Document Title	Specification of Platform Types for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	875

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-03

Document Change History

Date	Release	Changed by	Description
2018-03-29	18-03	AUTOSAR Release Management	• Editorial changes
2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

References

[1] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[2] Collection of blueprints for AUTOSAR M1 models
AUTOSAR_MOD_GeneralBlueprints

[3] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[4] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

[5] IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008)

1 Introduction

This document defines primitive ImplementationDataTypes that can be used in ServiceInterface descriptions provided in ARXML as defined in TPS_ManifestSpecification [1].

The definition of common used ImplementationDataTypes increases the portability of applications and prevents from re-defining the same types for each application.

Please note that AUTOSAR provides a Blueprint file AUTOSAR_MOD_CommonDataTypes_Blueprint.arxml in [2] that is released in the Classic Platform that can be used as basis for creation of a file that contains such common used ImplementationDataTypes.

1.1 Requirements Tracing

Requirements against this document are exclusively stated in the corresponding requirements document [3].

The following table 1.1 references the requirements specified in the corresponding requirements document and provides information about individual specification items that fulfill a given requirement.

Requirement	Description	Satisfied by
[RS_AP_00111]	The AUTOSAR Adaptive platform shall support source code portability for AUTOSAR Adaptive applications.	[SWS_APT_00001] [SWS_APT_00002] [SWS_APT_00003] [SWS_APT_00004] [SWS_APT_00005] [SWS_APT_00006] [SWS_APT_00007] [SWS_APT_00008] [SWS_APT_00009] [SWS_APT_00010] [SWS_APT_00011] [SWS_APT_00012] [SWS_APT_00013] [SWS_APT_00014] [SWS_APT_00015] [SWS_APT_00016] [SWS_APT_00017] [SWS_APT_00018] [SWS_APT_00019] [SWS_APT_00020] [SWS_APT_00021] [SWS_APT_00022] [SWS_APT_00023] [SWS_APT_00024] [SWS_APT_00025] [SWS_APT_00026] [SWS_APT_00027] [SWS_APT_00028] [SWS_APT_00029] [SWS_APT_00030] [SWS_APT_00031] [SWS_APT_00032] [SWS_APT_00033] [SWS_APT_00034] [SWS_APT_00035] [SWS_APT_00036] [SWS_APT_00037] [SWS_APT_00038] [SWS_APT_00039] [SWS_APT_00040] [SWS_APT_00041] [SWS_APT_00042] [SWS_APT_00043] [SWS_APT_00044] [SWS_APT_00045] [SWS_APT_00046] [SWS_APT_00047] [SWS_APT_00048] [SWS_APT_00049] [SWS_APT_00050] [SWS_APT_00051]

Table 1.1: RequirementsTracing

2 Primitive ImplementationDataTypes and their mapping to C++ datatypes

This chapter describes diverse primitive ImplementationDataTypes that are predefined by AUTOSAR for the usage in the Adaptive Platform and defines their mapping to C++ datatypes.

The mapping of a primitive ImplementationDataType that is used in a ServiceInterface to a C++ datatype is defined in SWS_CommunicationManagement [4] in [SWS_CM_00402]. According to this rule the symbol or the shortName of the ImplementationDataType will be used as alias for the nativeDeclaration.

2.1 Boolean

[SWS_APT_00049] primitive Implementation Data Type *boolean* ⌈ The primitive Implementation Data Type *boolean* is classified by the category VALUE and directly refers the *boolean* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.1: Boolean ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>boolean</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            boolean</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>Platform_Types.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00050] SwBaseType *boolean* ⌈ The *boolean* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *bool*. The baseTypeEncoding is set to BOOLEAN. ⌋*(RS_AP_00111)*

Listing 2.2: Boolean SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>boolean</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>BOOLEAN</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>OPAQUE</BYTE-ORDER>
  <NATIVE-DECLARATION>bool</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *boolean* `ImplementationDataType` will be mapped to the bool-type in C++, that is capable of holding one of the two values: true or false.

[SWS_APT_00051] Platform specific settings in `SwBaseType` *boolean* ⌈ The setting of `baseTypeSize`, `memAlignment` and `byteOrder` in `SwBaseType` *boolean* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for `baseTypeSize`, `memAlignment` and `byteOrder` that are shown in the listing above are only exemplary.

