Document Title	Specification of Language Binding for modeled AP data types
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	994

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

Document Change History			
Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• Added C++ language binding of ApApplicationErrors • Moved all header file specification to chapter 8
2023-11-23	R23-11	AUTOSAR Release Management	• API Table generation completed • Editorial changes • Rewording of "Orthogonal" to "Outside" for better clarity
2022-11-24	R22-11	AUTOSAR Release Management	• Specifications added regarding the descriptions of Allocator Usages • Specifications added regarding the supported Encodings for Strings
2021-11-25	R21-11	AUTOSAR Release Management	• Initial release (previously part of [1])

Disclaimer

Contents

1 Introduction and functional overview

1.1 Adaptive Platform Data Types

The AUTOSAR data type model defined in [2] allows varying levels of granularity for specifying data types. The fundamentals of AUTOSAR data types are described in [3] chapter *"Data Types"* and further specialized for the Adaptive Platform (AP) in [4] chapter *"Data Type"*.

This specification is **not** concerned with ApplicationDataTypes, it is **only** concerned with concrete sub-classes of AbstractImplementationDataType - it is at this point in the data type model that the Language Binding is selected.

In general, the data types are used by typed sub-classes of PortInterface which model a particular function, e.g. ServiceInterface. Interface elements of these sub-classes of PortInterface may reference AutosarDataPrototypes, further typed by concrete sub-classes of AutosarDataTypes; specifically, as stated in [3] these are "Application" level and "Implementation" level data types.

Figure 1.1 shows on meta-model level the usage of AutosarDataPrototypes in Adaptive Platform Interfaces.

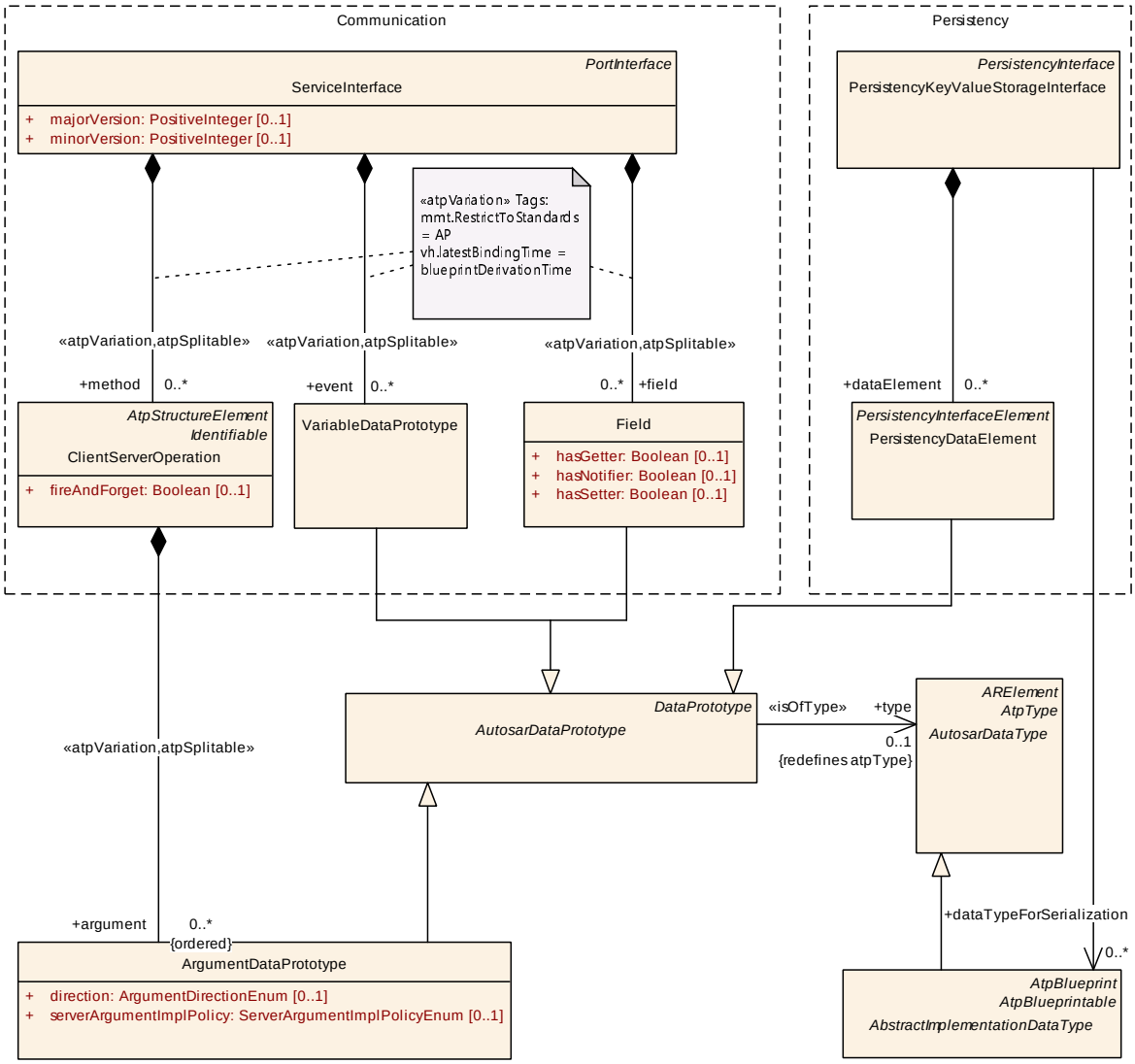

Figure 1.1: AUTOSAR data type usage in Adaptive Interfaces

1.2 Language Bindings

While the primary focus of the AP is targeted towards a C++ Language Binding (7.1), the chapter structure of the document allows for future versions to seamlessly insert "other" Language Bindings.

1.3 Methodology

This specification documents the generation/serialization[1] rules for transforming AP "modeled" Implementation Data Types to actual "language level" Data Types which can be processed by a compiler/interpreter of the bound language.

[1] the term "serialization" should not be mixed with (de-)serialization in the context of Communication

The general workflow step is described in *"Adaptive Software Generated Item"* in [5]; Figure 1.2 shows a very general workflow step for generation of data types from an Adaptive Platform Interface. Each "language specific" binding will have a "language specific" approach, and thus a respective chapter in this specification.

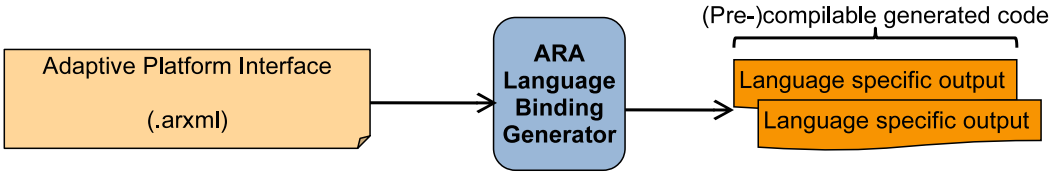

Figure 1.2: Methodology: Generic Language Binding generation

This specification is not concerned with the implementation details of an ARA Language Binding Generator, rather, the rules which an ARA Language Binding Generator must observe during generation/serialization.

[SWS_LBAP_00037] Principle of an ARA Language Binding Generator

Upstream requirements: RS_AP_00111

⌈The ARA Language Binding Generator is responsible for generating the Lanaguage Binding artifacts. These include data type declarations derived from the referenced AbstractImplementationDataTypes of the Adaptive Platform Interfaces.⌋

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations that are only relevant within this specification. A general list of acronyms and abbreviations is available in [6].

Abbreviation	Meaning
-	-

Table 2.1: Abbreviations used in the scope of this Document

Term	Meaning
Allocator	A language specific object responsible for (de-)allocation, (de-)initialization and ultimately limit impositions in memory/storage. C++ allocators must satisfy the requirements for an *Allocator* in ISO/IEC 14882 (version according to [RS_AP_00114]).
ARA Language Binding Generator	A workflow tool (e.g. a script) with the purpose to read/parse an ARXML model of data types in an `Adaptive Platform Interface` and generate a corresponding language specific representation thereof. Hereafter referred to as the **Generator**.
Adaptive Platform Interface	A typed (concrete) sub-class of `PortInterface` bound to the Adaptive Platform (in contrast to an "other" platform).
CppImplementationTypes Header File	A generated C++ header file created by an `ARA Language Binding Generator`.
C++ Bound Interface	An `Adaptive Platform Interface` which transitively references a `CppImplementationDataType` in it's usage (in contrast to an "other" language binding).
C++ Compound Type	See chapter *"Compound types"* in ISO/IEC 14882 (version according to [RS_AP_00114]).
C++ Fundamental Type	See chapter *"Fundamental types"* in ISO/IEC 14882 (version according to [RS_AP_00114]).
C++ Language Binding	A `Language Binding` in which the modeled representation is a `CppImplementationDataType` and the implementation language is C++.
Comparator	A language specific `Functor` responsible for binary comparison.
Functor	A language specific object which is treated as callable or executable. In C++ this is wrapped in std::function - ISO/IEC 14882 (version according to [RS_AP_00114])
Language Binding	A language binding is the point in which a representation on one side is selected (or bound) to a specific programming language on another side. In the context of this document a modeled representation is bound to a implementation language

Table 2.2: Terms used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Specification of Communication Management
AUTOSAR_AP_SWS_CommunicationManagement

[2] Meta Model
AUTOSAR_FO_MMOD_MetaModel

[3] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[4] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[5] Methodology for Adaptive Platform
AUTOSAR_AP_TR_Methodology

[6] Glossary
AUTOSAR_FO_TR_Glossary

[7] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[8] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[9] Specification of Platform Types for Adaptive Platform
AUTOSAR_AP_SWS_PlatformTypes

[10] ISO/IEC 14882:2014, Information technology – Programming languages – C++
https://www.iso.org

3.2 Further applicable specification

AUTOSAR provides a core specification [7] which is also applicable for this functional cluster. The chapter "General requirements for all Functional Clusters" of [7] shall be considered an additional and required specification for implementing this functional cluster.

4 Constraints and assumptions

4.1 Known limitations

- Although future versions of this specification may add further Language Bindings, the primary focus of the AP (and therefore this specification) is a binding to the C++ language.

5 Dependencies to other Functional Clusters

LBAP is not an AUTOSAR Functional Cluster (FC) and therefore has no dependencies to other FCs.

6 Requirements Tracing

The following tables reference requirements specified in [8] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement, this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00111]	Source Code Portability Support	[SWS_LBAP_00002] [SWS_LBAP_00003] [SWS_LBAP_00037]
[RS_AP_00114]	C++ interface shall be compatible with C++14	[SWS_LBAP_00005] [SWS_LBAP_00006] [SWS_LBAP_00008] [SWS_LBAP_00010] [SWS_LBAP_00011] [SWS_LBAP_00012] [SWS_LBAP_00013] [SWS_LBAP_00015] [SWS_LBAP_00017] [SWS_LBAP_00018] [SWS_LBAP_00023] [SWS_LBAP_00024] [SWS_LBAP_00026] [SWS_LBAP_00027] [SWS_LBAP_00028] [SWS_LBAP_00035] [SWS_LBAP_00047] [SWS_LBAP_00048] [SWS_LBAP_00049] [SWS_LBAP_00051] [SWS_LBAP_00052] [SWS_LBAP_00053] [SWS_LBAP_00063] [SWS_LBAP_00064]
[RS_AP_00116]	Header file name	[SWS_LBAP_00033] [SWS_LBAP_00050]
[RS_AP_00119]	Return values / application errors	[SWS_LBAP_00063] [SWS_LBAP_00064]
[RS_AP_00120]	Method and Function names	[SWS_LBAP_00054] [SWS_LBAP_00055] [SWS_LBAP_00057] [SWS_LBAP_00059] [SWS_LBAP_00060] [SWS_LBAP_00061] [SWS_LBAP_00062]
[RS_AP_00121]	Parameter names	[SWS_LBAP_00055] [SWS_LBAP_00057]
[RS_AP_00122]	Type names	[SWS_LBAP_00005] [SWS_LBAP_00056] [SWS_LBAP_00058]
[RS_AP_00127]	Usage of ara::core types	[SWS_LBAP_00008] [SWS_LBAP_00012] [SWS_LBAP_00013] [SWS_LBAP_00015] [SWS_LBAP_00016] [SWS_LBAP_00017] [SWS_LBAP_00018] [SWS_LBAP_00023] [SWS_LBAP_00024] [SWS_LBAP_00048] [SWS_LBAP_00056] [SWS_LBAP_00058] [SWS_LBAP_00063] [SWS_LBAP_00064]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment	[SWS_LBAP_00054] [SWS_LBAP_00055] [SWS_LBAP_00056] [SWS_LBAP_00057] [SWS_LBAP_00058] [SWS_LBAP_00059] [SWS_LBAP_00060] [SWS_LBAP_00061] [SWS_LBAP_00062]
[RS_AP_00136]	Usage of string types	[SWS_LBAP_00039] [SWS_LBAP_00040]

Table 6.1: Requirements Tracing

7 Functional Specification

LBAP is not an ARA Functional Cluster (FC) and therefore has no Functional Cluster behavior specification. Rather, in the following sub-chapters the serialization/binding rules are laid out certain model elements in a PortInterface are transformed to the respective language specific representation for use in ARA applications and FCs.

