr><td colspan="3">Template Description</td></tr>
<tr><td colspan="3">Provides a description of an abstract memory section used in the Implementation for code or data. It shall be declared by the Implementation Description of the module or component, which actually allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the generated Implementation Description of the RTE shall contain the corresponding MemorySections.

The attribute "symbol" (if symbol is missing: "shortName") defines the module or component specific section name used in the code. For details see the document "Specification of Memory Mapping". Typically the section name is build according the pattern:

<SwAddrMethod shortName>[_<further specialization nominator>][_<alignment>]

where

[<SwAddrMethod shortName>] is the shortName of the referenced SwAddrMethod
[_<further specialization nominator>] is an optional infix to indicate the specialization in the case that several MemorySections for different purpose of the same Implementation Description referring to the same or equally named SwAddrMethods.
[_<alignment>] is the alignment attributes value and is only applicable in the case that the memoryAllocation KeywordPolicy value of the referenced SwAddrMethod is set to addrMethodShortNameAndAlignment

MemorySection used to Implement the code of RunnableEntitys and BswSchedulableEntitys shall have a symbol (if missing: shortName) identical to the referred SwAddrMethod to conform to the generated RTE header files.

In addition to the section name described above, a prefix is used in the corresponding macro code in order to define a name space. This prefix is by default given by the shortName of the BswModuleDescription resp. the SwComponentType. It can be superseded by the prefix attribute.</td></tr>
<tr><td colspan="3">M2 Parameter</td></tr>
<tr><td colspan="3">CommonStructure::ResourceConsumption::MemorySectionUsage::MemorySection</td></tr>
<tr><td colspan="2">Mapping Rule</td><td>Mapping Type</td></tr>
<tr><td colspan="2">1:1 mapping</td><td>full</td></tr>
<tr><td colspan="2">Mapping Status</td><td>ECUC Parameter ID</td></tr>
<tr><td colspan="2">valid</td><td>[ECUC_MemMap_-00016]</td></tr>
</table>

G.9 RTE

BSW Module	BSW Context		
Rte	Rte/RteImplicitCommunication		
BSW Parameter		**BSW Type**	
RteVariableReadAccessRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to the VariableAccess in the dataReadAccess role.			
Template Description			
The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableDataPrototype. The kind of access is specified by the role in which the class is used.			
M2 Parameter			
SWComponentTemplate::SwcInternalBehavior::DataElements::VariableAccess			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Rte_09035]

BSW Module	BSW Context		
Rte	Rte/RteImplicitCommunication		
BSW Parameter		**BSW Type**	
RteVariableWriteAccessRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to the VariableAccess in the dataWriteAccess role.			
Template Description			
The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableDataPrototype. The kind of access is specified by the role in which the class is used.			
M2 Parameter			
SWComponentTemplate::SwcInternalBehavior::DataElements::VariableAccess			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Rte_09036]

BSW Module	BSW Context	
Rte	Rte/RteOsInteraction/RteModeToScheduleTableMapping/RteModeSchtblMapBsw	
BSW Parameter		**BSW Type**
RteModeSchtblMapBswProvidedModeGroupRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to an instance of a ModeDeclarationGroupPrototype of a Bsw-Module.		
Template Description		
The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.		
M2 Parameter		
CommonStructure::ModeDeclaration::ModeDeclarationGroupPrototype		

▽

△

Mapping Rule	Mapping Type
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09053]

BSW Module	BSW Context
Rte	Rte/RteOsInteraction/RteModeToScheduleTableMapping
BSW Parameter	**BSW Type**
RteModeSchtblMapModeDeclarationRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Reference to the ModeDeclarations.	
Template Description	
Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.	
M2 Parameter	
CommonStructure::ModeDeclaration::ModeDeclaration	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09054]

BSW Module	BSW Context
Rte	Rte/RteOsInteraction/RteModeToScheduleTableMapping/RteModeSchtblMapSwc
BSW Parameter	**BSW Type**
RteModeSchtblMapSwcPortRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Reference to the PPortPrototype of a SwComponentPrototype.	
Template Description	
Component port providing a certain port interface.	
M2 Parameter	
SWComponentTemplate::Components::PPortPrototype	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09057]

BSW Module	BSW Context
Rte	Rte/RteSwComponentInstance/RteEventToIsrMapping
BSW Parameter	**BSW Type**
RteIsrEventRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Reference to the description of the ExternalTriggerOccurredEvent or TimingEvent which is pointing to the RunnableEntity being mapped. This allows a fine grained mapping of RunnableEntites based on the activating RTEEvent.	
Template Description	

▽

△

Abstract base class for all RTE-related events	
M2 Parameter	
SWComponentTemplate::SwcInternalBehavior::RTEEvents::RTEEvent	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09153]

BSW Module	**BSW Context**	
Rte	Rte/RteSwComponentInstance/RteEventToTaskMapping	
BSW Parameter		**BSW Type**
RteActivationOffset		ECUC-FLOAT-PARAM-DEF
BSW Description		
Activation offset in seconds.		
Template Description		
The value makes an assumption about the time offset of the first activation of the RunnableEntity triggered by the mapped TimingEvent relative to the periodic activation of the time base of this TimingEvent. Unit: second.		
M2 Parameter		
SWComponentTemplate::SwcInternalBehavior::RTEEvents::TimingEvent.offset		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Rte_09018]