Please note that in C++ sizeof(bool) is implementation-defined. With the setting of the `category` to FIXED_LENGTH and with the `baseTypeSize` set to a value the sizeof(bool) would be fixed to 32 bit.

The `baseTypeEncoding` that is set to BOOLEAN defines that an unsigned integer value will be interpreted as true (1) or false (0).

2.2 Signed Integer

2.2.1 sint8

[SWS_APT_00001] primitive Implementation Data Type *sint8* ⌈ The primitive Implementation Data Type *sint8* is classified by the `category` VALUE and directly refers the *sint8* `SwBaseType` in its `SwDataDefProps`. ⌋*(RS_AP_00111)*

Listing 2.3: sint8 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint8</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            sint8</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00002] `SwBaseType` *sint8* ⌈ The *sint8* `SwBaseType` is classified by the `category` FIXED_LENGTH and has the `nativeDeclaration` set to *std::int8_t*. The `baseTypeSize` is set to 8 bit and the `baseTypeEncoding` to 2C (2's complement). ⌋*(RS_AP_00111)*

Listing 2.4: sint8 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint8</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
```

```
    <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
    <MEM-ALIGNMENT>8</MEM-ALIGNMENT>
    <NATIVE-DECLARATION>std::int8_t</NATIVE-DECLARATION>
  </SW-BASE-TYPE>
```

The *sint8* ImplementationDataType will be mapped to signed integer type of the C++ standard library with width of exactly 8 bit. Negative values are represented using 2's complement. No padding bits are defined.

[SWS_APT_00003] Platform specific settings in SwBaseType *sint8* ⌈ The setting of memAlignment in SwBaseType *sint8* is platform-specific. ⌋*(RS_AP_00111)*

It means that the value for memAlignment that is shown in the listing above is only exemplary.

2.2.2 sint16

[SWS_APT_00004] primitive Implementation Data Type *sint16* ⌈ The primitive Implementation Data Type *sint16* is classified by the category VALUE and directly refers the *sint16* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.5: sint16 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint16</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            sint16</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00005] SwBaseType *sint16* ⌈ The *sint16* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::int16_t*. The baseTypeSize is set to 16 bit and the baseTypeEncoding to 2C (2's complement). ⌋*(RS_AP_00111)*

Listing 2.6: sint16 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint16</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>16</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>16</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::int16_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint16* ImplementationDataType will be mapped to signed integer type of the C++ standard library with width of exactly 16 bit. Negative values are represented using 2's complement. No padding bits are defined.

[SWS_APT_00006] Platform specific settings in SwBaseType *sint16* ⌈ The setting of memAlignment and byteOrder in SwBaseType *sint16* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.2.3 sint32

[SWS_APT_00007] primitive Implementation Data Type *sint32* ⌈ The primitive Implementation Data Type *sint32* is classified by the category VALUE and directly refers the *sint32* SwBaseType in its SwDataDefProps. ⌋ *(RS_AP_00111)*

Listing 2.7: sint32 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint32</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            sint32</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00008] SwBaseType *sint32* ⌈ The *sint32* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::int32_t*. The baseTypeSize is set to 32 bit and the baseTypeEncoding to 2C (2's complement). ⌋ *(RS_AP_00111)*

Listing 2.8: sint32 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint32</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::int32_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint32* ImplementationDataType will be mapped to signed integer type of the C++ standard library with width of exactly 32 bit. Negative values are represented using 2's complement. No padding bits are defined.

[SWS_APT_00009] Platform specific settings in SwBaseType *sint32* ⌈ The setting of memAlignment and byteOrder in SwBaseType *sint32* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.2.4 sint64

[SWS_APT_00010] primitive Implementation Data Type *sint64* ⌈ The primitive Implementation Data Type *sint64* is classified by the category VALUE and directly refers the *sint64* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.9: sint64 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint64</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
           sint64</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00011] SwBaseType *sint64* ⌈ The *sint64* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::int64_t*. The baseTypeSize is set to 64 bit and the baseTypeEncoding to 2C (2's complement). ⌋*(RS_AP_00111)*