As explained in 1.1, AutosarDataTypes referenced by elements of any Adaptive Platform Interface, e.g.:

- ServiceInterface.event
- ServiceInterface.method
- ServiceInterface.field
- PersistencyKeyValueStorageInterface.dataElement

may be serialized/bound by a (generator/serializer) tool to an actual language bound compilable[1](or as near to as compilable as possible if they shall be further post-processed). The following sub-chapters specify the serialization rules for those Language Bindings supported by AUTOSAR.

7.1 C++ binding for CppImplementationDataTypes

This section describes the overall methodology and principles of the ARA Language Binding Generator for a binding to the C++ language; specifically, the version stated in [RS_AP_00114] specifies the C++ standards version for the AP.

In the context of this specification, any reference to C++ language level aspects, pertain to the ISO C++ standards version given by [RS_AP_00114].

Figure 7.1 shows the workflow steps for code generation for a C++ Language Binding, other languages may have other workflows.

This is a more detailed pictorial view of the high-level AP workflow step *"Adaptive Software Generated Item"* in [5] and thus the Language Binding generation would typically be done together with the *other* generations in the context of this workflow step.

[1]the term "compilable" is used generically here (use the term "interpretable" if the Language Binding implies an interpreter instead of a compiler)

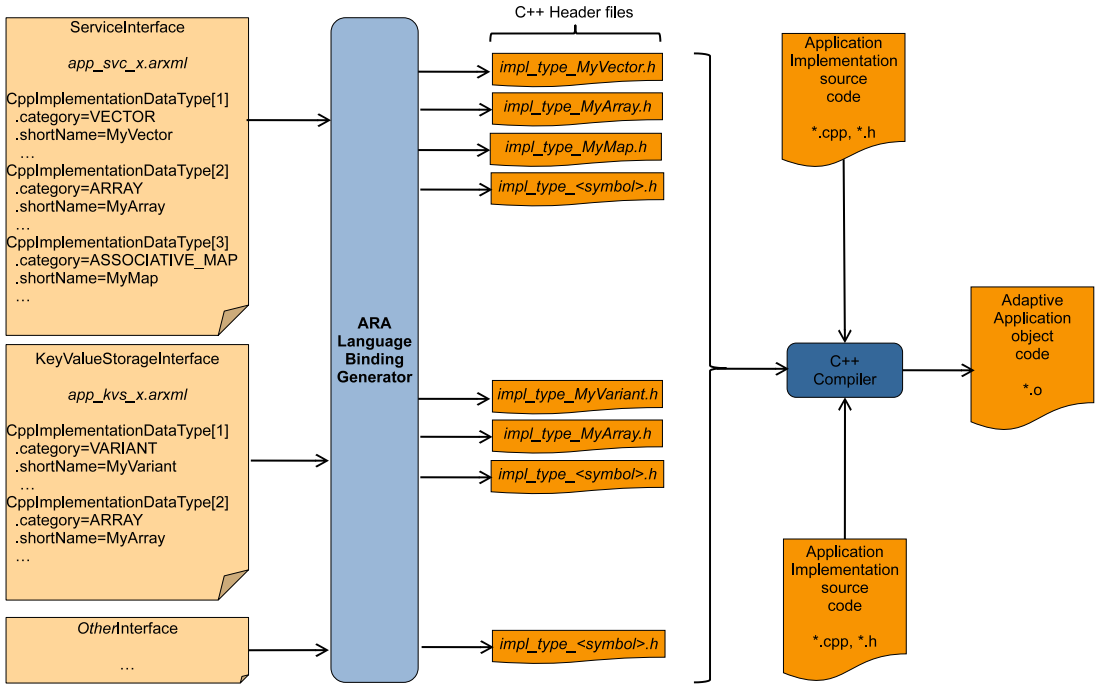

Figure 7.1: Methodology: C++ Language Binding generation

The attribute `typeEmitter` has an immediate direct influence on the behavior of the `ARA Language Binding Generator` i.e. whether generation shall take place or not.

[SWS_LBAP_00002] `ARA Language Binding Generator` usage of `typeEmitter`

Upstream requirements: RS_AP_00111

⌈The `ARA Language Binding Generator` shall generate a corresponding `C++ Language Binding` according to the rules defined in [TPS_MANI_01176], [TPS_MANI_01177] and [TPS_MANI_01212].⌋

[SWS_LBAP_00003] ARA generator rejection of symbol clashes

Upstream requirements: RS_AP_00111

⌈The `ARA Language Binding Generator` shall treat a potential symbol clash in a generated `Language Binding` as an error.⌋

A symbol clash results from a generated `Language Binding` containing > 1 C++ symbols in the same C++ namespace with same symbol name.

7.1.1 CppImplementationDataType

The basis for the C++ `Language Binding` is the C++ data type representation in [4] chapter *"CppImplementationDataType"*. The `CppImplementationDataType` is the

point in the AUTOSAR data type tree where the implementation of the data type becomes bound to the C++ language.

For the following sub-chapters, it is **essential** to have an understanding of the AUTOSAR data type model from the perspective of CppImplementationDataType shown here in Figure 7.2.

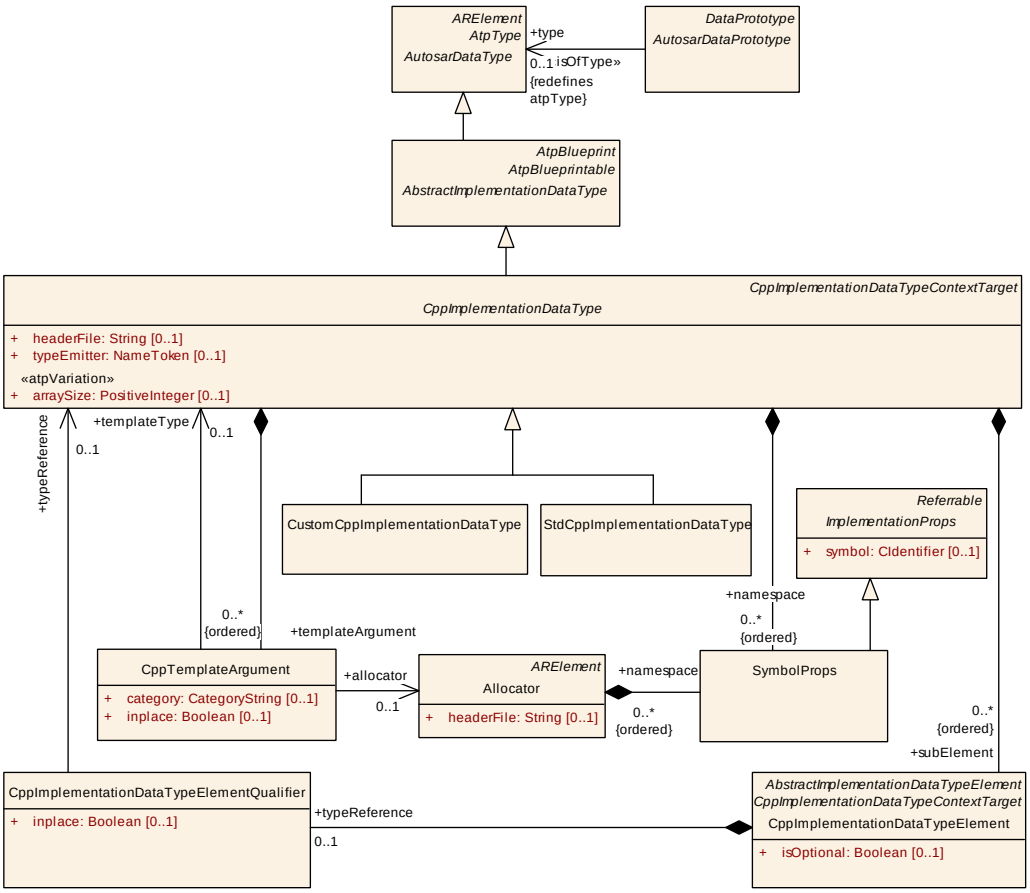

Figure 7.2: CppImplementationDataType

Further, [constr_1578] in [4] **must** be applied to all CppImplementationDataTypes in the following sub-chapters - this sets the necessary restriction of applicable category to CppImplementationDataType sub-element in the data type tree. The CppImplementationDataType is refined into two different sub-classes: StdCppImplementationDataType and CustomCppImplementationDataType and treated differently by the ARA Language Binding Generator.

7.1.1.1 StdCppImplementationDataType

The StdCppImplementationDataType is the basis for CppImplementationDataTypes, where the exact C++ serialization shall be provided by an AUTOSAR defined code implementation in [7].

7.1.1.1.1 Header File Generation

Note: [SWS_LBAP_00033] obviously makes sense for C++ Compound Types, but it is accepted that this rule may be relaxed for simple types which resolve to C++ Fundamental Types, i.e. it makes less sense to create an own C++ header (.h) for a simple `using` declaration.

7.1.1.1.2 Primitive Data Type

A Primitive CppImplementationDataType is classified by the category attribute of the CppImplementationDataType set to VALUE.

Models of Primitive CppImplementationDataType should conform to [TPS_MANI_03192] in [4].

[SWS_LBAP_00005] Standardized Primitive CppImplementationDataTypes

Upstream requirements: RS_AP_00114, RS_AP_00122

⌈The StdCppImplementationDataType of category=VALUE is allowed to have one of the following shortNames:

- int8_t : see [SWS_APT_00001] in [9],
- int16_t : see [SWS_APT_00004] in [9],
- int32_t : see [SWS_APT_00007] in [9],
- int64_t : see [SWS_APT_00010] in [9],
- uint8_t : see [SWS_APT_00022] in [9],
- uint16_t: see [SWS_APT_00025] in [9],
- uint32_t: see [SWS_APT_00028] in [9],
- uint64_t: see [SWS_APT_00031] in [9],
- bool : see [SWS_APT_00049] in [9],
- float : see [SWS_APT_00043] in [9],
- double : see [SWS_APT_00046] in [9],

⌋

Since only a defined set of StdCppImplementationDataTypes with category=VALUE are supported, the primitive C++ data types float, bool and double are supported in addition to chosen fixed width integer types defined in the C++ standard library header <cstdint>.

[SWS_LBAP_00006] `Primitive CppImplementationDataType` fixed width integers

Upstream requirements: RS_AP_00114

⌈If a `StdCppImplementationDataType` with the `category`=`VALUE` is referenced in a `C++ Bound Interface`, the C++ standard library header `<cstdint>` shall be included if the `StdCppImplementationDataType` has one of the following `shortName`s:

- `int8_t`
- `int16_t`
- `int32_t`
- `int64_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`
- `uint64_t`

⌋

7.1.1.1.3 String Data Type

API specified in 8.2.2.

7.1.1.1.3.1 String Encoding

Since the usage of `ApplicationDataType`s is not mandatory in AUTOSAR, it is necessary to stipulate the language binding behavior in both cases, where:

- `ApplicationDataType`s are used: [SWS_LBAP_00039]
- `ApplicationDataType`s are NOT used: [SWS_LBAP_00040]

It should be noted: the encoding scheme used for the language binding is independent of the configured encoding scheme for the network binding.