BSW Module	**BSW Context**	
Rte	Rte/RteSwComponentInstance/RteEventToTaskMapping	
BSW Parameter		**BSW Type**
RteEventRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to the description of the RTEEvent which is pointing to the RunnableEntity being mapped. This allows a fine grained mapping of RunnableEntites based on the activating RTEEvent.		
Template Description		
Abstract base class for all RTE-related events		
M2 Parameter		
SWComponentTemplate::SwcInternalBehavior::RTEEvents::RTEEvent		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Rte_09019]

BSW Module	**BSW Context**	
Rte	Rte/RteSwComponentInstance/RteExclusiveAreaImplementation	
BSW Parameter		**BSW Type**
RteExclusiveAreaRef		ECUC-FOREIGN-REFERENCE-DEF

▽

△

BSW Description	
Reference to the ExclusiveArea.	
Template Description	
Prevents an executable entity running in the area from being preempted.	
M2 Parameter	
CommonStructure::InternalBehavior::ExclusiveArea	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09032]

BSW Module	BSW Context
Rte	Rte/RteSwComponentInstance/RteInternalTriggerConfig
BSW Parameter	**BSW Type**
RteSwcTriggerSourceRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Reference to an InternalTriggeringPoint of the related component instance. The referenced InternalTriggeringPoint has to belong to the same software component instance as the RteSwComponent Instance owning this parameter configures.	
Template Description	
If a RunnableEntity owns an InternalTriggeringPoint it is entitled to trigger the execution of RunnableEntities of the corresponding software-component.	
M2 Parameter	
SWComponentTemplate::SwcInternalBehavior::Trigger::InternalTriggeringPoint	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09097]

BSW Module	BSW Context
Rte	Rte/RteSwComponentInstance/RteNvRamAllocation
BSW Parameter	**BSW Type**
RteSwNvBlockDescriptorRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Reference to the NvBlockDescriptor in case the RTE needs to call the NvM directly (e.g. for the supportDirtyFlag feature, storeCyclic feature, server invocation for NV data management or mode switch based invocation NvM services).	
Template Description	
Specifies the properties of exactly on NVRAM Block.	
M2 Parameter	
SWComponentTemplate::NvBlockComponent::NvBlockDescriptor	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_Rte_09132]

BSW Module	BSW Context		
Rte	Rte/RteSwComponentInstance/RteNvRamAllocation		
BSW Parameter		**BSW Type**	
RteSwNvRamMappingRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to the SwSeriveDependency which is used to specify the NvBlockNeeds.			
Template Description			
Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
M2 Parameter			
SWComponentTemplate::SwcInternalBehavior::ServiceMapping::SwcServiceDependency			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Rte_09044]

BSW Module	BSW Context		
Rte	Rte/RteSwComponentInstance		
BSW Parameter		**BSW Type**	
RteSoftwareComponentInstanceRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to a SwComponentPrototype.			
Template Description			
Role of a software component within a composition.			
M2 Parameter			
SWComponentTemplate::Composition::SwComponentPrototype			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Rte_09004]

BSW Module	BSW Context		
Rte	Rte/RteSwComponentType/RteComponentTypeCalibration		
BSW Parameter		**BSW Type**	
RteCalibrationSwAddrMethodRef		ECUC-FOREIGN-REFERENCE-DEF	
BSW Description			
Reference to the SwAddrMethod for which software calibration support shall be enabled.			
Template Description			
Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components.			
M2 Parameter			
DataDictionary::AuxillaryObjects::SwAddrMethod			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			[ECUC_Rte_09038]

BSW Module	BSW Context	
Rte	Rte/RteSwComponentType	
BSW Parameter		**BSW Type**
RteComponentTypeRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to either AtomicSwComponentType or ParameterSwComponentType.		
Template Description		
Base class for AUTOSAR software components.		
M2 Parameter		
SWComponentTemplate::Components::`SwComponentType`		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Rte_09003]

BSW Module	BSW Context	
Rte	Rte/RteSwComponentType	
BSW Parameter		**BSW Type**
RteImplementationRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
The Implementation which shall be assigned to the SwComponentType.		
Template Description		
This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software.		
M2 Parameter		
SWComponentTemplate::SwcImplementation::`SwcImplementation`		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Rte_09028]

G.10 ECUC

BSW Module	BSW Context	
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType	
BSW Parameter		**BSW Type**
MetaDataItem		ECUC-PARAM-CONF-CONTAINER-DEF
BSW Description		
The content of meta data in a Pdu consists of an ordered list of meta data items. This container represents a meta data item that is contained in meta data of a Pdu.		
Template Description		
This meta-class represents a single meta-data item.		
M2 Parameter		

▽

△

SWComponentTemplate::PortInterface::MetaDataItem	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_EcuC_00074]

BSW Module	**BSW Context**
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem
BSW Parameter	**BSW Type**
MetaDataItemLength	ECUC-INTEGER-PARAM-DEF
BSW Description	
This parameter defines the length of a meta data item in bytes.	
Template Description	
This attribute determines the length of the MetaDataItem at run-time.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.length	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_EcuC_00075]

BSW Module	**BSW Context**
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType
BSW Parameter	**BSW Type**
ADDRESS_EXTENSION_8	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Address extension field (N_AE) of the mixed addressing modes with 11bit and 29bit CAN ID of ISO 15765-2. Size: 8 bits.	
Template Description	
This aggregation contributes the specification of the concrete meta-data item type.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType	
Mapping Rule	**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == ADDRESS_EXTENSION_16	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	**BSW Context**
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType
BSW Parameter	**BSW Type**
CAN_ID_32	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
CAN ID according to ISO 11898-2, either 29 bits or 11 bits. Encoding according to Can_IdType. Size: 32 bits.	

