Document Title	Macro Encapsulation of Interpolation Calls
Document Category	Explanation (EXP)
Document UID	808
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• No content changes
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• No content changes
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes

△

2016-11-30	4.3.0	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

1 Acronyms and abbreviations 5
2 Related documentation 6
2.1 Input documents & Related specification 6
3 Introduction 7
4 Motivation 8
5 Disclaimer 9
6 Use Cases 10
6.1 Generate Encapsulation Macros 10
6.2 Use Encapsulation Macros 11
7 Solution Proposal 13
7.1 Definition of Terminology 13
7.2 Architectural Components 13
7.2.1 Encapsulation Macros Header File 13
7.3 Functional Description 13
7.3.1 Basic Concept Description 13
7.3.1.1 Principle of Encapsulation Concept 13
7.3.1.2 Concept Decision 14
7.3.1.3 Needed Information for the Macro Generation 15
7.3.1.4 Overview to get the Information for Macro Generation 16
7.3.1.5 Non-Ambiguous InterpolationRoutineMapping 18
7.3.1.6 General Information to BswModuleEntry 18
7.3.1.7 Interpolation Routine and Record layouts 22
7.3.1.8 Structure of the Name of a Interpolation Routine 23
7.3.1.9 Data Type of the Number of Axis Points 24
7.3.2 Implementation of Macro Encapsulation Concept 26
7.3.2.1 Generation of the Name of the Encapsulation Macro 26
7.3.2.2 Generation of the Name of the Interpolation Routine 27
7.3.2.3 Generation of the Parameters of the Interpolation Routine for ImplementationDataType of Category STRUCTURE 27
7.3.2.4 Generation of the Parameters of the Interpolation Routine for ImplementationDataType of Category ARRAY 28
A Mentioned Class Tables 30

1 Acronyms and abbreviations

Abbreviation / Acronym:	*Description:*
DEM	Diagnostic Event Manager
DET	Default Error Tracer

2 Related documentation

2.1 Input documents & Related specification

[1] Methodology for Classic Platform
AUTOSAR_CP_TR_Methodology

[2] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[3] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

3 Introduction

Interpolation routines are used by the application software for calculating the unknown points from the known points. The existing AUTOSAR interpolation routines supports two categories **curve (1D) and map (2D)** interpolation functionalities both in **integer and floating point**. It supports two methods per category **interpolation and lookup**. Additionally special variants called **group of curves/maps and fixed curves/maps** with two different calculation formulas are supported which can be either interpolation (or) lookup.

Interpolation routines are very frequently used routines in the application software. As a consequence, the design of interpolation routines has a significant impact on the efforts of software development and will be first addressed by optimization. The explanatory document "Macro Encapsulation of Interpolation Calls" is developed to guide the Application Developer to perform the simplified invocation of the AUTOSAR compatible and resource optimized interpolation routines.

4 Motivation

The motivation for the explanatory document "MacroEncapsulationofInterpolation-Calls" is to simplify the routine handling by introducing a single source principle. This will reduce the maintenance efforts and avoid false usage leading to bugs. They are the basis for the resulting cost reduction and increase in quality.

5 Disclaimer

This explanatory document represents the macro encapsulation of library calls as one of the possible methods to reduce the application overhead in calling the mathematical interpolation functionalities. This document does not mandate that the user shall use only macro encapsulation for making the interpolation calls.

6 Use Cases

6.1 Generate Encapsulation Macros

The document AUTOSAR_CP_TR_Methodology [1] illustrates the general approach of generation of atomic software component header files (Figure 6.1). The proposed encapsulation macros shall be saved in an "Encapsulation Macros Header File" similar to an "Application Header File".