[SWS_LBAP_00039] Encoding of strings with a `baseTypeEncoding`

Upstream requirements: RS_AP_00136

⌈For a `StdCppImplementationDataType`.`category`==`STRING` with a corresponding `ApplicationDataType`.`category`==`STRING` mapped via a `DataTypeMap` and

where that ApplicationDataType has a baseTypeEncoding=UTF-8, the generated string shall explicitly contain a UTF-8 encoding.⌋

[SWS_LBAP_00040] Encoding of strings without a baseTypeEncoding

Upstream requirements: RS_AP_00136

⌈For a StdCppImplementationDataType of category==STRING with no corresponding ApplicationDataType with category=STRING mapped via a DataTypeMap, the generated string shall assume to contain the platform specific character encoding of UTF-8.⌋

7.1.1.2 CustomCppImplementationDataType

The CustomCppImplementationDataType facilitates the usage of existing data type definitions that are taken as the basis for a C++ Language Binding. When processing a CustomCppImplementationDataType, instead of actually generating the "standard" language binding as with StdCppImplementationDataType, the generator shall defer to use a pre-existing implementation, identified by: a C++ header file, C++ namespace and C++ symbol identifier. See chapter 8.2.10.

API specified in 8.2.10.

7.2 C++ binding for ApApplicationErrors

The language binding for ApApplicationErrors and their ApApplicationErrorDomains utilizes the same AUTOSAR API as the ara::core error domain - i.e. [SWS_CORE_00110]. Similar to the language binding for CppImplementationDataType, the language binding for ApApplicationErrors and their ApApplicationErrorDomains shall be generated based partially on content from a PortInterface model, in particular the ClientServerOperations (fireAndForget==FALSE) referenced in the role method. In the example model in 7.3, all application error model actors and their relationships are shown. The example uses a ServiceInterface with modeled ApApplicationErrors, but the principle is the same with any sub-class of PortInterface.

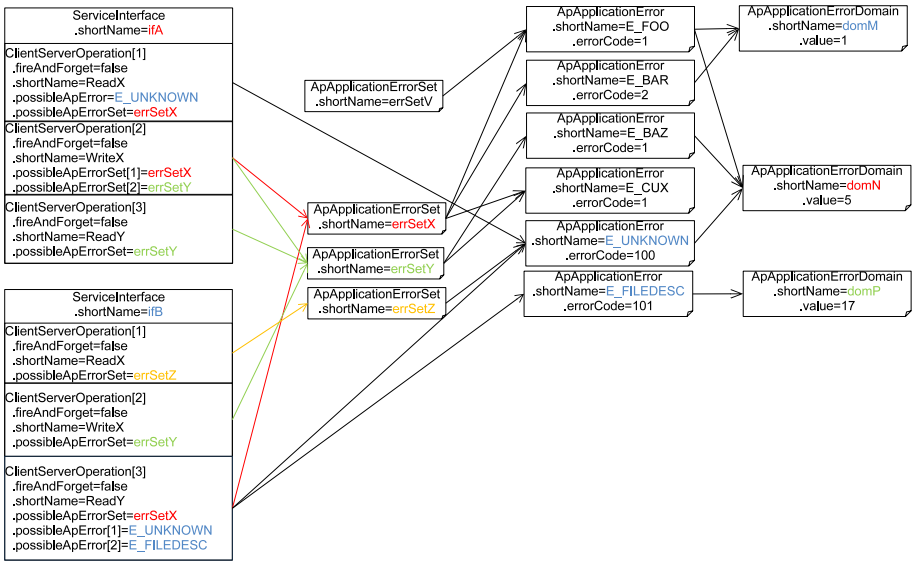

Figure 7.3: ApApplicationError - Model View

The workflow for a generator to process the `ApApplicationError`s is listed here:

1. The generator shall collect and iterate over all `ApApplicationError`s, (transitively) referenced either by:

   - `ClientServerOperation.possibleApError` **or**
   - `ClientServerOperation.possibleApErrorSet[].apApplicationError`

   which reference a `ApApplicationErrorDomain` in the role `errorDomain`.

2. For each referenced `ApApplicationErrorDomain`, an application error domain header file shall be created with the full header file API as per 7.2

3. In the final language binding:

   (a) A C++ header file shall exist for each `ApApplicationErrorDomain` with the domain identifier `value` ([SWS_LBAP_00050])

   (b) A C++ enumeration with the list of all `ApApplicationError`s + `errorCode`s shall exist ([SWS_LBAP_00052]). It is permitted to have $1+$ enumerators with the same value.

   (c) `ApApplicationErrorSet` semantics are not present (are dropped by the generator).

   (d) If the same `ApApplicationError.shortName` is (transitively) referenced $1+$ times via:

   - `ClientServerOperation.possibleApError` **and**
   - `ClientServerOperation.possibleApErrorSet[].apApplicationError`

only the first occurrence shall be present in the C++ enumeration. Remaining occurrences shall be dropped by the generator.

An example output is shown in 7.4. The precise processing rules for `ApApplicationError`s and the generated API are detailed in this chapter.

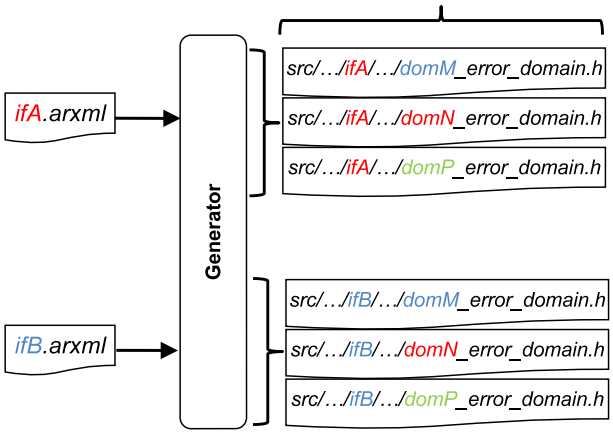

Figure 7.4: ApApplicationError - Example Language Binding

8 API specification

8.1 Header: {<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h

[SWS_LBAP_00050] Definition of Header File {<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h

Upstream requirements: RS_AP_00116

⌈

Kind:	Header File	
Syntax:	{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h	
Description:	For each modeled ApApplicationErrorDomain a C++ header file shall be generated according to this directory path/file name. A multiple inclusion guard shall be inserted around the whole header file as per [SWS_CORE_90002]	
Descriptors:	{<aaed-dir>}	The directory path as given by SymbolProps aggregated in the role ApApplicationErrorDomain. namespace. For each namespace in the **ordered** list: namespace[N+1] shall be an child directory of namespace[N] converted to lower-case.
	{ <aaed-file-prefix> }	The ApApplicationErrorDomain.shortName converted to lower-case
	{<aaed-sn-uc>}	The ApApplicationErrorDomain.shortName converted to upper-case
Example:	// File=n/n_plus_1/n_plus_2/{<aaed-file-prefix>}_error_domain.h #ifndef N_NPLUS1_NPLUS2_{<aaed-sn-uc>}_ERROR_DOMAIN_H_ #define N_NPLUS1_NPLUS2_{<aaed-sn-uc>}_ERROR_DOMAIN_H_ #include ".../path/to/{<aaed-file-prefix>}_error_domain.h" ... #endif // N_NPLUS1_NPLUS2_{<aaed-sn-uc>}_ERROR_DOMAIN_H_	

⌋

8.1.1 Namespaces

8.1.1.1 {<aaed-ns>}

[SWS_LBAP_00051] Definition of Namespace {<aaed-ns>}

Upstream requirements: RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"
Scope:	--

▽

△

Syntax:	`namespace {<aaed-ns>}`	
Description:	Namespace hierarchy of the `ApApplicationErrorDomain`	
Descriptors:	`{<aaed-ns>}`	The C++ namespace hierarchy as given by `SymbolProps` aggregated in the role `ApApplicationErrorDomain.namespace`. For each `namespace` in the **ordered** list: `namespace`[N+1] shall be an inner namespace of `namespace`[N] converted to lower-case.

Example:	`...`
	`namespace n {`
	`namespace n_plus_1 {`
	`namespace n_plus_2 { ... }`
	`}`
	`}`
	`...`	

⌋

8.1.2 Non-Member Types

8.1.2.1 Enumeration: {<aaed-sn>}Errc

[SWS_LBAP_00052] Definition of API enum `{<aaed-ns>}::{<aaed-sn>}Errc`

Upstream requirements: RS_AP_00114

⌈

Kind:	enumeration	
Header file:	#include "`{<aaed-dir>}/{<aaed-file-prefix>}`_error_domain.h"	
Scope:	`namespace {<aaed-ns>}`	
Symbol:	`{<aaed-sn>}`Errc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class {<aaed-sn>}Errc : ara::core::ErrorDomain::CodeType {...};`	
Values:	`{<aae-list>}`	--
Description:	Defines the list of application error codes for the domain	
Descriptors:	`{<aaed-ns>}`	As per `{<aaed-ns>}` in [SWS_LBAP_00051]
	`{<aaed-sn>}`	The `ApApplicationErrorDomain.shortName`
	`{<aae-list>}`	**Shown as "..." in Syntax**. The list of enumerations/error codes as given by: • `ClientServerOperation.possibleApError.errorCode`s • `ClientServerOperation.possibleApErrorSet[].apApplicationError.errorCode`s For each enumerator in the list, [SWS_LBAP_00053] applies.

⌋

8.1.3 Global Variables

8.1.3.1 {<symbol-aae-sn>}

[SWS_LBAP_00053] Definition of API variable {<aaed-ns>}::{<symbol-aae-sn>}

Upstream requirements: RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	namespace {<aaed-ns>}	
Symbol:	{<symbol-aae-sn>}	
Type:	--	
Syntax:	{<symbol-aae-sn>} = {<aae-ec>};	
Description:	For each error code in {<aae-list>} in [SWS_LBAP_00052] there shall exist a C++ enumerator declaration.	
Descriptors:	{<aaed-ns>}	As per {<aaed-ns>} in [SWS_LBAP_00051]
	{<symbol-aae-sn>}	The error code literal, given by ApApplicationError.shortName
	{<aae-ec>}	The error code value, given by ApApplicationError.errorCode.
See also:	[TPS_MANI_01190], [constr_10132], [TPS_MANI_01191], [constr_1664], [constr_1665]	

⌋

8.1.4 Non-Member Functions

8.1.4.1 Other

8.1.4.1.1 Get{<aaed-sn>}ErrorDomain

[SWS_LBAP_00054] Definition of API function {<aaed-ns>}::Get{<aaed-sn>}ErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130

⌈

Kind:	function
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"
Scope:	namespace {<aaed-ns>}
Syntax:	constexpr const ara::core::ErrorDomain & Get{<aaed-sn>}ErrorDomain () noexcept;

▽

△

Return value:	const ara::core::Error Domain &	Reference to the `aaedns::aaedsnErrorDomain` object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns a reference to the `aaedns::aaedsnErrorDomain` object	

⌋

8.1.4.1.2 MakeErrorCode

[SWS_LBAP_00055] Definition of API function `{<aaed-ns>}`::MakeErrorCode

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "`{<aaed-dir>}/{<aaed-file-prefix>}`_error_domain.h"	
Scope:	`namespace {<aaed-ns>}`	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode ({<aaed-ns>}::{<aaed-sn>}Errc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error
Return value:	ara::core::ErrorCode	An `ara::core::ErrorCode` object
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Creates an instance of `ara::core::ErrorCode`	

⌋

8.1.5 Class: {<aaed-sn>}ErrorDomain

[SWS_LBAP_00058] Definition of API class `{<aaed-ns>}`::`{<aaed-sn>}`ErrorDomain

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class
Header file:	#include "`{<aaed-dir>}/{<aaed-file-prefix>}`_error_domain.h"
Scope:	`namespace {<aaed-ns>}`