▽

△

Template Description	
This aggregation contributes the specification of the concrete meta-data item type.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType	
Mapping Rule	**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == CAN_ID_32	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType
BSW Parameter	**BSW Type**
ETHERNET_MAC_64	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
Ethernet MAC address. Size: 64 bits.	
Template Description	
This aggregation contributes the specification of the concrete meta-data item type.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType	
Mapping Rule	**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == ETHERNET_MAC_64	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType
BSW Parameter	**BSW Type**
LIN_NAD_8	ECUC-ENUMERATION-LITERAL-DEF
BSW Description	
LIN node address as used in the LIN transport protocol. Size: 8 bits.	
Template Description	
This aggregation contributes the specification of the concrete meta-data item type.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType	
Mapping Rule	**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == LIN_NAD_8	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType
BSW Parameter	**BSW Type**
PRIORITY_8	ECUC-ENUMERATION-LITERAL-DEF

▽

△

BSW Description	
Priority field of SAE J1939 IDs, or Ethernet QoS parameter. Size: 8 bits.	
Template Description	
This aggregation contributes the specification of the concrete meta-data item type.	
M2 Parameter	
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType	
Mapping Rule	**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == PRIORITY_8	full
Mapping Status	**ECUC Parameter ID**
valid	

BSW Module	BSW Context	
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType	
BSW Parameter		**BSW Type**
SOCKET_CONNECTION_ID_16		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
SoAd socket connection ID. Size: 16 bits.		
Template Description		
This aggregation contributes the specification of the concrete meta-data item type.		
M2 Parameter		
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType		
Mapping Rule		**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == SOCKET_CONNECTION_ID_16		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType	
BSW Parameter		**BSW Type**
SOURCE_ADDRESS_16		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Source address of CanTp, FrTp, or DoIP transport protocol messages, or of SAE J1939 messages. Size: 16 bits.		
Template Description		
This aggregation contributes the specification of the concrete meta-data item type.		
M2 Parameter		
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType		
Mapping Rule		**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == SOURCE_ADDRESS_16		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
EcuC	EcuC/EcucConfigSet/EcucPduCollection/MetaDataType/MetaDataItem/MetaDataItemType	
BSW Parameter		**BSW Type**
TARGET_ADDRESS_16		ECUC-ENUMERATION-LITERAL-DEF
BSW Description		
Target address of CanTp, FrTp, or DoIP transport protocol messages, or destination address of SAE J1939 messages. Size: 16 bits.		
Template Description		
This aggregation contributes the specification of the concrete meta-data item type.		
M2 Parameter		
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType		
Mapping Rule		**Mapping Type**
SWComponentTemplate::PortInterface::MetaDataItem.metaDataItemType == TARGET_ADDRESS_16		full
Mapping Status		**ECUC Parameter ID**
valid		

BSW Module	BSW Context	
EcuC	EcuC/EcucConfigSet/EcucPduCollection/Pdu	
BSW Parameter		**BSW Type**
MetaDataTypeRef		ECUC-REFERENCE-DEF
BSW Description		
Reference to meta data that is transported in the Pdu through the AUTOSAR layers.		
Template Description		
VariableDataPrototype: A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics. **SenderReceiverToSignalMapping:** Mapping of a sender receiver communication data element to a signal. **SystemSignal:** The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with exactly one system signal defined for each data element prototype sent and received by connected SW component instances. **ISignal:** Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different Signal IPdus to multiple receivers. To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping. ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping). In case of the SystemSignalGroup an ISignal shall be created for each SystemSignal contained in the SystemSignalGroup. **ISignalIPdu:** Represents the IPdus handled by Com. The ISignalIPdu assembled and disassembled in AUTOSAR COM consists of one or more signals. In case no multiplexing is performed this IPdu is routed to/from the Interface Layer. A maximum of one dynamic length signal per IPdu is allowed.		
M2 Parameter		

▽

△

SWComponentTemplate::Datatype::DataPrototypes::VariableDataPrototype, SystemTemplate::DataMapping::SenderReceiverToSignalMapping, SystemTemplate::Fibex::FibexCore::CoreCommunication::SystemSignal, SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignal, SystemTemplate::Fibex::FibexCore::CoreCommunication::ISignalIPdu	
Mapping Rule	**Mapping Type**
A MetaDataTypeRef shall be derived for a given Pdu if a MetaDataItemSet exists that refers to a VariablePrototype that is also referenced from a SenderReceiverToSignalMapping that in turn references a SystemSignal that is referenced by a ISignal that is mapped to an ISignalIPdu that is derived to the mentioned Pdu in EcuC.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_EcuC_00077]

BSW Module	**BSW Context**
EcuC	EcuC/EcucPartitionCollection/EcucPartition
BSW Parameter	**BSW Type**
EcucPartitionSoftwareComponentInstanceRef	ECUC-INSTANCE-REFERENCE-DEF
BSW Description	
References the SW Component instances from the Ecu Extract that shall be executed in this partition.	
Template Description	
M2 Parameter	
SystemTemplate::SWmapping::SwcToEcuMapping.partition	
Mapping Rule	**Mapping Type**
The EcucPartitionSoftwareComponentInstanceRef is derived from an SwcToEcuMapping which references an EcuPartition and one or several SwComponentPrototypes. For each SwComponentPrototype that is referenced by the SwcToEcuMapping in the component role an EcucPartitionSoftwareComponentInstanceRef shall be created that refers to the same SwComponentPrototype as the the SwcToEcuMapping.	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_EcuC_00036]