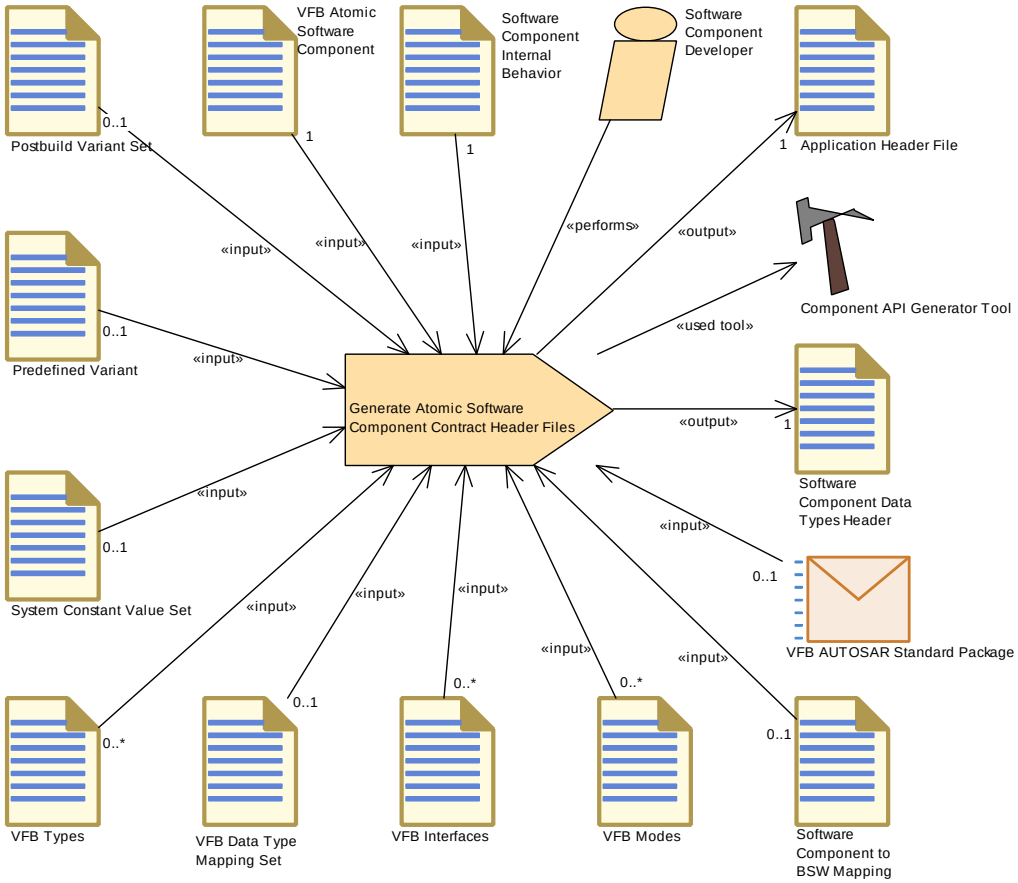

Figure 6.1: Generate Atomic Software Component Contract Header Files

Figure 6.2 shows the generation process which is parallel to the generation process of the application header file. The marked block suggests the new part. The Macro Encapsulation Generator Tool can be implemented as add-on to the Component API Generation Tool (RTE [2]). Inputs of both tools are information from the **VFB Atomic Software Component** and the **Software Component Internal Behaviour**.

The generated encapsulation macros need interfaces from the application header file e.g. to get access to the curve and maps. Therefore the macro encapsulation concept has to know the syntax and structure of the RTE [2] generated interfaces.

The proposed encapsulation macros shall be saved in an "Encapsulation Macros Header File" (see Figure 6.2) similar to an "Application Header File".