▽

△

Symbol:	{<aaed-sn>}ErrorDomain	
Base class:	ara::core::ErrorDomain	
Syntax:	`class {<aaed-sn>}ErrorDomain final : public ara::core::ErrorDomain {...};`	
Unique ID:	The domain identifier, given by ApApplicationErrorDomain.value	
Description:	Class representing an ApApplicationErrorDomain	
Descriptors:	{<aaed-sn>}	As per {<aaed-sn>} in [SWS_LBAP_00052]

⌋

8.1.5.1 Public Member Types

8.1.5.1.1 Type Alias: Errc

[SWS_LBAP_00059] Definition of API type {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Errc

Upstream requirements: RS_AP_00120, RS_AP_00130

⌈

Kind:	type alias	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain	
Symbol:	Errc	
Syntax:	`using Errc = {<aaed-sn>}Errc;`	
Description:	Alias for the error code value enumeration	
Descriptors:	{<aaed-sn>}	As per {<aaed-sn>} in [SWS_LBAP_00052]

⌋

8.1.5.1.2 Type Alias: Exception

[SWS_LBAP_00060] Definition of API type {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Exception

Upstream requirements: RS_AP_00120, RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain
Symbol:	Exception
Syntax:	using Exception = {<aaed-sn>}Exception;
Description:	Alias for the exception base class

⌋

8.1.5.2 Public Member Functions

8.1.5.2.1 Special Member Functions

8.1.5.2.1.1 Default Constructor

[SWS_LBAP_00061] Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::{<aaed-sn>}ErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130

⌈

Kind:	function
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain
Syntax:	{<aaed-sn>}ErrorDomain ()=delete;
Description:	Constructs a new {<aaed-sn>}ErrorDomain object

⌋

8.1.5.2.2 Member Functions

8.1.5.2.2.1 Message

[SWS_LBAP_00063] Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Message

Upstream requirements: RS_AP_00114, RS_AP_00119, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain	
Syntax:	const char * Message (CodeType errorCode) const noexcept override;	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the message associated with the error code	
See also:	[RS_CM_00211]	

⌋

8.1.5.2.2.2 Name

[SWS_LBAP_00062] Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Name

Upstream requirements: RS_AP_00120, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain	
Syntax:	const char * Name () const noexcept override;	
Return value:	const char *	A string constant associated with the aaedns::aaedsnErrorDomain
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Retrieve the name of the error domain	

⌋

8.1.5.2.2.3 ThrowAsException

[SWS_LBAP_00064] Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00114, RS_AP_00119, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	class {<aaed-ns>}::{<aaed-sn>}ErrorDomain	
Syntax:	void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throws the exception associated with the error code.	
See also:	[RS_CM_00211]	

⌋

8.1.6 Class: {<aaed-sn>}Exception

[SWS_LBAP_00056] Definition of API class {<aaed-ns>}::{<aaed-sn>}Exception

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class	
Header file:	#include "{<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h"	
Scope:	namespace {<aaed-ns>}	
Symbol:	{<aaed-sn>}Exception	
Base class:	ara::core::Exception	
Syntax:	class {<aaed-sn>}Exception : public ara::core::Exception {...};	
Description:	Defines a class for exceptions to be thrown by the API	
Descriptors:	{<aaed-sn>}	As per {<aaed-sn>} in [SWS_LBAP_00052]

⌋

8.1.6.1 Public Member Functions

8.1.6.1.1 Constructors

8.1.6.1.1.1 {<aaed-sn>}Exception

[SWS_LBAP_00057] Definition of API function `{<aaed-ns>}::{<aaed-sn>}Exception::{<aaed-sn>}Exception`

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "`{<aaed-dir>}/{<aaed-file-prefix>}`_error_domain.h"	
Scope:	`class {<aaed-ns>}::{<aaed-sn>}Exception`	
Syntax:	`explicit {<aaed-sn>}Exception (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Constructs a new `{<aaed-sn>}`Exception containing an `ara::core::ErrorCode`	
Descriptors:	`{<aaed-sn>}`	As per `{<aaed-sn>}` in [SWS_LBAP_00052]

⌋

8.2 Header: {<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h

[SWS_LBAP_00033] `CppImplementationDataType`s Header Files: file name and multiple inclusion guard

Upstream requirements: RS_AP_00116

⌈

Kind:	Header File	
Syntax:	`{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h`	
Description:	The generator shall construct: • The path/file name of each `CppImplementationTypes Header File` accordingly. • A multiple inclusion guard around the whole header file in each `CppImplementationTypes Header File`.	
Descriptors:	`{<ns-derived-dir-path>}`	as per [SWS_LBAP_00035] whereby: for each inner namespace in the hierarchy, an inner directory shall be created to contain the header file

▽

△

	{<shortname-lower>}	CppImplementationDataType. shortName converted to lower-case.
	{<ns-derived-include>}	relative path of the CppImplementationTypes Header File according to {<ns-derived-dir-path>} up to but omitting the file extension, with all path components separated by an underscore, converted to upper-case.
Example:		

```
// Example where: {<shortname-lower>} = example
//          where: {<ns-derived-dir-path>} = n/n_plus_1/n_plus_2
//
// Filename: .../n/n_plus_1/n_plus_2/impl_type_example.h

#ifndef N_NPLUS1_NPLUS2_IMPL_TYPE_EXAMPLE_H_
#define N_NPLUS1_NPLUS2_IMPL_TYPE_EXAMPLE_H_
...
#endif // N_NPLUS1_NPLUS2_IMPL_TYPE_EXAMPLE_H_
```

See also:	[TPS_MANI_01309], [TPS_MANI_01168], [SWS_CORE_90002]

⌋

8.2.1 Namespaces

8.2.1.1 {<cppidt-ns-list-lc>}

[SWS_LBAP_00035] CppImplementationDataTypes Header Files namespace hierarchy

Upstream requirements: RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"
Scope:	--
Syntax:	namespace {<cppidt-ns-list-lc>}
Description:	The generator shall use the SymbolProps aggregated in the role CppImplementationDataType. namespace to construct the encapsulating C++ namespace hierarchy for the C++ data type inside the CppImplementationTypes Header File. For each namespace in the **ordered** list: namespace[N+1] shall be an inner namespace of namespace[N] converted to lower-case.
Example:	

```
...
  namespace n {
    namespace n_plus_1 {
      namespace n_plus_2 { ... }
    }
  }
...
```

See also:	[TPS_MANI_01168]

⌋

8.2.2 String Data Type

8.2.2.1 Type Alias: {<symbol-string>}

[SWS_LBAP_00015] `StdCppImplementationDataType. category ==STRING` without an `Allocator`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"
Scope:	namespace {<cppidt-ns-list-lc>}
Symbol:	{<symbol-string>}
Syntax:	using {<symbol-string>} = ara::core::String;
Description:	For each StdCppImplementationDataType. category ==STRING without an Allocator, there shall exist a C++ type alias. The storage is managed by the default allocator std::allocator [10].
Descriptors:	{<symbol-string>} — The symbol name of the type alias as given by CppImplementationDataType. shortName
Example:	`// Example: string allocator=FALSE` `using T_S = ara::core::String;`
See also:	[TPS_MANI_03179], [SWS_CORE_03001]

⌋

8.2.2.2 Type Alias: {<symbol-string-alloc>}

[SWS_LBAP_00016] `StdCppImplementationDataType. category ==STRING` with an `Allocator`

Upstream requirements: RS_AP_00127

⌈

Kind:	type alias
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"
Scope:	namespace {<cppidt-ns-list-lc>}
Symbol:	{<symbol-string-alloc>}
Syntax:	using {<symbol-string-alloc>} = ara::core::BasicString< {<fq-allocator>}<char> >;
Description:	For each StdCppImplementationDataType. category ==STRING with an Allocator, there shall exist a C++ type alias.

▽

△

Descriptors:	{<symbol-string-alloc>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<fq-allocator>}	Fully namespace-qualified signature of the Allocator where: • the C++ header file containing the allocator is given by Allocator. headerFile • the C++ namespace containng the allocator is given by Allocator. namespace • the symbol name of the struct/class which provides the allocator implementation is given by Allocator. shortName A type alias shall be generated for the allocator as per [SWS_LBAP_00047]. If the headerFile is not specified or does not exist, the generator shall terminate with an **error**. If the namespace is not specified, the generator shall terminate with an **error**.

Example:	`// Example: string, allocator=TRUE`
	`using T_BS = ara::core::BasicString<`
	`  ns1::OuterAllocator<char, 100>`
	`>;`	

See also:	[TPS_MANI_03188], [SWS_CORE_03000], [SWS_LBAP_00047]	

⌋

8.2.3 Array Data Type

8.2.3.1 Type Alias: {<symbol-array>}

[SWS_LBAP_00008] StdCppImplementationDataType. category ==ARRAY

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-array>}	
Syntax:	using {<symbol-array>} = ara::core::Array<{<containerized-type>}, {<max-num-elements>}>;	
Description:	For each StdCppImplementationDataType. category ==ARRAY, there shall exist a C++ type alias.	
Descriptors:	{<symbol-array>}	as per {<symbol-string>} in [SWS_LBAP_00015]

▽

△

	{<containerized-type>}	The containerized type given by CppImplementationDataType.templateArgument.templateType. If the CppImplementationDataType.templateArgument.templateType refers to a type which is the same as this owning CppImplementationDataType, it has the semantics of a nested (multi-dimensional) type, e.g. ARRAY of ARRAY, VECTOR of ARRAY or VECTOR of ARRAY of ASSOCIATIVE_MAP. There is no limit to the depth of such nested {<containerized-type>}s, but an overly deep use of inplace usually indicates a need for re-design due to over-complexity of generated code. • If CppTemplateArgument.inplace ==FALSE or is undefined, the CppImplementationDataType.templateType.shortName shall be used as the {<containerized-type>} and a **further** C++ type alias shall be generated in the same namespace scope as this C++ type alias where the CppImplementationDataType.templateType.shortName shall be the *identifier* and the {<containerized-type>} shall be the *type-id* as per [10]. • If CppTemplateArgument.inplace ==TRUE, the C++ data type representing the category of the CppImplementationDataType.templateType shall be generated as the {<containerized-type>} directly in-place.
	{<max-num-elements>}	Number of elements - defined by arraySize
Example:		

```
// Example: 1-dim. array<string>, inplace==TRUE, max-num-elements=5
using T_1DA_S_IPT = ara::core::Array< ara::core::String, 5 >;

// Example: 1-dimensional array<string>, inplace==FALSE
using T_1DA_S_IPF_T = ara::core::String;
using T_1DA_S_IPF = ara::core::Array< T_1DA_S_IPF_T, 5 >;

// Example: 3-dimensional array<string>
using T_3DA_S_IPT =
  ara::core::Array<          // inplace==TRUE, max-num-elements=5
    ara::core::Array<        // inplace==TRUE, max-num-elements=4
      ara::core::Array<      // inplace==TRUE, max-num-elements=3
        ara::core::String, 3
      >, 4
    >, 5
  >;

// Example: 3-dimensional array<string>, inplace==FALSE
using T_3DA_S_IPF_T3 = ara::core::String;
using T_3DA_S_IPF_T2 =
  ara::core::Array<T_3DA_S_IPF_T3, 25>;  // max-num-elements=25
using T_3DA_S_IPF_T1 =
  ara::core::Array<T_3DA_S_IPF_T2, 50>;  // max-num-elements=50
using T_3DA_S_IPF =
  ara::core::Array<T_3DA_S_IPF_T1, 100>; // max-num-elements=100
```

See also:	[TPS_MANI_03201], [SWS_CORE_01201], [TPS_MANI_03170], [TPS_MANI_03171], [TPS_MANI_03172], [TPS_MANI_03173], [constr_3433], [constr_1660], [SWS_CORE_01201]

⌋

8.2.4 Vector Data Type

8.2.4.1 Type Alias: {<symbol-vector>}

[SWS_LBAP_00017] `StdCppImplementationDataType.category ==VECTOR` without an `Allocator`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-vector>}	
Syntax:	using {<symbol-vector>} = ara::core::Vector<{<containerized-type>}>;	
Description:	For each StdCppImplementationDataType.category ==VECTOR without an Allocator, there shall exist a C++ type alias. The storage is managed by the default allocator std::allocator [10].	
Descriptors:	{<symbol-vector>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<containerized-type>}	as per {<containerized-type>} in [SWS_LBAP_00008]