BSW Module	**BSW Context**
EcuC	EcuC/EcucUnitGroupAssignment
BSW Parameter	**BSW Type**
EcucUnitGroupRef	ECUC-FOREIGN-REFERENCE-DEF
BSW Description	
Optional reference to the UnitGroup to support the generation of ASAM MCD file. These UnitGroups are selecting a set of units for a specific country.	
Template Description	
This meta-class represents the ability to specify a logical grouping of units.The category denotes the unit system that the referenced units are associated to. In this way, e.g. country-specific unit systems (CATEGORY="COUNTRY") can be defined as well as specific unit systems for certain application domains. In the same way a group of equivalent units, can be defined which are used in different countries, by setting CATEGORY="EQUIV_UNITS". KmPerHour and MilesPerHour could such be combined to one group named "vehicle_speed". The unit MeterPerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named "speed". Note that the UnitGroup does not ensure the physical compliance of the units. This is maintained by the physical dimension.	
M2 Parameter	

▽

△

AsamHdo::Units::UnitGroup	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**ECUC Parameter ID**
valid	[ECUC_EcuC_00062]

G.11 OS

BSW Module	**BSW Context**	
Os	Os/OsIoc/OsIocCommunication/OsIocDataProperties	
BSW Parameter		**BSW Type**
OsIocDataTypeRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
This is the type of the data to be transferred on the IOC communication channel. This attribute is necessary to generate the parameter type of the Ioc functions. Additionally this information should be used to compute the data size for necessary data copy operations within the Ioc module. If more than one attribute is defined, the IOC generator should generate an IocXxxGroup function (Xxx= CHOICE [Send, Receive, Write, Read]). N:1 or N:M communication (Multiplicity of OsIocSenderProperties > 1) is only allowed for multiplicity of OsIocDataTypeRef = 1		
Template Description		
Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code.		
M2 Parameter		
CommonStructure::ImplementationDataTypes::ImplementationDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Os_01005]

BSW Module	**BSW Context**	
Os	Os/OsIoc/OsIocCommunication/OsIocDataProperties	
BSW Parameter		**BSW Type**
OsMemoryMappingCodeLocationRef		ECUC-FOREIGN-REFERENCE-DEF
BSW Description		
Reference to the memory mapping containing details about the section where the IOC buffer is placed.		
Template Description		
Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code.		
M2 Parameter		
CommonStructure::ImplementationDataTypes::ImplementationDataType		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**ECUC Parameter ID**
valid		[ECUC_Os_00405]

G.12 SecOC

BSW Module	BSW Context		
SecOC	SecOC/SecOCRxPduProcessing/SecOCClientServerVerificationStatusPropagationMode		
BSW Parameter		**BSW Type**	
BOTH		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Both "TRUE" and "FALSE" AuthenticationStatus is propagated to SW-C			
Template Description			
Verification attempts that came out "false" or "true" shall be forwarded to the application software.			
M2 Parameter			
CommonStructure::ServiceNeeds::VerificationStatusIndicationModeEnum.failureAndSuccess			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			

BSW Module	BSW Context		
SecOC	SecOC/SecOCRxPduProcessing/SecOCClientServerVerificationStatusPropagationMode		
BSW Parameter		**BSW Type**	
FAILURE_ONLY		ECUC-ENUMERATION-LITERAL-DEF	
BSW Description			
Only "FALSE" Authentication Status is propagated to SW-C			
Template Description			
Only verification attempts that came out "false" shall be forwarded to the application software.			
M2 Parameter			
CommonStructure::ServiceNeeds::VerificationStatusIndicationModeEnum.failureOnly			
Mapping Rule			**Mapping Type**
1:1 mapping			full
Mapping Status			**ECUC Parameter ID**
valid			

H Splitable Elements in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpSplitable≫ in the scope of this document.

Each entry in the table consists of the identification of the specific model element itself and the applicable value of the tagged value atp.Splitkey.

For more information about the concept of splitable model elements and how these shall be treated please refer to [11].

Name of splitable element	***Splitkey***
AdminData.sdg	sdg
ApplicationRecordDataType.element	element.shortName, element.variationPoint.short Label
ARPackage.arPackage	arPackage.shortName, arPackage.variation Point.shortLabel
ARPackage.element	element.shortName, element.variationPoint.short Label
ARPackage.referenceBase	referenceBase.shortLabel
ArrayValueSpecification.element	element, element.variationPoint.shortLabel
AtomicSwComponentType.internalBehavior	internalBehavior.shortName, internal Behavior.variationPoint.shortLabel
AtomicSwComponentType.symbolProps	symbolProps.shortName
AutosarDataType.swDataDefProps	swDataDefProps
BulkNvDataDescriptor.nvBlockDataMapping	nvBlockDataMapping, nvBlockData Mapping.variationPoint.shortLabel
CalibrationParameterValueSet.calibrationParameterValue	calibrationParameterValue, calibrationParameter Value.variationPoint.shortLabel
ClientServerInterface.operation	operation.shortName, operation.variationPoint.short Label
ClientServerOperation.argument	argument.shortName, argument.variationPoint.short Label
CompositeNetworkRepresentation.networkRepresentation	networkRepresentation
CompositionSwComponentType.component	component.shortName, component.variation Point.shortLabel
CompositionSwComponentType.connector	connector.shortName, connector.variationPoint.short Label
CompositionSwComponentType.constantValueMapping	constantValueMapping
CompositionSwComponentType.dataTypeMapping	dataTypeMapping
CompositionSwComponentType.instantiationRTEEventProps	instantiationRTEEventProps.shortLabel, instantiationRTEEventProps.variationPoint.short Label
CompuScales.compuScale	compuScale, compuScale.variationPoint.shortLabel
ConsistencyNeeds.dpgDoesNotRequireCoherency	dpgDoesNotRequireCoherency.shortName, dpg DoesNotRequireCoherency.variationPoint.short Label
ConsistencyNeeds.dpgRequiresCoherency	dpgRequiresCoherency.shortName, dpgRequires Coherency.variationPoint.shortLabel
ConsistencyNeeds.regDoesNotRequireStability	regDoesNotRequireStability.shortName, regDoes NotRequireStability.variationPoint.shortLabel
ConsistencyNeeds.regRequiresStability	regRequiresStability.shortName, regRequires Stability.variationPoint.shortLabel
DataPrototype.swDataDefProps	swDataDefProps