Listing 2.10: sint64 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint64</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>64</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>64</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::int64_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint64* ImplementationDataType will be mapped to signed integer type of the C++ standard library with width of exactly 64 bit. Negative values are represented using 2's complement. No padding bits are defined.

[SWS_APT_00012] Platform specific settings in SwBaseType *sint64* ⌈ The setting of memAlignment and byteOrder in SwBaseType *sint64* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.2.5 sint8_least

[SWS_APT_00013] primitive Implementation Data Type *sint8_least* ⌈ The primitive Implementation Data Type *sint8_least* is classified by the category VALUE and directly refers the *sint8_least* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.11: sint8_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint8_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            sint8_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00014] SwBaseType *sint8_least* ⌈ The *sint8_least* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::int_least8_t*. The baseTypeEncoding is set to 2C (2's complement). ⌋*(RS_AP_00111)*

Listing 2.12: sint8_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint8_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <NATIVE-DECLARATION>std::int_least8_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint8_least* ImplementationDataType will be mapped to signed integer type of the C++ standard library with a minimum of 8 bits.

[SWS_APT_00015] Platform specific settings in SwBaseType *sint8_least* ⌈ The setting of baseTypeSize and memAlignment in SwBaseType *sint8_least* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for baseTypeSize and memAlignment that are shown in the listing above are only exemplary.

2.2.6 sint16_least

[SWS_APT_00016] primitive Implementation Data Type *sint16_least* ⌈ The primitive Implementation Data Type *sint16_least* is classified by the `category` VALUE and directly refers the *sint16_least* `SwBaseType` in its `SwDataDefProps`. ⌋ *(RS_AP_00111)*

Listing 2.13: sint16_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint16_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
           sint16_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00017] `SwBaseType` *sint16_least* ⌈ The *sint16_least* `SwBaseType` is classified by the `category` FIXED_LENGTH and has the `nativeDeclaration` set to *std::int_least16_t*. The `baseTypeEncoding` is set to 2C (2's complement). ⌋ *(RS_AP_00111)*

Listing 2.14: sint16_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint16_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::int_least16_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint16_least* `ImplementationDataType` will be mapped to signed integer type of the C++ standard library with a minimum of 16 bits.

[SWS_APT_00018] Platform specific settings in `SwBaseType` *sint16_least* ⌈ The setting of `baseTypeSize`, `memAlignment` and `byteOrder` in `SwBaseType` *sint16_least* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for `baseTypeSize`, `memAlignment` and `byteOrder` that are shown in the listing above are only exemplary.

2.2.7 sint32_least

[SWS_APT_00019] primitive Implementation Data Type *sint32_least* ⌈ The primitive Implementation Data Type *sint32_least* is classified by the category VALUE and directly refers the *sint32_least* SwBaseType in its SwDataDefProps. ⌋ *(RS_AP_00111)*

Listing 2.15: sint32_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>sint32_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            sint32_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00020] SwBaseType *sint32_least* ⌈ The *sint32_least* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::int_least32_t*. The baseTypeEncoding is set to 2C (2's complement). ⌋ *(RS_AP_00111)* .

Listing 2.16: sint32_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>sint32_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::int_least32_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *sint32_least* ImplementationDataType will be mapped to signed integer type of the C++ standard library with a minimum of 32 bits.

[SWS_APT_00021] Platform specific settings in SwBaseType *sint32_least* ⌈ The setting of baseTypeSize, memAlignment and byteOrder in SwBaseType *sint32_least* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for baseTypeSize, memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.3 Unsigned Integer

2.3.1 uint8

[SWS_APT_00022] primitive Implementation Data Type *uint8* ⌈ The primitive Implementation Data Type *uint8* is classified by the category VALUE and directly refers the *uint8* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.17: uint8 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint8</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint8</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00023] SwBaseType *uint8* ⌈ The *uint8* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::uint8_t*. The baseTypeSize is set to 8 bit and the baseTypeEncoding to NONE. ⌋ *(RS_AP_00111)*

Listing 2.18: uint8 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint8</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>8</MEM-ALIGNMENT>