Example:	// Example: 3-dim. vector<string>, inplace==FALSE, allocator=FALSE
	using T_3DV_S_IPF_T2 =
	  ara::core::Vector<ara::core::String>;
	using T_3DV_S_IPF_T1 =
	  ara::core::Vector<T_3DV_S_IPF_T2>;
	using T_3DV_S_IPF =
	  ara::core::Vector<T_3DV_S_IPF_T1>;
	
	// Example: 3-dim. vector<string>, inplace==TRUE, allocator=FALSE
	using T_3DV_S_IPT_AN =
	  ara::core::Vector<
	  ara::core::Vector<
	  ara::core::Vector<
	  ara::core::String
	  >
	  >
	  >;	

See also:	[TPS_MANI_03174], [TPS_MANI_03175], [TPS_MANI_03176], [TPS_MANI_03186], [TPS_MANI_03177], [TPS_MANI_03186], [SWS_CORE_01301]	

⌋

8.2.4.2 Type Alias: {<symbol-vector-alloc>}

[SWS_LBAP_00018] `StdCppImplementationDataType. category` ==VECTOR with an `Allocator`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-vector-alloc>}	
Syntax:	using {<symbol-vector-alloc>} = ara::core::Vector<{<containerized-type>}, {<fq-allocator>}<{<containerized-type>}> >;	
Description:	For each StdCppImplementationDataType. category ==VECTOR with an Allocator, there shall exist a C++ type alias.	
Descriptors:	{<symbol-vector-alloc>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<containerized-type>}	as per {<containerized-type>} in [SWS_LBAP_00008]
	{<fq-allocator>}	as per {<fq-allocator>} in [SWS_LBAP_00016]

Example:	`// Example: 3-dimensional vector<string>, inplace==TRUE`
	`using T_3DV_S_IPT_AX =`
	`  ara::core::Vector<  // allocator=FALSE`
	`  ara::core::Vector<  // allocator=TRUE, max-num-elements=100`
	`  ara::core::Vector<  // allocator=TRUE, max-num-elements=50`
	`  ara::core::String,`
	`  ns1::ns2::ns3::InnerAllocator< ara::core::String, 50 >`
	`  >,`
	`  ns1::OuterAllocator< ara::core::Vector<ara::core::String>, 100 >`
	`  >`
	`  >;`	

See also:	[TPS_MANI_03174], [TPS_MANI_03175], [TPS_MANI_03176], [TPS_MANI_03186], [SWS_CORE_01301], [TPS_MANI_03177]	

⌋

8.2.4.3 Type Alias: {<symbol-vector-alloc-maxsize>}

[SWS_LBAP_00048] `StdCppImplementationDataType.category` ==VECTOR with an `Allocator` and `arraySize`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-vector-alloc-maxsize>}	
Syntax:	using {<symbol-vector-alloc-maxsize>} = ara::core::Vector< {<containerized-type>}, {<fq-allocator-maxsize>}< {<containerized-type>}, {<max-num-elements>}> >;	
Description:	For each StdCppImplementationDataType.category ==VECTOR with an Allocator and arraySize, there shall exist a C++ type alias.	
Descriptors:	{<symbol-vector-alloc-maxsize>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<containerized-type>}	as per {<containerized-type>} in [SWS_LBAP_00008]
	{<fq-allocator>}	as per {<fq-allocator>} in [SWS_LBAP_00016]
	{<max-num-elements>}	as per {<max-num-elements>} in [SWS_LBAP_00008]
Example:		
See also:	[TPS_MANI_03174], [TPS_MANI_03175], [TPS_MANI_03176], [TPS_MANI_03186]	

```
// Example: 3-dimensional vector
using T_3DV_S_IPF_AX_T1 =
  ara::core::String;
using ALLOC_T_3DV_S_IPF_AX_T1 =
  ns1::ns2::ns3::InnerAllocator< T_3DV_S_IPF_AX_T1, 50 >;
using T_3DV_S_IPF_AX_T2 =  // inplace==FALSE, allocator=TRUE
  ara::core::Vector<       // max-num-elements=50
    T_3DV_S_IPF_AX_T1,
    ALLOC_T_3DV_S_IPF_AX_T1
  >;
using ALLOC_T_3DV_S_IPF_AX_T2 =
  ns1::OuterAllocator<
    ara::core::Vector<ara::core::String>,
    100
  >;
using T_3DV_S_IPF_AX_T3 =  // inplace==FALSE, allocator=TRUE
  ara::core::Vector<       //  max-num-elements=100
    T_3DV_S_IPF_AX_T2,
    ALLOC_T_3DV_S_IPF_AX_T2
  >;
using T_3DV_S_IPF_AX =     // inplace==FALSE, allocator=FALSE
  ara::core::Vector< T_3DV_S_IPF_AX_T3 >;
```

⌋

8.2.5 Structure Data Type

8.2.5.1 Struct: {<symbol-struct>}

[SWS_LBAP_00010] `StdCppImplementationDataType. category ==STRUCTURE`

Upstream requirements: RS_AP_00114

⌈

Kind:	struct	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-struct>}	
Syntax:	struct {<symbol-struct>} {...};	
Description:	For each StdCppImplementationDataType. category ==STRUCTURE, there shall exist a C++ POD struct declaration.	
Descriptors:	{<symbol-struct>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<struct-element-list>}	*Shown as ... in Syntax.* The list of ordered struct elements/ members given by CppImplementationDataType. subElement. For each subElement in the **ordered** list, either: • [SWS_LBAP_00011] shall be applied, if CppImplementationDataTypeElement. isOptional ==FALSE or undefined • [SWS_LBAP_00012] shall be applied, if CppImplementationDataTypeElement. isOptional ==TRUE
Example:	See SWS_LBAP_00012	
See also:	[TPS_MANI_03180], [TPS_MANI_03181], [constr_10417]	

⌋

8.2.5.2 {<symbol-struct-element>}

[SWS_LBAP_00011] `CppImplementationDataTypeElement. isOptional ==FALSE` or undefined

Upstream requirements: RS_AP_00114

⌈

Kind:	variable
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"
Scope:	namespace {<cppidt-ns-list-lc>}
Symbol:	{<symbol-struct-element>}

⌋

△

Type:	{<struct-element-type>}	
Syntax:	{<struct-element-type>} {<symbol-struct-element>};	
Description:	For each struct member/ subElement specified in [SWS_LBAP_00010] with CppImplementationDataTypeElement. isOptional ==FALSE or undefined, there shall exist a C++ struct element declaration.	
Descriptors:	{<struct-element-type>}	The data type of the struct element/member as given by CppImplementationDataTypeElement. typeReference. The reference CppImplementationDataTypeElement. typeReference. typeReference gives the 'actual' C++ data type which shall be generated to code. • If the CppImplementationDataTypeElement. typeReference. typeReference refers to a CppImplementationDataType. category ==STRUCTURE, it has the semantics of a nested C++ struct and [SWS_LBAP_00010] shall be applied. • If the CppImplementationDataTypeElement. typeReference. typeReference refers to a CppImplementationDataType. category !=STRUCTURE, the rules of {<containerized-type>} as per [SWS_LBAP_00008] shall apply. • If CppImplementationDataTypeElement. typeReference. inplace ==FALSE or is undefined, the C++ data type representing the CppImplementationDataTypeElement. typeReference. typeReference. shortName shall be used as the {<struct-element-type>} and a further C++ type alias shall be generated in the same namespace scope, but outside of this struct, where the CppImplementationDataTypeElement. typeReference. typeReference. shortName shall be the *identifier* and the {<struct-element-type>} shall be the *type-id*. • If CppImplementationDataTypeElement. typeReference. inplace ==TRUE, the C++ data type representing the CppImplementationDataTypeElement. typeReference. typeReference shall be generated as the {<struct-element-type>} directly in-place.
	{<symbol-struct-element>}	Symbol name of the struct element as given by CppImplementationDataTypeElement. shortName
Example:	See SWS_LBAP_00012	
See also:	[TPS_MANI_03180], [TPS_MANI_03181], [TPS_MANI_03196], [constr_10417], [constr_1659]	

⌋

8.2.5.3 {<symbol-struct-opt-element>}

[SWS_LBAP_00012] `CppImplementationDataTypeElement. isOptional ==TRUE`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	variable	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-struct-opt-element>}	
Type:	ara::core::Optional< {<struct-element-type>} >	
Syntax:	ara::core::Optional<{<struct-element-type>}> {<symbol-struct-opt-element>};	
Description:	For each struct member/ (subElement) specified in [SWS_LBAP_00010], with CppImplementationDataTypeElement. isOptional ==TRUE there shall exist a C++ struct element declaration. The combined usage of CppImplementationDataTypeElement. isOptional ==TRUE and CppImplementationDataTypeElement. typeReference. inplace ==TRUE is forbidden as per [constr_1708].	
Descriptors:	{<struct-element-type>}	as per [SWS_LBAP_00011]
	{<symbol-struct-opt-element>}	as per [SWS_LBAP_00011]

Example:	// Example: struct
	using T_S_TR3 =
	ara::core::Vector<ara::core::String>; // modelled TYPE_REF
	using T_S_TR2 = ara::core::String; // modelled TYPE_REF
	using T_S_IPX_T2 = ara::core::String; // generated
	struct T_S2 {
	T_S_IPX_T2 a; // inplace==FALSE
	T_S_TR2 b; // inplace==undef
	ara::core::Map< // inplace==TRUE
	std::uint8_t, // inplace==TRUE
	T_S_TR2 // inplace==undef
	> c;
	struct { // inplace==TRUE
	std::uint8_t s1; // inplace==TRUE
	T_S_TR3 s2; // inplace==undef
	} d;
	ara::core::Optional<T_S_TR2> e; // isOptional==TRUE
	};	

See also:	[TPS_MANI_03180], [TPS_MANI_03181], [TPS_MANI_03196], [constr_10417], [constr_1659], [constr_1708], [SWS_CORE_01033]	

⌋

8.2.6 Enumeration

8.2.6.1 Enumeration: {<symbol-enum>}

[SWS_LBAP_00027] Enumeration Data Type

Upstream requirements: RS_AP_00114

⌈

Kind:	enumeration	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-enum>}	
Underlying type:	{<enum-underlying-type>}	
Syntax:	enum class {<symbol-enum>} : {<enum-underlying-type>} {...};	
Values:	{<enumerator-list>}	--
Description:	For each: • StdCppImplementationDataType. category ==TYPE_REFERENCE which type-resolves to a • StdCppImplementationDataType. category ==VALUE, and that aggregates a • StdCppImplementationDataType. swDataDefProps. compuMethod. category ==TEXTTABLE there shall exist a C++ enum declaration.	
Descriptors:	{<symbol-enum>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<enum-underlying-type>}	The underlying integral base for the enum, given by the StdCppImplementationDataType. category ==VALUE after type-resolution has been applied to the referring StdCppImplementationDataType. category ==TYPE_REFERENCE
	{<enumerator-list>}	*Shown as "..." in Syntax.* The **ordered** list of enumerators as given by StdCppImplementationDataType. swDataDefProps. compuMethod. compuPhysToInternal. compuContent. compuScale. For each enumerator/ compuScale in the list, [SWS_LBAP_00028] shall be applied.
Example:	See SWS_LBAP_00028	
See also:	[TPS_MANI_03187], [TPS_SWCT_01276], [TPS_SWCT_01548], [TPS_SWCT_01278]	

⌋

8.2.6.2 {<symbol-enum-literal>}

[SWS_LBAP_00028] Enumeration Data Type - enumerators

Upstream requirements: RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-enum-literal>}	
Type:	--	
Syntax:	{<symbol-enum-literal>} = {<enum-initializer>} {<enum-literal-sign>};	
Description:	For each enumerator/compuScale specified in [SWS_LBAP_00027], if • lowerLimit == upperLimit and • lowerLimit. intervalType == **CLOSED** or undefined there shall exist a C++ enumerator declaration.	
Descriptors:	{<symbol-enum-literal>}	If, for the StdCppImplementationDataType. swDataDefProps. compuMethod. compuPhysToInternal compuContent. compuScale, the: • lowerLimit == upperLimit and • lowerLimit. intervalType == upperLimit. intervalType == **CLOSED** or undefined then the generator shall examine the StdCppImplementationDataType. swDataDefProps. compuMethod. compuPhysToInternal in the following sequence and select the first case which provides a valid C++ identifer: 1. compuContent. compuScale. symbol 2. compuDefaultValue. compuConstContentType. vt 3. compuContent. compuScale. shortLabel If **none** of the above are satisfied, the generator shall terminate with an **error**.
	{<enum-initializer>}	The point range as given by StdCppImplementationDataType. swDataDefProps. compuMethod. compuPhysToInternal compuContent. compuScale lowerLimit/ upperLimit. If neither is present, there shall be no {<enum-initializer>} value for the enumerator.