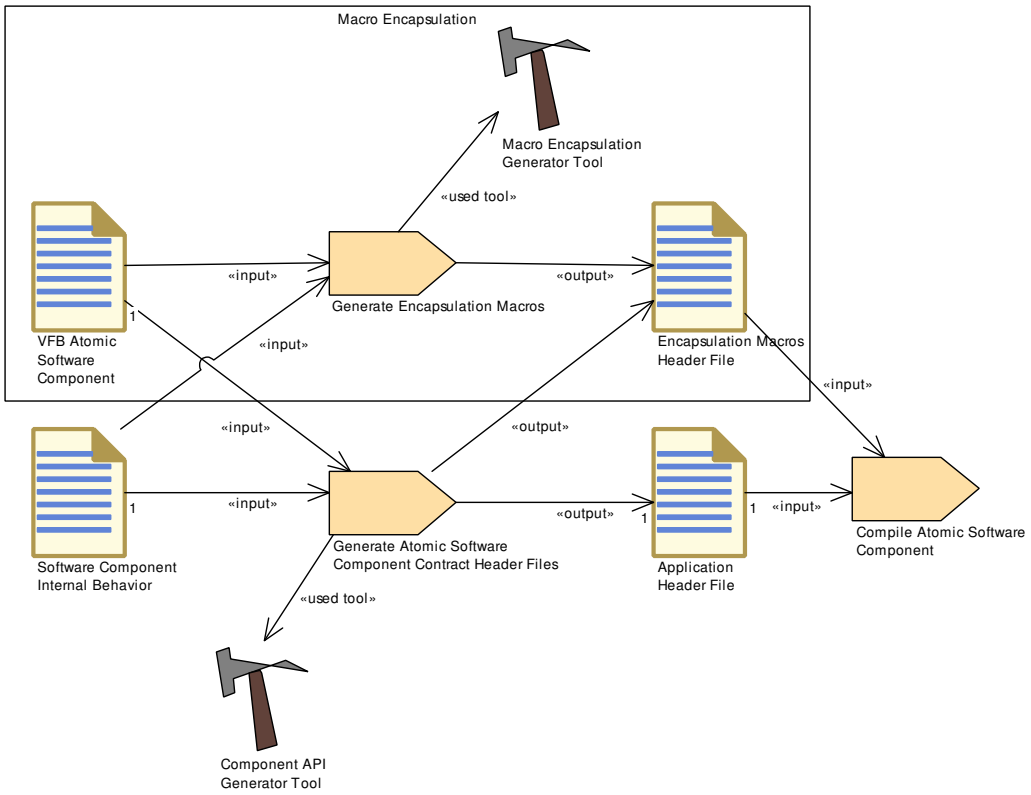

Figure 6.2: Generation Process of Encapsulation Macros Header File

6.2 Use Encapsulation Macros

Following matrix show all types of interpolations services, provided by IFX Libraries, which shall be handled by the macro encapsulation concept:

	Linear	*Lookup*	*Fix (interval)*	*Fix (shift)*	*Lookup Fix (interval)*	*Lookup Fix (shift)*
Curve	x	x	x	x	x	x
Map	x	x	x	x	x	x
Grouped Curve	x	x				
Grouped Map	x	x				
Axis Search	x					

Following matrix show all types of interpolations services, provided by IFL Libraries, which shall be handled by the macro encapsulation concept:

	Linear	*Lookup*	*Fix (interval)*	*Fix (shift)*	*Lookup Fix (interval)*	*Lookup Fix (shift)*
Curve	x					
Map	x					

▽

△

Grouped Curve	x					
Grouped Map	x					
Axis Search	x					

Interpolation methods:

Linear: Interpolates result considering two data points

Lookup: No interpolation, returns entry data point

Fix: No explicit axis available, distribution points are calculated via Offset and Shift or Offset and Interval

Lookup Fix: Mixture of Lookup and Fix

Interpolation calculation:

Curve / **Map:** Integrated data point search and interpolation

Grouped Curve / **Grouped Map:** Distributed data point search and interpolation

For the grouped interpolation method the data point search is separated from the interpolation calculation. The data point search results in a structure which contains index and ratio information. This information can be used by curve interpolation, curve look-up interpolation, map interpolation and map look-up interpolation.Currently this document details on linear curve and map interpolations. The other types of interpolations can be handled similar but are not specified in this document.

7 Solution Proposal

7.1 Definition of Terminology

This concept will provide an additional header file, the "Encapsulation Macros Header File". It contains generated macros to encapsulate the call of curve and map interpolation routines.

There are no other new terminologies provided.

7.2 Architectural Components

7.2.1 Encapsulation Macros Header File

Artifact	**Encapsulation Macros Header File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Header file for macros that encapsulate the call of curve and map interpolation routines		
Description	Header generated for an AtomicSoftwareComponentType from Macro Encapsulation Generator Tool. It represents the complete encapsulation macro interfaces between the component code and the RTE (calls into the RTE as well as prototypes called by the RTE). All calls of encapsulation interpolation routines are routed through this header		
Kind	Code		
Relation Type	***Related Element***	***Mult.***	***Note***
Aggregated by	Delivered Atomic Software Components	1	
Produced by	Generate Atomic Software Component Contract Header Files	1	
Consumed by	Compile Atomic Software Component	1	

Table 7.1: Encapsulation Macros Header File

The name of the header will have following form: "<component>_Elc.h" where <component> is the name of the component for that the header is generated.

7.3 Functional Description

7.3.1 Basic Concept Description

7.3.1.1 Principle of Encapsulation Concept

For illustration of the encapsulation macros, an example of the processing of a curve interpolation is demonstrated as follows. (Given names are possibly not conforming to naming conventions because the focus is set to the principle of the concept.)