▽

△

Name of splitable element	*Splitkey*
DataPrototypeGroup.dataPrototypeGroup	dataPrototypeGroup.contextSwComponent Prototype, dataPrototypeGroup.targetDataPrototype Group, dataPrototypeGroup.variationPoint.short Label
DataPrototypeGroup.implicitDataAccess	implicitDataAccess.contextSwComponentPrototype, implicitDataAccess.contextPortPrototype, implicit DataAccess.targetVariableDataPrototype, implicit DataAccess.variationPoint.shortLabel
Describable.adminData	adminData
EndToEndProtection.endToEndProfile	endToEndProfile
EndToEndProtection.endToEndProtectionISignalIPdu	endToEndProtectionISignalIPdu, endToEnd ProtectionISignalIPdu.variationPoint.shortLabel
EndToEndProtection.endToEndProtectionVariablePrototype	endToEndProtectionVariablePrototype.shortLabel, endToEndProtectionVariablePrototype.variation Point.shortLabel
EndToEndProtectionSet.endToEndProtection	endToEndProtection.shortName, endToEnd Protection.variationPoint.shortLabel
ErrorTracerNeeds.tracedFailure	tracedFailure.shortName, tracedFailure.variation Point.shortLabel
ExecutableEntity.canEnter	canEnter.exclusiveArea, canEnter.variation Point.shortLabel
ExecutableEntity.runsInside	runsInside.exclusiveArea, runsInside.variation Point.shortLabel
Identifiable.adminData	adminData
Implementation.buildActionManifest	buildActionManifest.buildActionManifest, buildAction Manifest.variationPoint.shortLabel
Implementation.generatedArtifact	generatedArtifact.shortName, generated Artifact.variationPoint.shortLabel
Implementation.mcSupport	mcSupport
Implementation.requiredArtifact	requiredArtifact.shortName, required Artifact.variationPoint.shortLabel
Implementation.requiredGeneratorTool	requiredGeneratorTool.shortName, required GeneratorTool.variationPoint.shortLabel
Implementation.resourceConsumption	resourceConsumption.shortName
ImplementationDataType.subElement	subElement.shortName, subElement.variation Point.shortLabel
ImplementationDataType.symbolProps	symbolProps.shortName
ImplementationDataTypeElement.subElement	subElement.shortName, subElement.variation Point.shortLabel
ImplementationDataTypeElement.swDataDefProps	swDataDefProps
InstantiationDataDefProps.swDataDefProps	swDataDefProps
InternalBehavior.constantMemory	constantMemory.shortName, constant Memory.variationPoint.shortLabel
InternalBehavior.constantValueMapping	constantValueMapping
InternalBehavior.dataTypeMapping	dataTypeMapping
InternalBehavior.exclusiveArea	exclusiveArea.shortName, exclusiveArea.variation Point.shortLabel
InternalBehavior.exclusiveAreaNestingOrder	exclusiveAreaNestingOrder.shortName, exclusive AreaNestingOrder.variationPoint.shortLabel
InternalBehavior.staticMemory	staticMemory.shortName, staticMemory.variation Point.shortLabel
ModeDeclarationGroup.modeDeclaration	modeDeclaration.shortName, mode Declaration.variationPoint.shortLabel
NvBlockDescriptor.clientServerPort	clientServerPort, clientServerPort.variation Point.shortLabel