▽

△

	{<enum-literal-sign>}	If the: • StdCppImplementationDataType. category ==TYPE_REFERENCE in [SWS_LBAP_00027] transitively type-resolves to a StdCppImplementationDataType. category ==VALUE and the • StdCppImplementationDataType. shortName of that, is either: – uint8_t – uint16_t – uint32_t – uint64_t the {<enum-literal-sign>} shall be "U". • Otherwise the {<enum-literal-sign>} shall not be present.

Example:		`// Enumeration Data Type`
		`enum class T_E : std::uint8_t {`
		`  kA,  // without point range <enum-initializer>`
		`  kB = 1U,  // with point range <enum-initializer>`
		`  kC = 2U,`
		`  kD = 4  // without <enum-literal-sign>`
		`};`

See also:	[TPS_MANI_03187], [TPS_SWCT_01276], [TPS_SWCT_01548], [TPS_SWCT_01278], [TPS_SWCT_01569], [TPS_SWCT_01431]	

⌋

8.2.7 Associative Map Data Type

8.2.7.1 Type Alias: {<symbol-assocmap>}

[SWS_LBAP_00023] StdCppImplementationDataType. category ==ASSOCIATIVE_MAP without an Allocator

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-assocmap>}	
Syntax:	using {<symbol-assocmap>} = ara::core::Map<{<assocmap-key-type>}, {<assocmap-value-type>}>;	
Description:	For each StdCppImplementationDataType. category ==ASSOCIATIVE_MAP without an Allocator there shall exist a C++ type alias. The storage is managed by the default allocator std::allocator [10].	
Descriptors:	{<symbol-assocmap>}	as per {<symbol-string>} in [SWS_LBAP_00015]

▽

△

	{<assocmap-key-type>}	as per {<containerized-type>} in [SWS_LBAP_00008]. Refer to [10] for requirements on {<assocmap-key-type>}
	{<assocmap-value-type>}	as per {<containerized-type>} in [SWS_LBAP_00008]

Example:	`// Example: map<typeref,string>`
	`using T_M_IPX_TR = ara::core::String;  // modelled TYPE_REF`
	`using T_M_IPX_T1 = ara::core::String;  // generated`
	`using T_M_IPX =`
	`  ara::core::Map<`
	`  T_M_IPX_TR,  // inplace==undef`
	`  T_M_IPX_T1  // inplace==FALSE`
	`  >;`	

See also:	[TPS_MANI_03183], [TPS_MANI_03184], [TPS_MANI_03185], [SWS_CORE_01400]	

⌋

8.2.7.2 Type Alias: {<symbol-assocmap-alloc>}

[SWS_LBAP_00024] StdCppImplementationDataType. category ==ASSOCIATIVE_MAP with an Allocator

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-assocmap-alloc>}	
Syntax:	using {<symbol-assocmap-alloc>} = ara::core::Map<{<assocmap-key-type>}, {<assocmap-value-type>}, std::less<{<assocmap-key-type>}>, {<fq-allocator-keyval>}<{<assocmap-key-type>}, {<assocmap-value-type>}> >;	
Description:	For each StdCppImplementationDataType. category ==ASSOCIATIVE_MAP with a Allocator there shall exist a C++ type alias.	
Descriptors:	{<symbol-assocmap-alloc>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<assocmap-key-type>}	as per [SWS_LBAP_00023]. Refer to [10] for requirements on {<assocmap-key-type>}
	{<assocmap-value-type>}	as per [SWS_LBAP_00023]
	{<fq-allocator>}	as per {<fq-allocator>} in [SWS_LBAP_00016]

▽

△

Example:	`// Example: map<typeref,string> allocator=TRUE`
	`using T_MA_IPX_TR = ara::core::String;  // modelled TYPE_REF`
	`using T_MA_IPX_T1 = ara::core::String;  // generated`
	`using T_MA_IPX =`
	`  ara::core::Map<`
	`  T_MA_IPX_TR,  // inplace==undef`
	`  T_MA_IPX_T1,  // inplace==FALSE`
	`  std::less<T_MA_IPX_TR>,`
	`  ns1::OuterAllocator< T_MA_IPX_TR, 100 >`
	`  >;`

See also:	[TPS_MANI_03183], [TPS_MANI_03184], [TPS_MANI_03185], [SWS_CORE_01400]

⌋

8.2.8 Variant Data Type

8.2.8.1 Type Alias: {<symbol-variant>}

[SWS_LBAP_00013] `StdCppImplementationDataType.category ==VARIANT`

Upstream requirements: RS_AP_00114, RS_AP_00127

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-variant>}	
Syntax:	using {<symbol-variant>} = ara::core::Variant<{<alt-type-list>}>;	
Description:	For each StdCppImplementationDataType.category ==VARIANT, there shall exist a C++ type alias.	
Descriptors:	{<symbol-variant>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<alt-type-list>}	An **ordered** list of *"alternative types"*. Each *"alternative type"* shall follow the rules of {<containerized-type>} as per [SWS_LBAP_00008]. While an {<alt-type-list>} containing only a single {<containerized-type>} is an edge case, it is permitted by [10].

▽

△

Example:	Example: 3-alternate variant
	`using T_V3_IPX_TR = ara::core::String;  // modelled TYPE_REF`
	`using T_V3_IPX_T1 = ara::core::Array<  // generated`
	`  std::uint8_t, 3`
	`>;`
	`using T_V3_IPX =`
	`  ara::core::Variant<`
	`    T_V3_IPX_T1,  // inplace==FALSE`
	`    ara::core::Variant<  // inplace==TRUE`
	`      ara::core::String,`
	`      ara::core::Vector<T_V3_IPX_TR>`
	`    >`
	`  >;`

See also:	[TPS_MANI_03189], [TPS_MANI_03190], [TPS_MANI_03191], [constr_3429], [SWS_CORE_01601]

⌋

8.2.9 Type Reference

8.2.9.1 Type Alias: {<symbol-typeref>}

[SWS_LBAP_00026] `StdCppImplementationDataType.category ==TYPE_REFERENCE`

Upstream requirements: RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "`{<ns-derived-dir-path>}`/impl_type_`{<shortname-lower>}`.h"	
Scope:	`namespace {<cppidt-ns-list-lc>}`	
Symbol:	`{<symbol-typeref>}`	
Syntax:	`using {<symbol-typeref>} = {<other-symbol>};`	
Description:	For each `StdCppImplementationDataType.category ==TYPE_REFERENCE` there shall exist a C++ type alias.	
Descriptors:	`{<symbol-typeref>}`	as per `{<symbol-string>}` in [SWS_LBAP_00015]
	`{<other-symbol>}`	a reference to any other `CppImplementationDataType` given by `CppImplementationDataType.typeReference`.
Example:	`// Example: type alias` `using T_V3_IPX_TR = ara::core::String;`	
See also:	[TPS_MANI_03193], [constr_10417]	

⌋

8.2.10 CustomCppImplementationDataType

8.2.10.1 Type Alias: {<symbol-custom>}

[SWS_LBAP_00049] `CustomCppImplementationDataType`

Upstream requirements: RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "{<ns-derived-dir-path>}/impl_type_{<shortname-lower>}.h"	
Scope:	namespace {<cppidt-ns-list-lc>}	
Symbol:	{<symbol-custom>}	
Syntax:	using {<symbol-custom>} = {<fq-other-symbol-custom>};	
Description:	For each CustomCppImplementationDataType there shall exist a C++ type alias.	
Descriptors:	{<symbol-custom>}	as per {<symbol-string>} in [SWS_LBAP_00015]
	{<fq-other-symbol-custom>}	Fully namespace-qualified signature of the CustomCppImplementationDataType where the C++ namespace is given by CustomCppImplementationDataType.namespace and the symbol which provides the implementation is given by CustomCppImplementationDataType.shortName

Example:	`// Example: in cust_types.h`
	`namespace cust {`
	`  template <typename T, std::size_t Min, std::size_t Max,`
	`  std::size_t WarnAt>`
	`  class CustVector{};`
	
	`  template <typename T>`
	`  using FixedSizeCustVector = CustVector<T, 10, 50, 42>;`
	`}`
	
	`// generated`
	`using FSCV = cust::FixedSizeCustVector<ara::core::String>;`	

See also:	[TPS_MANI_01309], [TPS_MANI_01212], [constr_1578]	

⌋

8.2.11 Custom Allocator

8.2.11.1 Type Alias: {<symbol-alloc>}

[SWS_LBAP_00047] Custom `Allocator`

Upstream requirements: RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "`{<ns-derived-dir-path>}`/impl_type_`{<shortname-lower>}`.h"	
Scope:	`namespace {<cppidt-ns-list-lc>}`	
Symbol:	`{<symbol-alloc>}`	
Syntax:	`using {<symbol-alloc>} = {<fq-allocator>}<{<alloc-type>}>;`	
Description:	For a `CppImplementationDataType` which aggregates a `templateArgument`. `allocator` there shall exist a C++ type alias.	
Descriptors:	`{<symbol-alloc>}`	The symbol name of the allocator as given by `Allocator`. `shortName`
	`{<fq-allocator>}`	as per `{<fq-allocator>}` in [SWS_LBAP_00016]
	`{<alloc-type>}`	as per `{<containerized-type>}` [SWS_LBAP_00008]
Example:		

```
// Example:
namespace ns4 {
  template <typename T, typename... Args> struct Allocator1
  { };

  template <typename T, std::size_t N> struct Allocator2
  { };
};

using T_OuterAlloc = ns4::Allocator1<ara::core::String>;
using T_InnerAlloc = ns4::Allocator1<
  ara::core::String,
  std::integral_constant<std::uint8_t, 50>
>;
using T_AnotherAlloc = ns4::Allocator2<
  ara::core::String,
  50
>;
```

⌋

A Mentioned Manifest Elements

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.

This chapter is generated.

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	*CppImplementationDataType*, ImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: AbstractImplementationDataType

Class	**Allocator**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the ability to specify an optional custom C++ allocator for a C++ type which may dynamically grow beyond it's initial allocated size during it's lifetime. Any storage principles are defined in the implementation of the allocator itself, which should implement the ISO C++ std::allocator_ traits interface. **Tags:** atp.recommendedPackage=Allocators			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition of a namespace of an Allocator.

Table A.2: Allocator

Class	**ApApplicationError**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform **Tags:** atp.recommendedPackage=ApplicationErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	ApApplicationError			
errorCode	Integer	0..1	attr	This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error.
errorDomain	ApApplicationError Domain	0..1	ref	This reference represents the error domain of the Ap ApplicationError.

Table A.3: ApApplicationError

Class	ApApplicationErrorDomain			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a global error domain for an ApApplicationError. **Tags:** atp.recommendedPackage=ApplicationErrorDomains			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
namespace (ordered)	SymbolProps	*	aggr	This aggregation defines the namespace of the Ap ApplicationErrorDomain
value	PositiveUnlimitedInteger	0..1	attr	This attribute identifies the error category.