  <NATIVE-DECLARATION>std::uint8_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint8* ImplementationDataType will be mapped to unsigned integer type of the C++ standard library with width of exactly 8 bit. No padding bits are defined.

[SWS_APT_00024] Platform specific settings in SwBaseType *uint8* ⌈ The setting of memAlignment in SwBaseType *uint8* is platform-specific. ⌋*(RS_AP_00111)*

It means that the value for memAlignment that is shown in the listing above is only exemplary.

2.3.2 uint16

[SWS_APT_00025] primitive Implementation Data Type *uint16* ⌈ The primitive Implementation Data Type *uint16* is classified by the category VALUE and directly refers the *uint16* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.19: uint16 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint16</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint16</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00026] SwBaseType *uint16* ⌈ The *uint16* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::uint16_t*. The baseTypeSize is set to 16 bit and the baseTypeEncoding to NONE. ⌋*(RS_AP_00111)*

Listing 2.20: uint16 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint16</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>16</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>16</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::uint16_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint16* ImplementationDataType will be mapped to unsigned integer type of the C++ standard library with width of exactly 16 bit. No padding bits are defined.

[SWS_APT_00027] Platform specific settings in SwBaseType *uint16* ⌈ The setting of memAlignment and byteOrder in SwBaseType *uint16* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.3.3 uint32

[SWS_APT_00028] primitive Implementation Data Type *uint32* ⌈ The primitive Implementation Data Type *uint32* is classified by the `category` VALUE and directly refers the *uint32* `SwBaseType` in its `SwDataDefProps`. ⌋*(RS_AP_00111)*

Listing 2.21: uint32 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint32</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint32</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00029] `SwBaseType` *uint32* ⌈ The *uint32* `SwBaseType` is classified by the `category` FIXED_LENGTH and has the `nativeDeclaration` set to *std::uint32_t*. The `baseTypeSize` is set to 32 bit and the `baseTypeEncoding` to NONE. ⌋*(RS_AP_00111)*

Listing 2.22: uint32 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint32</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::uint32_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint32* `ImplementationDataType` will be mapped to unsigned integer type of the C++ standard library with width of exactly 32 bit. No padding bits are defined.

[SWS_APT_00030] Platform specific settings in `SwBaseType` *uint32* ⌈ The setting of `memAlignment` and `byteOrder` in `SwBaseType` *uint32* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for `memAlignment` and `byteOrder` that are shown in the listing above are only exemplary.

2.3.4 uint64

[SWS_APT_00031] primitive Implementation Data Type *uint64* ⌈ The primitive Implementation Data Type *uint64* is classified by the category VALUE and directly refers the *uint64* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.23: uint64 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint64</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint64</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00032] SwBaseType *uint64* ⌈ The *uint64* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::uint64_t*. The baseTypeSize is set to 64 bit and the baseTypeEncoding to NONE. ⌋*(RS_AP_00111)*

Listing 2.24: uint64 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint64</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>64</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>64</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::uint64_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint64* ImplementationDataType will be mapped to unsigned integer type of the C++ standard library with width of exactly 64 bit. No padding bits are defined.

[SWS_APT_00033] Platform specific settings in SwBaseType *uint64* ⌈ The setting of memAlignment and byteOrder in SwBaseType *uint64* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.3.5 uint8_least

[SWS_APT_00034] primitive Implementation Data Type *uint8_least* ⌈ The primitive Implementation Data Type *uint8_least* is classified by the `category` VALUE and directly refers the *uint8_least* `SwBaseType` in its `SwDataDefProps`. ⌋*(RS_AP_00111)*

Listing 2.25: uint8_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint8_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint8_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00035] `SwBaseType` *uint8_least* ⌈ The *uint8_least* `SwBaseType` is classified by the `category` FIXED_LENGTH and has the `nativeDeclaration` set to *std::uint_least8_t*. The `baseTypeEncoding` is set to NONE. ⌋*(RS_AP_00111)*