▽

△

Name of splitable element	*Splitkey*
NvBlockDescriptor.constantValueMapping	constantValueMapping
NvBlockDescriptor.dataTypeMapping	dataTypeMapping
NvBlockDescriptor.instantiationDataDefProps	instantiationDataDefProps, instantiationDataDefProps.variationPoint.shortLabel
NvBlockDescriptor.modeSwitchEventTriggeredActivity	modeSwitchEventTriggeredActivity, modeSwitchEventTriggeredActivity.variationPoint.shortLabel
NvBlockDescriptor.nvBlockDataMapping	nvBlockDataMapping, nvBlockDataMapping.variationPoint.shortLabel
NvBlockSwComponentType.bulkNvDataDescriptor	bulkNvDataDescriptor.shortName, bulkNvDataDescriptor.variationPoint.shortLabel
NvBlockSwComponentType.nvBlockDescriptor	nvBlockDescriptor.shortName, nvBlockDescriptor.variationPoint.shortLabel
ParameterAccess.swDataDefProps	swDataDefProps
ParameterSwComponentType.constantMapping	constantMapping
ParameterSwComponentType.dataTypeMapping	dataTypeMapping
ParameterSwComponentType.instantiationDataDefProps	instantiationDataDefProps, instantiationDataDefProps.variationPoint.shortLabel
PerInstanceMemory.swDataDefProps	swDataDefProps
PortGroup.outerPort	outerPort.portPrototype, outerPort.variationPoint.shortLabel
PortInterfaceMappingSet.portInterfaceMapping	portInterfaceMapping.shortName, portInterfaceMapping.variationPoint.shortLabel
RapidPrototypingScenario.rptContainer	rptContainer.shortName, rptContainer.variationPoint.shortLabel
RapidPrototypingScenario.rptProfile	rptProfile.shortName
RapidPrototypingScenario.rptSystem	rptSystem
ReceiverComSpec.networkRepresentation	networkRepresentation
RecordValueSpecification.field	field, field.variationPoint.shortLabel
RptContainer.byPassPoint	byPassPoint.contextElement, byPassPoint.target, byPassPoint.variationPoint.shortLabel
RptContainer.explicitRptProfileSelection	explicitRptProfileSelection
RptContainer.rptContainer	rptContainer.shortName, rptContainer.variationPoint.shortLabel
RptContainer.rptHook	rptHook, rptHook.variationPoint.shortLabel
RTEEvent.disabledMode	disabledMode.contextPort, disabledMode.contextModeDeclarationGroupPrototype, disabledMode.targetModeDeclaration
RuleArguments.vtf	vtf, vtf.variationPoint.shortLabel
RuleBasedValueSpecification.arguments	arguments, arguments.variationPoint.shortLabel
RunnableEntity.asynchronousServerCallResultPoint	asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel
RunnableEntity.dataReadAccess	dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel
RunnableEntity.dataReceivePointByArgument	dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel
RunnableEntity.dataReceivePointByValue	dataReceivePointByValue.shortName, dataReceivePointByValue.variationPoint.shortLabel
RunnableEntity.dataSendPoint	dataSendPoint.shortName, dataSendPoint.variationPoint.shortLabel
RunnableEntity.dataWriteAccess	dataWriteAccess.shortName, dataWriteAccess.variationPoint.shortLabel

▽

△

Name of splitable element	***Splitkey***
RunnableEntity.externalTriggeringPoint	externalTriggeringPoint.ident.shortName, externalTriggeringPoint.variationPoint.shortLabel
RunnableEntity.internalTriggeringPoint	internalTriggeringPoint.shortName, internalTriggeringPoint.variationPoint.shortLabel
RunnableEntity.modeAccessPoint	modeAccessPoint.ident.shortName, modeAccessPoint.variationPoint.shortLabel
RunnableEntity.modeSwitchPoint	modeSwitchPoint.shortName, modeSwitchPoint.variationPoint.shortLabel
RunnableEntity.parameterAccess	parameterAccess.shortName, parameterAccess.variationPoint.shortLabel
RunnableEntity.readLocalVariable	readLocalVariable.shortName, readLocalVariable.variationPoint.shortLabel
RunnableEntity.serverCallPoint	serverCallPoint.shortName, serverCallPoint.variationPoint.shortLabel
RunnableEntity.writtenLocalVariable	writtenLocalVariable.shortName, writtenLocalVariable.variationPoint.shortLabel
RunnableEntityGroup.runnableEntity	runnableEntity.contextSwComponentPrototype, runnableEntity.targetRunnableEntity, runnableEntity.variationPoint.shortLabel
RunnableEntityGroup.runnableEntityGroup	runnableEntityGroup.contextSwComponentPrototype, runnableEntityGroup.targetRunnableEntityGroup, runnableEntityGroup.variationPoint.shortLabel
SenderComSpec.networkRepresentation	networkRepresentation
ServiceDependency.assignedDataType	assignedDataType, assignedDataType.variationPoint.shortLabel
SubElementMapping.firstElement	firstElement, firstElement.variationPoint.shortLabel
SubElementMapping.secondElement	secondElement, secondElement.variationPoint.shortLabel
SupervisedEntityNeeds.checkpoints	checkpoints.supervisedEntityCheckpointNeeds, checkpoints.variationPoint.shortLabel
SwcBswMapping.runnableMapping	runnableMapping, runnableMapping.variationPoint.shortLabel
SwcBswMapping.synchronizedModeGroup	synchronizedModeGroup, synchronizedModeGroup.variationPoint.shortLabel
SwcBswMapping.synchronizedTrigger	synchronizedTrigger, synchronizedTrigger.variationPoint.shortLabel
SwcImplementation.perInstanceMemorySize	perInstanceMemorySize, perInstanceMemorySize.variationPoint.shortLabel
SwcInternalBehavior.arTypedPerInstanceMemory	arTypedPerInstanceMemory.shortName, arTypedPerInstanceMemory.variationPoint.shortLabel
SwcInternalBehavior.event	event.shortName, event.variationPoint.shortLabel
SwcInternalBehavior.exclusiveAreaPolicy	exclusiveAreaPolicy, exclusiveAreaPolicy.variationPoint.shortLabel
SwcInternalBehavior.explicitInterRunnableVariable	explicitInterRunnableVariable.shortName, explicitInterRunnableVariable.variationPoint.shortLabel
SwcInternalBehavior.implicitInterRunnableVariable	implicitInterRunnableVariable.shortName, implicitInterRunnableVariable.variationPoint.shortLabel
SwcInternalBehavior.includedDataTypeSet	includedDataTypeSet
SwcInternalBehavior.includedModeDeclarationGroupSet	includedModeDeclarationGroupSet
SwcInternalBehavior.instantiationDataDefProps	instantiationDataDefProps, instantiationDataDefProps.variationPoint.shortLabel
SwcInternalBehavior.perInstanceMemory	perInstanceMemory.shortName, perInstanceMemory.variationPoint.shortLabel