Table A.4: ApApplicationErrorDomain

Class	ApApplicationErrorSet			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class acts as a reference target that represents an entire collection of APApplicationErrors. This takes the burden from ClientServerOperations that reference a larger number of ApApplication Errors. **Tags:** atp.recommendedPackage=ApplicationErrorSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
apApplication Error	ApApplicationError	*	ref	This reference represents the collection of ApApplication Error represented by the enclosing ApApplicationErrorSet

Table A.5: ApApplicationErrorSet

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

▽

△

Class	**ApplicationDataType** (abstract)			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.6: ApplicationDataType

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, Field, ParameterDataPrototype, PersistencyDataElement, VariableDataPrototype			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	AutosarDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table A.7: AutosarDataPrototype

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*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractImplementationDataType*, *ApplicationDataType*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps

Table A.8: AutosarDataType

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**
baseTypeEncoding	BaseTypeEncodingString	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 A.9: BaseTypeDirectDefinition

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
fireAndForget	Boolean	0..1	attr	This attribute defines whether this method is a fire&forget method (true) or not (false).
possibleApError	ApApplicationError	*	ref	This reference identifies AdaptivePlatformApplicationErrors as a possible error raised by the enclosing ClientServerOperation.

▽

△

Class	**ClientServerOperation**			
possibleApError Set	ApApplicationErrorSet	*	ref	This reference represents the ability to refer to an entire group of ApApplicationErrors as one model element instead of having to refer to all the represented Ap ApplicationErrors separately.

Table A.10: ClientServerOperation

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. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=compuContent 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 A.11: Compu

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.compu Const			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
compuConst ContentType	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 A.12: CompuConst

Class	***CompuConstContent*** (abstract)			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the constant value of the computation method can be numerical or textual.			
Base	*ARObject*			
Subclasses	CompuConstFormulaContent, CompuConstNumericContent, CompuConstTextContent			
Aggregated by	CompuConst.compuConstContentType			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.13: CompuConstContent

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 A.14: CompuConstTextContent

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 A.15: CompuContent

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*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**CompuMethod**			
compuInternalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=compuInternalToPhys xml.sequenceOffset=80
compuPhysToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=compuPhysToInternal 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 A.16: CompuMethod

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***
a2lDisplayText	String	0..1	attr	The value of this attribute shall be taken for generating one display text (specifically the OutVal) within the equivalent of the enclosing `CompuMethod` in A2L.
compuInverseValue	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
compuScaleContents	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
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

▽

△

Class	**CompuScale**			
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 A.17: CompuScale

Class	***CppImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration.
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type.
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments

▽

△

Class	***CppImplementationDataType*** (abstract)			
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding.
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef).

Table A.18: CppImplementationDataType

Class	**CppImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. A CppImplementationDataTypeElement is used to represent an element of a structure, defining its type.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *CppImplementationDataType*.subElement			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing CppImplementationDataTypeElement as optional. This means the that, at runtime, the Cpp ImplementationDataTypeElement 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.
swDataDef Props	SwDataDefProps	0..1	aggr	This aggregation allows for the definition of qualifying properties of the enclosing CppImplementationDataType Element. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps
typeReference	CppImplementation DataTypeElement Qualifier	0..1	aggr	This aggregation defines the type of the Cpp ImplementationDataTypeElement and determines whether in C++ the CppImplementationDataTypeElement is defined inside or outside of the enclosing Cpp ImplementationDataType.

Table A.19: CppImplementationDataTypeElement

Class	**CppImplementationDataTypeElementQualifier**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This element qualifies the typeReference of the CppImplementationDataTypeElement to the Cpp ImplementationDataType.			
Base	*ARObject*			
Aggregated by	CppImplementationDataTypeElement.typeReference			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
inplace	Boolean	0..1	attr	This attribute defines whether the member type of the CppImplementationDataTypeElement in C++ is an embedded type element inside of the enclosing struct (true) or whether the type declaration is defined outside of the struct.
typeReference	CppImplementation DataType	0..1	ref	This reference defines a type reference.

Table A.20: CppImplementationDataTypeElementQualifier

Class	**CppTemplateArgument**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class has the ability to define properties for template arguments.			
Base	*ARObject*			
Aggregated by	*CppImplementationDataType*.templateArgument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allocator	Allocator	0..1	ref	This reference identifies the applicable allocator.
category	CategoryString	0..1	attr	This attribute shall be used to contribute further clarification regarding the semantics of the enclosing Cpp TemplateArgument.
inplace	Boolean	0..1	attr	This attribute specifies whether the shortName of the referenced templateType is used in the code generation and the type declaration is defined outside of the enclosing CppImplementationDataType (true) or whether the type definition is embedded inside of the enclosing CppImplementationDataType and the shortName is ignored (false).
templateType	CppImplementation DataType	0..1	ref	This reference identifies the data type of the specific template argument required for the language binding.

Table A.21: CppTemplateArgument

Class	**CustomCppImplementationDataType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a custom implementation that is declared in the configured header file. The Short Name of this CustomCppImplementationDataType defines the Class-Name of the custom implementation. **Tags:** atp.recommendedPackage=CppImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationData TypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.22: CustomCppImplementationDataType

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 A.23: DataTypeMap

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*, *AbstractFunctionalClusterDesign*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstance Filter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternal Behavior, ApApplicationEndpoint, *ApmcAbstractDefinition*, *ApmcConfigurationElementDef*, *Apmc ContainerElementValue*, ApmcContainerValue, ApmcEnumerationLiteralDef, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildAction Entity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientId Definition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *Comm ConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, Data Transformation, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic FunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovd MethodPrimitive, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIp LogicalAddress, DoIpNetworkConfigurationDesign, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, Ethernet WakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget MethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, Global Supervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBus Part*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, Method Mapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *Phm Supervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwComposition Prototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecution Context, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *Service MethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslation Props, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, *StateManagementActionItem*, State ManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, Static SocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionMode Condition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwitchAsynchronous TrafficShaperGroupEntry, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSync Accuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, Timing ModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Identifiable*** (abstract)			
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
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 A.24: Identifiable

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 A.25: ImplementationDataType

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 A.26: Limit

Class	**PersistencyKeyValueStorageInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::Persistency			
Note	This meta-class provides the ability to implement a `PortInterface` for supporting persistency use cases for data. **Tags:** atp.recommendedPackage=PersistencyKeyValueStorageInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PersistencyInterface*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataElement	PersistencyData Element	*	aggr	This aggregation represents the collection of Persistency DataElements in the context of the enclosing Persistency KeyValueStorageInterface.
dataTypeFor Serialization	AbstractImplementation DataType	*	ref	This reference identifies the AbstractImplementationData Types that shall be supported for storing in a key-value storage in addition to the types already determined from tha aggregation of PersistencyDataElement.
dataType Mapping	PersistencyKeyValue DataTypeMapping	0..1	aggr	This aggregation provides a collection of replacement rules for data types used in the context of the enclosing PersistencyKeyValueStorageInterface.

Table A.27: PersistencyKeyValueStorageInterface

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	*AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallStateSwitchInterface, *IdsmAbstractPort Interface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *Persistency Interface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName

Table A.28: PortInterface

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, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescription Entity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, Pnc MappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, Someip RequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent

▽

△

Class	***Referrable*** (abstract)			
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 A.29: Referrable

Class	**ServiceInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=field.shortName, field.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=40
majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** xml.sequenceOffset=10
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=method.shortName, method.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=50
minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** xml.sequenceOffset=20

▽

△

Class	**ServiceInterface**			
trigger	Trigger	*	aggr	This represents the collection of triggers defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=trigger.shortName, trigger.variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=60

Table A.30: ServiceInterface

Class	**StdCppImplementationDataType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a C++ Standard Library feature. **Tags:** atp.recommendedPackage=CppImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.31: StdCppImplementationDataType

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to contribute a part of a namespace.			
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 A.32: SymbolProps

B Change History

This chapter provides an overview of the 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.

B.1 Change History of this document according to AUTOSAR Release R21-11

B.1.1 Added Specification Items in R21-11

Number	Heading
[SWS_LBAP_00001]	ARA generator rejection of unmapped data types
[SWS_LBAP_00002]	`ARA Language Binding Generator` usage of `typeEmitter`
[SWS_LBAP_00003]	ARA generator rejection of symbol clashes
[SWS_LBAP_00004]	Naming of data types by `shortName`
[SWS_LBAP_00005]	Standardized `Primitive CppImplementationDataType`s
[SWS_LBAP_00006]	`Primitive CppImplementationDataType` fixed width integers
[SWS_LBAP_00007]	`StdCppImplementationDataType` of `category`=`ARRAY` with one dimension
[SWS_LBAP_00008]	`StdCppImplementationDataType` of `category`=`ARRAY` with multiple dimensions
[SWS_LBAP_00009]	`CustomCppImplementationDataType` of `category`=`ARRAY`
[SWS_LBAP_00010]	`StdCppImplementationDataType` of `category`=`STRUCTURE`
[SWS_LBAP_00011]	Structure element specification typed by `CppImplementationDataType`
[SWS_LBAP_00012]	Accessing optional record elements inside a `Structure CppImplementationDataType` that are serialized with the Tag-Length-Value principle.
[SWS_LBAP_00013]	`StdCppImplementationDataType` of `category`=`VARIANT`
[SWS_LBAP_00014]	`CustomCppImplementationDataType` of `category`=`VARIANT`
[SWS_LBAP_00015]	`StdCppImplementationDataType` of `category`=`STRING` without `Allocator`
[SWS_LBAP_00016]	`StdCppImplementationDataType` of `category`=`STRING` with `Allocator`
[SWS_LBAP_00017]	`StdCppImplementationDataType` of `category`=`VECTOR` with one dimension, without `Allocator`
[SWS_LBAP_00018]	`StdCppImplementationDataType` of `category`=`VECTOR` with one dimension, with `Allocator`
[SWS_LBAP_00019]	`StdCppImplementationDataType` of `category`=`VECTOR` with multiple dimensions
[SWS_LBAP_00020]	`CppImplementationDataType` with `category`=`VECTOR` size semantics

△

Number	Heading
[SWS_LBAP_00021]	Imposing memory limits with `Allocator`
[SWS_LBAP_00022]	`CustomCppImplementationDataType` of `category`=VECTOR
[SWS_LBAP_00023]	`StdCppImplementationDataType` with `category`=ASSOCIATIVE_MAP without an `Allocator`
[SWS_LBAP_00024]	`StdCppImplementationDataType` with `category`=ASSOCIATIVE_MAP with an `Allocator`
[SWS_LBAP_00025]	`CustomCppImplementationDataType` of `category`=ASSOCIATIVE_MAP without `Allocator`
[SWS_LBAP_00026]	`StdCppImplementationDataType` of `category`=TYPE_REFERENCE
[SWS_LBAP_00027]	Enumeration Data Type
[SWS_LBAP_00028]	`Enumeration Data Type` - enumerators
[SWS_LBAP_00029]	`Enumeration Data Type` - skip `CompuScale`s with non-point range
[SWS_LBAP_00030]	ARA generator rejection of incomplete `Enumeration Data Type`s
[SWS_LBAP_00031]	`Scale Linear And Texttable Data Type`
[SWS_LBAP_00032]	`CppImplementationTypes Header File`s artifact generation
[SWS_LBAP_00033]	`CppImplementationTypes Header File`s file names
[SWS_LBAP_00034]	`CppImplementationTypes Header File`s directory names
[SWS_LBAP_00035]	`CppImplementationTypes Header File`s namespace hierarchy
[SWS_LBAP_00036]	`CppImplementationTypes Header File`s multiple inclusion guard
[SWS_LBAP_00037]	Principle of an `ARA Language Binding Generator`
[SWS_LBAP_00038]	`CustomCppImplementationDataType` of `category`=ASSOCIATIVE_MAP with `Allocator`

Table B.1: Added Specification Items in R21-11

B.1.2 Changed Specification Items in R21-11

none

B.1.3 Deleted Specification Items in R21-11

none

B.2 Change History of this document according to AUTOSAR Release R22-11

B.2.1 Added Specification Items in R22-11

Number	Heading
[SWS_LBAP_00039]	Encoding of strings with a baseTypeEncoding
[SWS_LBAP_00040]	Encoding of strings without a baseTypeEncoding
[SWS_LBAP_00047]	cppidt_ns_list_lc::symbol_alloc::symbol_allocCustom Allocator
[SWS_LBAP_00048]	cppidt_ns_list_lc::symbol_vector_alloc_maxsize:: symbol_vector_alloc_maxsizeStdCppImplementationDataType. category ==VECTOR with an Allocator and arraySize
[SWS_LBAP_00049]	cppidt_ns_list_lc::symbol_custom::symbol_custom CustomCppImplementationDataType