Listing 2.26: uint8_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint8_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <NATIVE-DECLARATION>std::uint_least8_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint8_least* `ImplementationDataType` will be mapped to unsigned integer type of the C++ standard library with a minimum of 8 bits.

[SWS_APT_00036] Platform specific settings in `SwBaseType` *uint8_least* ⌈ The setting of `baseTypeSize` and `memAlignment` in `SwBaseType` *uint8_least* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for `baseTypeSize` and `memAlignment` that are shown in the listing above are only exemplary.

2.3.6 uint16_least

[SWS_APT_00037] primitive Implementation Data Type *uint16_least* ⌈ The primitive Implementation Data Type *uint16_least* is classified by the `category` VALUE and directly refers the *uint16_least* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.27: uint16_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint16_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint16_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00038] SwBaseType *uint16_least* ⌈The *uint16_least* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::uint_least16_t*. The baseTypeEncoding is set to NONE. ⌋*(RS_AP_00111)*

Listing 2.28: uint16_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint16_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::uint_least16_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint16_least* ImplementationDataType will be mapped to unsigned integer type of the C++ standard library with a minimum of 16 bits.

[SWS_APT_00039] Platform specific settings in SwBaseType *uint16_least* ⌈The setting of baseTypeSize, memAlignment and byteOrder in SwBaseType *uint16_least* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for baseTypeSize, memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.3.7 uint32_least

[SWS_APT_00040] primitive Implementation Data Type *uint32_least* ⌈The primitive Implementation Data Type *uint32_least* is classified by the category VALUE and directly refers the *uint32_least* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.29: uint32_least ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint32_least</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            uint32_least</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>cstdint.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00041] SwBaseType *uint32_least* ⌈ The *uint32_least* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *std::uint_least32_t*. The baseTypeEncoding is set to NONE. ⌋*(RS_AP_00111)*

Listing 2.30: uint32_least SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint32_least</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>std::uint_least32_t</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *uint32_least* ImplementationDataType will be mapped to unsigned integer type of the C++ standard library with a minimum of 32 bits.

[SWS_APT_00042] Platform specific settings in SwBaseType *uint32_least* ⌈ The setting of baseTypeSize, memAlignment and byteOrder in SwBaseType *uint32_least* is platform-specific. ⌋*(RS_AP_00111)*

It means that the values for baseTypeSize, memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.4 Float

2.4.1 float32

[SWS_APT_00043] primitive Implementation Data Type *float32* ⌈ The primitive Implementation Data Type *float32* is classified by the category VALUE and directly refers the *float32* SwBaseType in its SwDataDefProps. ⌋*(RS_AP_00111)*

Listing 2.31: Float32 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>float32</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            float32</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>Platform_Types.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00044] SwBaseType *float32* ⌈ The *float32* SwBaseType is classified by the category FIXED_LENGTH and has the nativeDeclaration set to *float*. The baseTypeSize is set to 32 bit and the baseTypeEncoding to IEEE754 [5]. ⌋ *(RS_AP_00111)*

Listing 2.32: Float32 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>float32</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>32</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>IEEE754</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>32</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>float</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *float32* ImplementationDataType will be mapped in C++ to the single precision IEEE754 32 bit floating point type.

[SWS_APT_00045] Platform specific settings in SwBaseType *float32* ⌈ The setting of memAlignment and byteOrder in SwBaseType *float32* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for memAlignment and byteOrder that are shown in the listing above are only exemplary.

2.4.2 float64

[SWS_APT_00046] primitive Implementation Data Type *float64* ⌈ The primitive Implementation Data Type *float64* is classified by the `category` VALUE and directly refers the *float64* `SwBaseType` in its `SwDataDefProps`. ⌋*(RS_AP_00111)*

Listing 2.33: Float64 ImplementationDataType

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>float64</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_Platform/BaseTypes/
            float64</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
  <TYPE-EMITTER>Platform_Types.h</TYPE-EMITTER>
</IMPLEMENTATION-DATA-TYPE>
```