Suggest the data specification (**VFB Atomic Software Component** description) of a particular SWC component defines a data prototype, named "**IgnitionCurve**". This

data prototype is typed by an ApplicationDataType named "IgnitionCurveType" inclusive their x- and y-axis. The ApplicationDataType corresponds to an ImplementationDataType (e.g. "GenericCurve"). This ImplementationDataType specifies the details of the resulting structure including their **BaseTypes** (e.g. data type **uint8** for curve values, **sint16** for the x-axis used by following example).

The prototype of an interpolation service which looks like below:

```
uint8 Ifx_IntIpoCur_s16_u8(sint16 Xin, sint16 N, const sint16* X_Array,
    const uint8* Val_Array);
// where,
// Xin: Input value
// N: Number of axis points
// X_Array: Pointer to X distribution
// Val_Array: Pointer to Curve values
```

Without the encapsulation concept the interpolation service has to be called as given below:

```
CurveValue = Ifx_IntIpoCur_s16_u8(Xinput, Curve.N, Curve.Axis, Curve.Values
    );
```

The encapsulation concept now provides a macro to encapsulate the interpolation service call:

```
CurveValue = Elc_Get_myRunnable_IgnitionCurve();
```

Because the encapsulation macro is generated as below:

```
#define Elc_Get_myRunnable_IgnitionCurve Ifx_IntIpoCur_s16_u8(Xinput, Curve
    .N, Curve.Axis, Curve.Values);
```

The order of parameters is not implicit; an explicit behavior is needed via a semantic mapping (details are defined in 7.3.2.3). To provide values and pointers for single parameter of interpolation service, RTE [2] accesses are used. `Ex:Rte_CData()`

7.3.1.2 Concept Decision

Generally there are two types of parameters:

- First type is the input values to the curve or map.
- Second type is the values and pointers to the respective curve or map.

The input values are normally derived from physical values which are represented as ApplicationDataTypes. But it is possible that such input values are slightly pre-processed before calling the interpolation routine. In this case the interpolation routine is called with local variables which are not passed through RTE [2] contract phase. An explicit communication shall be needed but would be costly regarding resources. This will make the complete encapsulation of Interpolation calls complex and should be avoided.

The parameters for the values and pointers of the respective curve or map will have a more internal view, because they are derived from the memory representation of a curve or map which is described via **RecordLayouts**.

To limit the complexity of the handling of the input values two alternatives are possible:

1. The generated macro has parameter(s) for the input value(s)

   ```
   CurveValue = Elc_Get_myRunnable_IgnitionCurve(local_input);
   ```

   or

   ```
   CurveValue = Elc_Get_myRunnable_IgnitionCurve(Rte_X_input);
   ```

2. Temporary variables are used in front of macro call without parameters

   ```
   local_input = X_input;
   ```

   or

   ```
   local_input = Rte_X_input;

   CurveValue = Elc_Get_myRunnable_IgnitionCurve();
   ```

In solution 2 there must be specific knowledge of the name of the temporary variable because this variable is fixed within the generated macro. This might be too complex and hence solution 1 is chosen.

Note, these macros are SWC specific and therefore particular naming schemes shall be applied. Only the input values of the curve or map has to be provided by the user. The remaining parameters of an interpolation routine and the interpolation routine itself are encapsulation from the generated macro. This information can be extracted from the data specification. With this approach fault introduction by non consistent definitions are eliminated. Additionally the software developer of a SWC component is freed completely from storage assignment and routine assignment which are performed automatically. As a consequence, the effort for software development decreases significantly.

7.3.1.3 Needed Information for the Macro Generation

Based on the concept decision in chapter 7, the macro to be generated looks like below.

(Example for a curve):

```
#define Elc_Get_{Runnable}_{Accesspoint} {RoutineName}((X), {ImplTypeStruct
    }.{N}, {ImplTypeStruct}.{Axis}, {ImplTypeStruct}.{Values}
```

To generate this macro following information is needed:

Name of the generated Macro: Elc_Get_myRunnable_{NameOfAccessPoint}
The generated macro is individual generated for each access point.

Name of the Interpolation Routine: {RoutineName} The name of an interpolation routine depends on the type of the interpolation routine and the data types of the axis and output values. Each combination of data types of axis and output of interpolation values has an individual implementation and an individual name of the interpolation routine. To create the name of the interpolation routine it the most complex part of this concept.