▽

△

Name of splitable element	*Splitkey*
SwcInternalBehavior.perInstanceParameter	perInstanceParameter.shortName, perInstance Parameter.variationPoint.shortLabel
SwcInternalBehavior.portAPIOption	portAPIOption, portAPIOption.variationPoint.short Label
SwcInternalBehavior.runnable	runnable.shortName, runnable.variationPoint.short Label
SwcInternalBehavior.serviceDependency	serviceDependency.shortName, service Dependency.variationPoint.shortLabel
SwcInternalBehavior.sharedParameter	sharedParameter.shortName, shared Parameter.variationPoint.shortLabel
SwcInternalBehavior.variationPointProxy	variationPointProxy.shortName
SwComponentDocumentation.chapter	chapter.shortName, chapter.variationPoint.short Label
SwComponentType.consistencyNeeds	consistencyNeeds.shortName, consistency Needs.variationPoint.shortLabel
SwComponentType.port	port.shortName, port.variationPoint.shortLabel
SwComponentType.portGroup	portGroup.shortName, portGroup.variation Point.shortLabel
SwComponentType.swComponentDocumentation	swComponentDocumentation, swComponent Documentation.variationPoint.shortLabel
SwcServiceDependency.assignedData	assignedData, assignedData.variationPoint.short Label
SwcServiceDependency.assignedPort	assignedPort, assignedPort.variationPoint.short Label
SwPointerTargetProps.swDataDefProps	swDataDefProps
SwServiceArg.swDataDefProps	swDataDefProps
SwSystemconst.swDataDefProps	swDataDefProps
SwValues.vtf	vtf, vtf.variationPoint.shortLabel

Table H.1: Usage of splitable elements

I Variation Points in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpVariation≫ in the scope of this document.

Each entry in the table consists of the identification of the model element itself and the applicable value of the tagged value vh.latestBindingTime.

For more information about the concept of variation points and how model elements that contain variation points shall be treated please refer to [11].