[SWS_APT_00047] `SwBaseType` *float64* ⌈ The *float64* `SwBaseType` is classified by the `category` FIXED_LENGTH and has the `nativeDeclaration` set to *double*. The `baseTypeSize` is set to 64 bit and the `baseTypeEncoding` to IEEE754 [5]. ⌋ *(RS_AP_00111)*

Listing 2.34: Float64 SwBaseType

```
<SW-BASE-TYPE>
  <SHORT-NAME>float64</SHORT-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <BASE-TYPE-SIZE>64</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>IEEE754</BASE-TYPE-ENCODING>
  <MEM-ALIGNMENT>64</MEM-ALIGNMENT>
  <BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</BYTE-ORDER>
  <NATIVE-DECLARATION>double</NATIVE-DECLARATION>
</SW-BASE-TYPE>
```

The *float64* `ImplementationDataType` will be mapped in C++ to the double precision IEEE754 64 bit floating point type.

[SWS_APT_00048] Platform specific settings in `SwBaseType` *float64* ⌈ The setting of `memAlignment` and `byteOrder` in `SwBaseType` *float64* is platform-specific. ⌋ *(RS_AP_00111)*

It means that the values for `memAlignment` and `byteOrder` that are shown in the listing above are only exemplary.

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *BaseTypeDefinition*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
baseTypeEncoding	BaseTypeEncodingString	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
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:** xml.sequenceOffset=110
maxBaseTypeSize	PositiveInteger	0..1	attr	Describes the maximum length of the BaseType in bits. **Tags:** atp.Status=obsolete xml.sequenceOffset=80
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

Attribute	***Type***	***Mul.***	***Kind***	***Note***

nativeDeclaration	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"`
				`nativeDeclaration:  "unsigned short"`
				Results in
				`typedef unsigned short MyUnsignedInt;`
				If the attribute is not defined the referring ImplementationDataTypes will not be generated as a typedef by RTE.
				If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseTypeSize.
				This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.
				Tags: xml.sequenceOffset=120

Table A.1: BaseTypeDirectDefinition

Attribute	**Type**	**Mul.**	**Kind**	**Note**
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, *AbstractEvent*, *AbstractServiceInstance*, Action, *ActionItem*, ActionList, *AdaptiveModuleInstantiation*, AdaptiveSwcInternalBehavior, AliveSupervision, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, Arbitration, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CppImplementationDataTypeElement, CryptoJob, CryptoKeySlot, CryptoNeedToCryptoJobMapping, CryptoPrimitive, DataPrototypeGroup, DataTransformation, DeadlineSupervision, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DoIpLogicAddress, E2EProfileConfiguration, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImplementationDataTypeElement, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, LocalSupervision, LogicalExpression, LogicalSupervision, MacMulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduToFrameMapping, PduTriggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, PresharedKeyIdentity, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, RestResourceDef, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, Rule, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCallPoint*, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElementMapping*, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, *ServiceMethodDeployment*, *ServiceNeeds*, SignalBasedFieldToISignalTriggeringMapping, SocketAddress, SomeipEventGroup, SomeipProvidedEventGroup, *SpecElementReference*, *StackUsage*, StartupConfig, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TcpOptionFilterList, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource,			

Attribute	***Type***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagraph	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
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
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** 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
introduction	Documentation Block	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

Attribute	***Type***	***Mul.***	***Kind***	***Note***
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.2: 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*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
subElement (ordered)	ImplementationDataTypeElement	*	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 ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	***Type***	***Mul.***	***Kind***	***Note***
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=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.3: ImplementationDataType

Class	***ImplementationProps*** **(abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
symbol	CIdentifier	1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table A.4: ImplementationProps

Class	***Referrable*** **(abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, DiagnosticDebounceAlgorithmProps, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, EventHandler, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mul.***	***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. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Attribute	***Type***	***Mul.***	***Kind***	***Note***

Table A.5: 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.Status=draft; atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
optionalElement	ServiceInterfaceSubElement	*	aggr	This aggregation represents the collectionof optional elements within the scope of the enclosing ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=optionalElement, variationPoint.shortLabel; atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	This represents the collection of ApplicationErrors defined in the context of the enclosing ServiceInterface. **Tags:** atp.Status=draft

Table A.6: ServiceInterface

Class	**SwBaseType**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *BaseType*, *Collectable Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.7: SwBaseType

Class	≪**atpVariation**≫ **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

Attribute	***Type***	***Mul.***	***Kind***	***Note***
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210

Attribute	***Type***	***Mul.***	***Kind***	***Note***
implementationDataType	AbstractImplementationDataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTargetProps), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70

Attribute	*Type*	*Mul.*	*Kind*	*Note*
swCalprmAxisSet	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false
swDataDependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220; xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntendedResolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolationMethod	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250

Attribute	Type	Mul.	Kind	Note
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefreshTiming	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisDataType	ApplicationPrimitiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table A.8: SwDataDefProps

B History of Specification Items

B.1 Constraint and Specification Item History of this document according to AUTOSAR Release 17-10