E.g. following curve interpolation routines has to be distinguished:

Ifx_IntIpoCur_U8_U8
Ifx_IntIpoCur_U8_U16
Ifx_IntIpoCur_U8_S8
Ifx_IntIpoCur_U8_S16
Ifx_IntIpoCur_U16_U8
Ifx_IntIpoCur_U16_U16
Ifx_IntIpoCur_U16_S8
Ifx_IntIpoCur_U16_S16
Ifx_IntIpoCur_S8_U8
Ifx_IntIpoCur_S8_U16
Ifx_IntIpoCur_S8_S8
Ifx_IntIpoCur_S8_S16
Ifx_IntIpoCur_S16_U8
Ifx_IntIpoCur_S16_U16
Ifx_IntIpoCur_S16_S8
Ifx_IntIpoCur_S16_S16

Parameters of the Interpolation Routine: {ImplTypeStruct}.{N}, ... Provision of the parameter of the interpolation routine. RTE [2] generates a structure corresponding to an `ImplementationDataType` and based on a `SwRecordLayout`. The macro encapsulation tool has to generate the accesses to the number of axis points, the axis and the values of the curve or map. The number of pointers needed from the interpolation routine differs from kind of interpolation.

The data type of the number of axis points has a special relevance. This information is not needed explicitly but must be defined strictly within the `ImplementationDataType`. In chapter 7.3.1.9 rules are given to define the data type of the number of distribution points.

7.3.1.4 Overview to get the Information for Macro Generation

Figure 7.1 illustrates a rough overview of the workflow of the Macro Encapsulation Concept. The picture anticipates which information has to be prepared by the concept and which information are still available within the MetaModel of AUTOSAR.