Variation Point	***Latest Binding Time***
ApplicationArrayElement.maxNumberOfElements	preCompileTime
ApplicationRecordDataType.element	preCompileTime
ARPackage.arPackage	blueprintDerivationTime
ARPackage.element	systemDesignTime
ArrayValueSpecification.element	preCompileTime
AtomicSwComponentType.internalBehavior	preCompileTime
BulkNvDataDescriptor.nvBlockDataMapping	preCompileTime
CalibrationParameterValueSet.calibrationParameterValue	preCompileTime
ClientServerInterface.operation	blueprintDerivationTime
ClientServerOperation.argument	blueprintDerivationTime
CompositionSwComponentType.component	postBuild
CompositionSwComponentType.connector	postBuild
CompositionSwComponentType.instantiationRTEEventProps	codeGenerationTime
CompuConstFormulaContent.vf	codeGenerationTime
CompuNominatorDenominator.v	preCompileTime
CompuScale.lowerLimit	preCompileTime
CompuScale.upperLimit	preCompileTime
CompuScales.compuScale	blueprintDerivationTime
ConsistencyNeeds.dpgDoesNotRequireCoherency	preCompileTime
ConsistencyNeeds.dpgRequiresCoherency	preCompileTime
ConsistencyNeeds.regDoesNotRequireStability	preCompileTime
ConsistencyNeeds.regRequiresStability	preCompileTime
DataPrototypeGroup.dataPrototypeGroup	preCompileTime
DataPrototypeGroup.implicitDataAccess	preCompileTime
DiagEventDebounceCounterBased.counterDecrementStepSize	preCompileTime
DiagEventDebounceCounterBased.counterFailedThreshold	preCompileTime
DiagEventDebounceCounterBased.counterIncrementStepSize	preCompileTime
DiagEventDebounceCounterBased.counterJumpDown	preCompileTime
DiagEventDebounceCounterBased.counterJumpDownValue	preCompileTime
DiagEventDebounceCounterBased.counterJumpUp	preCompileTime
DiagEventDebounceCounterBased.counterJumpUpValue	preCompileTime
DiagEventDebounceCounterBased.counterPassedThreshold	preCompileTime
DiagEventDebounceTimeBased.timeBasedFdcThresholdStorageValue	preCompileTime
DiagEventDebounceTimeBased.timeFailedThreshold	preCompileTime
DiagEventDebounceTimeBased.timePassedThreshold	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
EndToEndProtection.endToEndProtectionISignalIPdu	preCompileTime
EndToEndProtection.endToEndProtectionVariablePrototype	preCompileTime
EndToEndProtectionSet.endToEndProtection	preCompileTime
ErrorTracerNeeds.tracedFailure	preCompileTime
ExecutableEntity.canEnter	preCompileTime
ExecutableEntity.runsInside	preCompileTime
Implementation.buildActionManifest	codeGenerationTime
Implementation.generatedArtifact	preCompileTime
Implementation.requiredArtifact	preCompileTime
Implementation.requiredGeneratorTool	preCompileTime
ImplementationDataType.subElement	preCompileTime
ImplementationDataTypeElement.arraySize	preCompileTime
ImplementationDataTypeElement.subElement	preCompileTime
InternalBehavior.constantMemory	preCompileTime
InternalBehavior.exclusiveArea	preCompileTime
InternalBehavior.exclusiveAreaNestingOrder	preCompileTime
InternalBehavior.staticMemory	preCompileTime
InternalConstrs.lowerLimit	preCompileTime
InternalConstrs.upperLimit	preCompileTime
ModeDeclarationGroup.modeDeclaration	blueprintDerivationTime
NumericalOrText.vf	preCompileTime
NumericalValueSpecification.value	preCompileTime
NvBlockDescriptor.clientServerPort	preCompileTime
NvBlockDescriptor.instantiationDataDefProps	preCompileTime
NvBlockDescriptor.modeSwitchEventTriggeredActivity	preCompileTime
NvBlockDescriptor.nvBlockDataMapping	preCompileTime
NvBlockSwComponentType.bulkNvDataDescriptor	preCompileTime
NvBlockSwComponentType.nvBlockDescriptor	preCompileTime
ParameterSwComponentType.instantiationDataDefProps	preCompileTime
PerInstanceMemorySize.size	preCompileTime
PhysConstrs.lowerLimit	preCompileTime
PhysConstrs.upperLimit	preCompileTime
PortGroup.outerPort	preCompileTime
PortInterface.isService	blueprintDerivationTime
PortInterfaceMappingSet.portInterfaceMapping	blueprintDerivationTime
RapidPrototypingScenario.rptContainer	preCompileTime
ReceiverComSpec.maxDeltaCounterInit	preCompileTime
ReceiverComSpec.usesEndToEndProtection	preCompileTime
RecordValueSpecification.field	preCompileTime
RptContainer.byPassPoint	preCompileTime
RptContainer.rptContainer	preCompileTime
RptContainer.rptHook	preCompileTime
RuleArguments.vf	preCompileTime
RuleArguments.vtf	preCompileTime
RuleBasedValueSpecification.arguments	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
RunnableEntity.asynchronousServerCallResultPoint	preCompileTime
RunnableEntity.dataReadAccess	preCompileTime
RunnableEntity.dataReceivePointByArgument	preCompileTime
RunnableEntity.dataReceivePointByValue	preCompileTime
RunnableEntity.dataSendPoint	preCompileTime
RunnableEntity.dataWriteAccess	preCompileTime
RunnableEntity.externalTriggeringPoint	preCompileTime
RunnableEntity.internalTriggeringPoint	preCompileTime
RunnableEntity.modeAccessPoint	preCompileTime
RunnableEntity.modeSwitchPoint	preCompileTime
RunnableEntity.parameterAccess	preCompileTime
RunnableEntity.readLocalVariable	preCompileTime
RunnableEntity.serverCallPoint	preCompileTime
RunnableEntity.writtenLocalVariable	preCompileTime
RunnableEntityGroup.runnableEntity	preCompileTime
RunnableEntityGroup.runnableEntityGroup	preCompileTime
ScaleConstr.lowerLimit	preCompileTime
ScaleConstr.upperLimit	preCompileTime
SenderComSpec.usesEndToEndProtection	preCompileTime
ServiceDependency.assignedDataType	preCompileTime
SubElementMapping.firstElement	preCompileTime
SubElementMapping.secondElement	preCompileTime
SupervisedEntityNeeds.checkpoints	preCompileTime
SwAxisIndividual.swMaxAxisPoints	preCompileTime
SwAxisIndividual.swMinAxisPoints	preCompileTime
SwcBswMapping.runnableMapping	preCompileTime
SwcBswMapping.synchronizedModeGroup	preCompileTime
SwcBswMapping.synchronizedTrigger	preCompileTime
SwcImplementation.perInstanceMemorySize	preCompileTime
SwcInternalBehavior.arTypedPerInstanceMemory	preCompileTime
SwcInternalBehavior.event	preCompileTime
SwcInternalBehavior.exclusiveAreaPolicy	preCompileTime
SwcInternalBehavior.explicitInterRunnableVariable	preCompileTime
SwcInternalBehavior.implicitInterRunnableVariable	preCompileTime
SwcInternalBehavior.instantiationDataDefProps	preCompileTime
SwcInternalBehavior.perInstanceMemory	preCompileTime
SwcInternalBehavior.perInstanceParameter	preCompileTime
SwcInternalBehavior.portAPIOption	preCompileTime
SwcInternalBehavior.runnable	preCompileTime
SwcInternalBehavior.serviceDependency	preCompileTime
SwcInternalBehavior.sharedParameter	preCompileTime
SwComponentDocumentation.chapter	postBuild
SwComponentType.consistencyNeeds	preCompileTime
SwComponentType.port	preCompileTime
SwComponentType.portGroup	preCompileTime

▽

△

Variation Point	*Latest Binding Time*
SwComponentType.swComponentDocumentation	preCompileTime
SwcServiceDependency.assignedData	preCompileTime
SwcServiceDependency.assignedPort	preCompileTime
SwDataDefProps	codeGenerationTime
SwDataDefProps.swValueBlockSize	preCompileTime
SwDataDefProps.swValueBlockSizeMult	preCompileTime
SwGenericAxisParam.vf	preCompileTime
SwTextProps.swMaxTextSize	preCompileTime
SwValues.vf	preCompileTime
SwValues.vtf	preCompileTime
TextTableMapping.bitfieldTextTableMaskFirst	preCompileTime
TextTableMapping.bitfieldTextTableMaskSecond	preCompileTime
TextTableValuePair.firstValue	preCompileTime
TextTableValuePair.secondValue	preCompileTime
ValueList.vf	preCompileTime

Table I.1: Usage of variation points