Table B.2: Added Specification Items in R22-11

B.2.2 Changed Specification Items in R22-11

Number	Heading
[SWS_LBAP_00005]	Standardized Primitive CppImplementationDataTypes
[SWS_LBAP_00008]	cppidt_ns_list_lc::symbol_array::symbol_array StdCppImplementationDataType. category ==ARRAY
[SWS_LBAP_00010]	cppidt_ns_list_lc::symbol_struct::symbol_struct StdCppImplementationDataType. category ==STRUCTURE
[SWS_LBAP_00011]	cppidt_ns_list_lc::symbol_struct_element:: symbol_struct_elementCppImplementationDataTypeElement. isOptional ==FALSE or undefined
[SWS_LBAP_00012]	cppidt_ns_list_lc::symbol_struct_opt_element:: symbol_struct_opt_element CppImplementationDataTypeElement. isOptional ==TRUE
[SWS_LBAP_00013]	cppidt_ns_list_lc::symbol_variant::symbol_variant StdCppImplementationDataType. category ==VARIANT
[SWS_LBAP_00015]	cppidt_ns_list_lc::symbol_string::symbol_string StdCppImplementationDataType. category ==STRING without an Allocator
[SWS_LBAP_00016]	cppidt_ns_list_lc::symbol_string_alloc:: symbol_string_allocStdCppImplementationDataType. category ==STRING with an Allocator
[SWS_LBAP_00017]	cppidt_ns_list_lc::symbol_vector::symbol_vector StdCppImplementationDataType. category ==VECTOR without an Allocator

▽

△

Number	Heading
[SWS_LBAP_00018]	cppidt_ns_list_lc::symbol_vector_alloc::symbol_vector_allocStdCppImplementationDataType. category ==VECTOR with an Allocator
[SWS_LBAP_00023]	cppidt_ns_list_lc::symbol_assocmap::symbol_assocmap StdCppImplementationDataType. category ==ASSOCIATIVE_MAP without an Allocator
[SWS_LBAP_00024]	cppidt_ns_list_lc::symbol_assocmap_alloc::symbol_assocmap_allocStdCppImplementationDataType. category ==ASSOCIATIVE_MAP with an Allocator
[SWS_LBAP_00026]	cppidt_ns_list_lc::symbol_typeref::symbol_typeref StdCppImplementationDataType. category ==TYPE_REFERENCE
[SWS_LBAP_00027]	cppidt_ns_list_lc::symbol_enum::symbol_enumEnumeration Data Type
[SWS_LBAP_00028]	cppidt_ns_list_lc::symbol_enum_literal::symbol_enum_literalEnumeration Data Type - enumerators
[SWS_LBAP_00033]	ns_derived_dir_path_impl_type_shortname_lower.h::ns_derived_dir_pathCppImplementationDataTypes Header Files: file name and multiple inclusion guard
[SWS_LBAP_00035]	cppidt_ns_list_lc::cppidt_ns_list_lc CppImplementationDataTypes Header Files namespace hierarchy cppidt-ns-list-lc

Table B.3: Changed Specification Items in R22-11

B.2.3 Deleted Specification Items in R22-11

Number	Heading
[SWS_LBAP_00001]	ARA generator rejection of unmapped data types
[SWS_LBAP_00004]	Naming of data types by shortName
[SWS_LBAP_00007]	StdCppImplementationDataType of category=ARRAY with one dimension
[SWS_LBAP_00009]	CustomCppImplementationDataType of category=ARRAY
[SWS_LBAP_00014]	CustomCppImplementationDataType of category=VARIANT
[SWS_LBAP_00019]	StdCppImplementationDataType of category=VECTOR with multiple dimensions
[SWS_LBAP_00020]	CppImplementationDataType with category=VECTOR size semantics
[SWS_LBAP_00021]	Imposing memory limits with Allocator
[SWS_LBAP_00022]	CustomCppImplementationDataType of category=VECTOR
[SWS_LBAP_00025]	CustomCppImplementationDataType of category=ASSOCIATIVE_MAP without Allocator
[SWS_LBAP_00029]	Enumeration Data Type - skip CompuScales with non-point range
[SWS_LBAP_00030]	ARA generator rejection of incomplete Enumeration Data Types

▽

△

Number	Heading
[SWS_LBAP_00032]	CppImplementationTypes Header Files artifact generation
[SWS_LBAP_00034]	CppImplementationTypes Header Files directory names
[SWS_LBAP_00036]	CppImplementationTypes Header Files multiple inclusion guard
[SWS_LBAP_00038]	CustomCppImplementationDataType of category=ASSOCIATIVE_MAP with Allocator

Table B.4: Deleted Specification Items in R22-11

B.3 Change History of this document according to AUTOSAR Release R23-11

B.3.1 Added Specification Items in R23-11

Number	Heading
[SWS_LBAP_00048]	StdCppImplementationDataType. category ==VECTOR with an Allocator and arraySize
[SWS_LBAP_00049]	CustomCppImplementationDataType

Table B.5: Added Specification Items in R23-11

B.3.2 Changed Specification Items in R23-11

none

B.3.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_LBAP_00001]	ARA generator rejection of unmapped data types
[SWS_LBAP_00004]	Naming of data types by shortName
[SWS_LBAP_00007]	StdCppImplementationDataType of category=ARRAY with one dimension
[SWS_LBAP_00009]	CustomCppImplementationDataType of category=ARRAY
[SWS_LBAP_00014]	CustomCppImplementationDataType of category=VARIANT
[SWS_LBAP_00019]	StdCppImplementationDataType of category=VECTOR with multiple dimensions
[SWS_LBAP_00020]	CppImplementationDataType with category=VECTOR size semantics

▽

△

Number	Heading
[SWS_LBAP_00021]	Imposing memory limits with `Allocator`
[SWS_LBAP_00022]	`CustomCppImplementationDataType` of `category`=`VECTOR`
[SWS_LBAP_00025]	`CustomCppImplementationDataType` of `category`=`ASSOCIATIVE_MAP` without `Allocator`
[SWS_LBAP_00029]	`Enumeration Data Type` - skip `CompuScale`s with non-point range
[SWS_LBAP_00030]	ARA generator rejection of incomplete `Enumeration Data Type`s
[SWS_LBAP_00031]	`Scale Linear And Texttable Data Type`
[SWS_LBAP_00032]	`CppImplementationTypes Header File`s artifact generation
[SWS_LBAP_00034]	`CppImplementationTypes Header File`s directory names
[SWS_LBAP_00036]	`CppImplementationTypes Header File`s multiple inclusion guard
[SWS_LBAP_00038]	`CustomCppImplementationDataType` of `category`=`ASSOCIATIVE_MAP` with `Allocator`
[SWS_LBAP_00041]	Usage of an Allocator
[SWS_LBAP_00042]	Usage of a Default Allocator
[SWS_LBAP_00043]	Usage of a Custom Allocator
[SWS_LBAP_00044]	Header file location of a Custom Allocator
[SWS_LBAP_00045]	Namespace of a Custom Allocator
[SWS_LBAP_00046]	Include declaration for a Custom Allocator

Table B.6: Deleted Specification Items in R23-11

B.3.4 Added Constraints in R23-11

none

B.3.5 Changed Constraints in R23-11

none

B.3.6 Deleted Constraints in R23-11

Number	Heading
[SWS_LBAP_-CONSTR_-00001]	Invalid header file location of a Custom Allocator

▽

△

Number	Heading
[SWS_LBAP_-CONSTR_-00002]	Unspecified namespace of a Custom Allocator

Table B.7: Deleted Constraints in R23-11

B.4 Change History of this document according to AUTOSAR Release R24-11

B.4.1 Added Specification Items in R24-11

Number	Heading
[SWS_LBAP_00050]	Definition of Header File {<aaed-dir>}/{<aaed-file-prefix>}_error_domain.h
[SWS_LBAP_00051]	Definition of Namespace {<aaed-ns>}
[SWS_LBAP_00052]	Definition of API enum {<aaed-ns>}::{<aaed-sn>}Errc
[SWS_LBAP_00053]	Definition of API variable {<aaed-ns>}::{<symbol-aae-sn>}
[SWS_LBAP_00054]	Definition of API function {<aaed-ns>}::Get{<aaed-sn>}ErrorDomain
[SWS_LBAP_00055]	Definition of API function {<aaed-ns>}::MakeErrorCode
[SWS_LBAP_00056]	Definition of API class {<aaed-ns>}::{<aaed-sn>}Exception
[SWS_LBAP_00057]	Definition of API function {<aaed-ns>}::{<aaed-sn>}Exception::{<aaed-sn>}Exception
[SWS_LBAP_00058]	Definition of API class {<aaed-ns>}::{<aaed-sn>}ErrorDomain
[SWS_LBAP_00059]	Definition of API type {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Errc
[SWS_LBAP_00060]	Definition of API type {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Exception
[SWS_LBAP_00061]	Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::{<aaed-sn>}ErrorDomain
[SWS_LBAP_00062]	Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Name
[SWS_LBAP_00063]	Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::Message
[SWS_LBAP_00064]	Definition of API function {<aaed-ns>}::{<aaed-sn>}ErrorDomain::ThrowAsException

Table B.8: Added Specification Items in R24-11

B.4.2 Changed Specification Items in R24-11

Number	Heading
[SWS_LBAP_00002]	ARA Language Binding Generator usage of typeEmitter
[SWS_LBAP_00003]	ARA generator rejection of symbol clashes
[SWS_LBAP_00005]	Standardized Primitive CppImplementationDataTypes
[SWS_LBAP_00006]	Primitive CppImplementationDataType fixed width integers
[SWS_LBAP_00008]	StdCppImplementationDataType. category ==ARRAY
[SWS_LBAP_00010]	StdCppImplementationDataType. category ==STRUCTURE
[SWS_LBAP_00011]	CppImplementationDataTypeElement. isOptional ==FALSE or undefined
[SWS_LBAP_00012]	CppImplementationDataTypeElement. isOptional ==TRUE
[SWS_LBAP_00013]	StdCppImplementationDataType. category ==VARIANT
[SWS_LBAP_00015]	StdCppImplementationDataType. category ==STRING without an Allocator
[SWS_LBAP_00016]	StdCppImplementationDataType. category ==STRING with an Allocator
[SWS_LBAP_00017]	StdCppImplementationDataType. category ==VECTOR without an Allocator
[SWS_LBAP_00018]	StdCppImplementationDataType. category ==VECTOR with an Allocator
[SWS_LBAP_00023]	StdCppImplementationDataType. category ==ASSOCIATIVE_MAP without an Allocator
[SWS_LBAP_00024]	StdCppImplementationDataType. category ==ASSOCIATIVE_MAP with an Allocator
[SWS_LBAP_00026]	StdCppImplementationDataType. category ==TYPE_REFERENCE
[SWS_LBAP_00027]	Enumeration Data Type
[SWS_LBAP_00028]	Enumeration Data Type - enumerators
[SWS_LBAP_00033]	CppImplementationDataTypes Header Files: file name and multiple inclusion guard
[SWS_LBAP_00035]	CppImplementationDataTypes Header Files namespace hierarchy
[SWS_LBAP_00037]	Principle of an ARA Language Binding Generator
[SWS_LBAP_00039]	Encoding of strings with a baseTypeEncoding
[SWS_LBAP_00040]	Encoding of strings without a baseTypeEncoding
[SWS_LBAP_00047]	Custom Allocator
[SWS_LBAP_00048]	StdCppImplementationDataType. category ==VECTOR with an Allocator and arraySize
[SWS_LBAP_00049]	CustomCppImplementationDataType

Table B.9: Changed Specification Items in R24-11

B.4.3 Deleted Specification Items in R24-11

none

B.4.4 Added Constraints in R24-11

none

B.4.5 Changed Constraints in R24-11

none

B.4.6 Deleted Constraints in R24-11

none