B.1.1 Added Traceables in 17-10

Number	Heading
[SWS_APT_00001]	primitive Implementation Data Type *sint8*
[SWS_APT_00002]	SwBaseType *sint8*
[SWS_APT_00003]	Platform specific settings in SwBaseType *sint8*
[SWS_APT_00004]	primitive Implementation Data Type *sint16*
[SWS_APT_00005]	SwBaseType *sint16*
[SWS_APT_00006]	Platform specific settings in SwBaseType *sint16*
[SWS_APT_00007]	primitive Implementation Data Type *sint32*
[SWS_APT_00008]	SwBaseType *sint32*
[SWS_APT_00009]	Platform specific settings in SwBaseType *sint32*
[SWS_APT_00010]	primitive Implementation Data Type *sint64*
[SWS_APT_00011]	SwBaseType *sint64*
[SWS_APT_00012]	Platform specific settings in SwBaseType *sint64*
[SWS_APT_00013]	primitive Implementation Data Type *sint8_least*
[SWS_APT_00014]	SwBaseType *sint8_least*
[SWS_APT_00015]	Platform specific settings in SwBaseType *sint8_least*
[SWS_APT_00016]	primitive Implementation Data Type *sint16_least*
[SWS_APT_00017]	SwBaseType *sint16_least*
[SWS_APT_00018]	Platform specific settings in SwBaseType *sint16_least*
[SWS_APT_00019]	primitive Implementation Data Type *sint32_least*
[SWS_APT_00020]	SwBaseType *sint32_least*
[SWS_APT_00021]	Platform specific settings in SwBaseType *sint32_least*
[SWS_APT_00022]	primitive Implementation Data Type *uint8*
[SWS_APT_00023]	SwBaseType *uint8*
[SWS_APT_00024]	Platform specific settings in SwBaseType *uint8*
[SWS_APT_00025]	primitive Implementation Data Type *uint16*
[SWS_APT_00026]	SwBaseType *uint16*
[SWS_APT_00027]	Platform specific settings in SwBaseType *uint16*
[SWS_APT_00028]	primitive Implementation Data Type *uint32*
[SWS_APT_00029]	SwBaseType *uint32*
[SWS_APT_00030]	Platform specific settings in SwBaseType *uint32*
[SWS_APT_00031]	primitive Implementation Data Type *uint64*
[SWS_APT_00032]	SwBaseType *uint64*
[SWS_APT_00033]	Platform specific settings in SwBaseType *uint64*

Number	Heading
[SWS_APT_00034]	primitive Implementation Data Type *uint8_least*
[SWS_APT_00035]	SwBaseType *uint8_least*
[SWS_APT_00036]	Platform specific settings in SwBaseType *uint8_least*
[SWS_APT_00037]	primitive Implementation Data Type *uint16_least*
[SWS_APT_00038]	SwBaseType *uint16_least*
[SWS_APT_00039]	Platform specific settings in SwBaseType *uint16_least*
[SWS_APT_00040]	primitive Implementation Data Type *uint32_least*
[SWS_APT_00041]	SwBaseType *uint32_least*
[SWS_APT_00042]	Platform specific settings in SwBaseType *uint32_least*
[SWS_APT_00043]	primitive Implementation Data Type *float32*
[SWS_APT_00044]	SwBaseType *float32*
[SWS_APT_00045]	Platform specific settings in SwBaseType *float32*
[SWS_APT_00046]	primitive Implementation Data Type *float64*
[SWS_APT_00047]	SwBaseType *float64*
[SWS_APT_00048]	Platform specific settings in SwBaseType *float64*
[SWS_APT_00049]	primitive Implementation Data Type *boolean*
[SWS_APT_00050]	SwBaseType *boolean*
[SWS_APT_00051]	Platform specific settings in SwBaseType *boolean*

Table B.1: Added Traceables in 17-10

B.1.2 Changed Traceables in 17-10

none

B.1.3 Deleted Traceables in 17-10

none

B.2 Constraint and Specification Item History of this document according to AUTOSAR Release 18-03

B.2.1 Added Traceables in 18-03

none

B.2.2 Changed Traceables in 18-03

Number	Heading

Number	Heading
[SWS_APT_00003]	Platform specific settings in `SwBaseType` *sint8*
[SWS_APT_00015]	Platform specific settings in `SwBaseType` *sint8_least*
[SWS_APT_00024]	Platform specific settings in `SwBaseType` *uint8*
[SWS_APT_00036]	Platform specific settings in `SwBaseType` *uint8_least*

Table B.2: Changed Traceables in 18-03

B.2.3 Deleted Traceables in 18-03

none