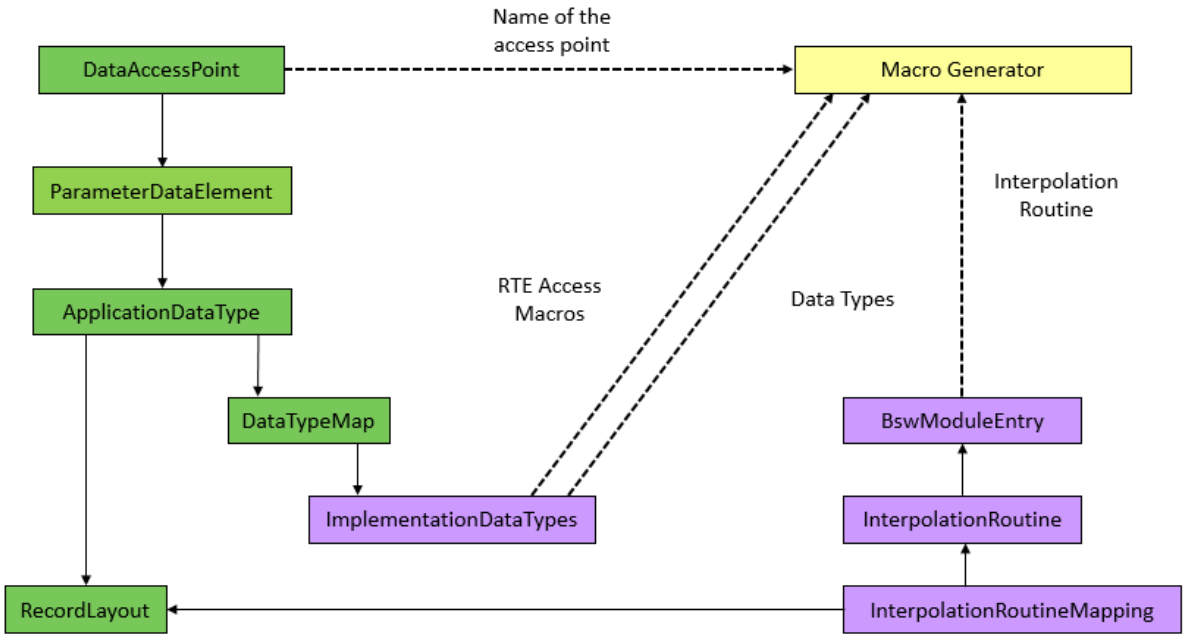

Figure 7.1: Overview of Workflow of Encapsulation Concept based on Meta Model

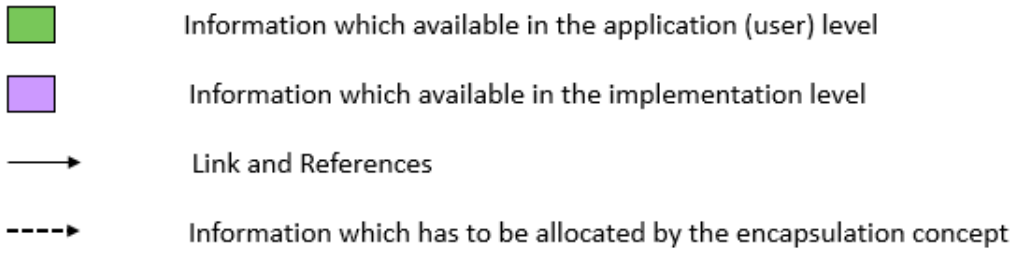

Figure 7.2: Legend

Starting from the **DataAccessPoint** all information has to be collected to generate a macro which encapsulates the call of an interpolation routine. At the **DataAccessPoint** it is known what interpolation routine will be used and which values shall be the input and output of the interpolation routine. The name of the access point can be chosen directly from the **DataAccessPoint**. The name of the interpolation routine is taken from the `BswModuleEntry`. The `BswModuleEntry` is related to the **DataAccessPoint** via `InterpolationRoutineMapping`, **RecordLayout** and `ApplicationDataTypes`. RTE [2] access macros and data types can be derived from the `ImplementationDataTypes` which are linked to a **DataAccessPoint** over `DataTypeMap` and `ApplicationDataTypes`.

Interpolation routines varies depending on data types of the input and output values.

Up to now no AUTOSAR SWS describes the complete mechanism to specify a BswModuleEntry with an interpolation routine for corresponding to **Application-Datatypes**, **SwRecordlayouts** and ImplementationDataTypes. In order that the Macro Encapsulation Concept can use the content of the BswModuleEntry it has to be defined. A concept how to do that is described in the next chapter.

7.3.1.5 Non-Ambiguous InterpolationRoutineMapping

There are scenarios where the InterpolationRoutineMapping is not ambiguous and the same RecordLayout fits to more than one Interpolation function. In this scenario from point of data specification it is not clear for the macro encapsulation tool to find out which kind interpolation routine is used. A curve or map can be interpolated or only the lookup behavior can be used. The reason here is the data of the curve or map in memory are still identical in both cases. The user only specifies the data and properties of the curve or map in ARXML and the kind of interpolation is than chosen by the call of a related interpolation routine in code.

For example, Ifx_IntIpoCur_s16_s16 and Ifx_IntLkUpCur_s16_s16.

The possible solution for such a non-ambiguous scenario would be, the macro encapsulation tool generates more than one macros for different interpolation routines. In the case the macros shall have different names to distinguish the different kinds of interpolation routines.

Example, consider Ifx_IntIpoCur_s16_s16 and Ifx_IntLkUpCur_s16_s16,

```
#define Elc_Get_myRunnable_IgnitionCurve_Ipo Ifx_IntIpoCur_s16_s16(X_input,
    Curve.N, Curve.Axis, Curve.Values);

#define Elc_Get_myRunnable_IgnitionCurve_Lkup Ifx_IntLkUpCur_s16_s16(
   X_input, Curve.N, Curve.Axis, Curve.Values);
```

The user can now invoke,

```
CurveValue = Elc_Get_myRunnable_IgnitionCurve_Ipo(); // for Interpolation
   method
// or
CurveValue = Elc_Get_myRunnable_IgnitionCurve_Lkup(); // for Lookup method
```

7.3.1.6 General Information to BswModuleEntry

The BswModuleEntry represents a single API entry (C-function prototype) into the BSW module or cluster. For IFX and IFL the BswModuleEntry is the reference to the interpolation routine and derived from the APIs of the interpolation defined from AUTOSAR in the SWS documents.

For Example, the IntIpoCur_u16_u16 corresponds to the API Ifx_IntIpoCur_u16_u16.

More information is available in the AUTOSAR blueprint files in "AUTOSAR_MOD_GeneralBlueprints.zip" in below files.

AUTOSAR_MOD_BswModuleEntrys_Blueprint.arxml

AUTOSAR_MOD_IFX_RecordLayout_Blueprint.arxml

AUTOSAR_MOD_IFL_RecordLayout_Blueprint.arxml

Figure 7.3 and Figure 7.4 describes the complete overview with different focus.

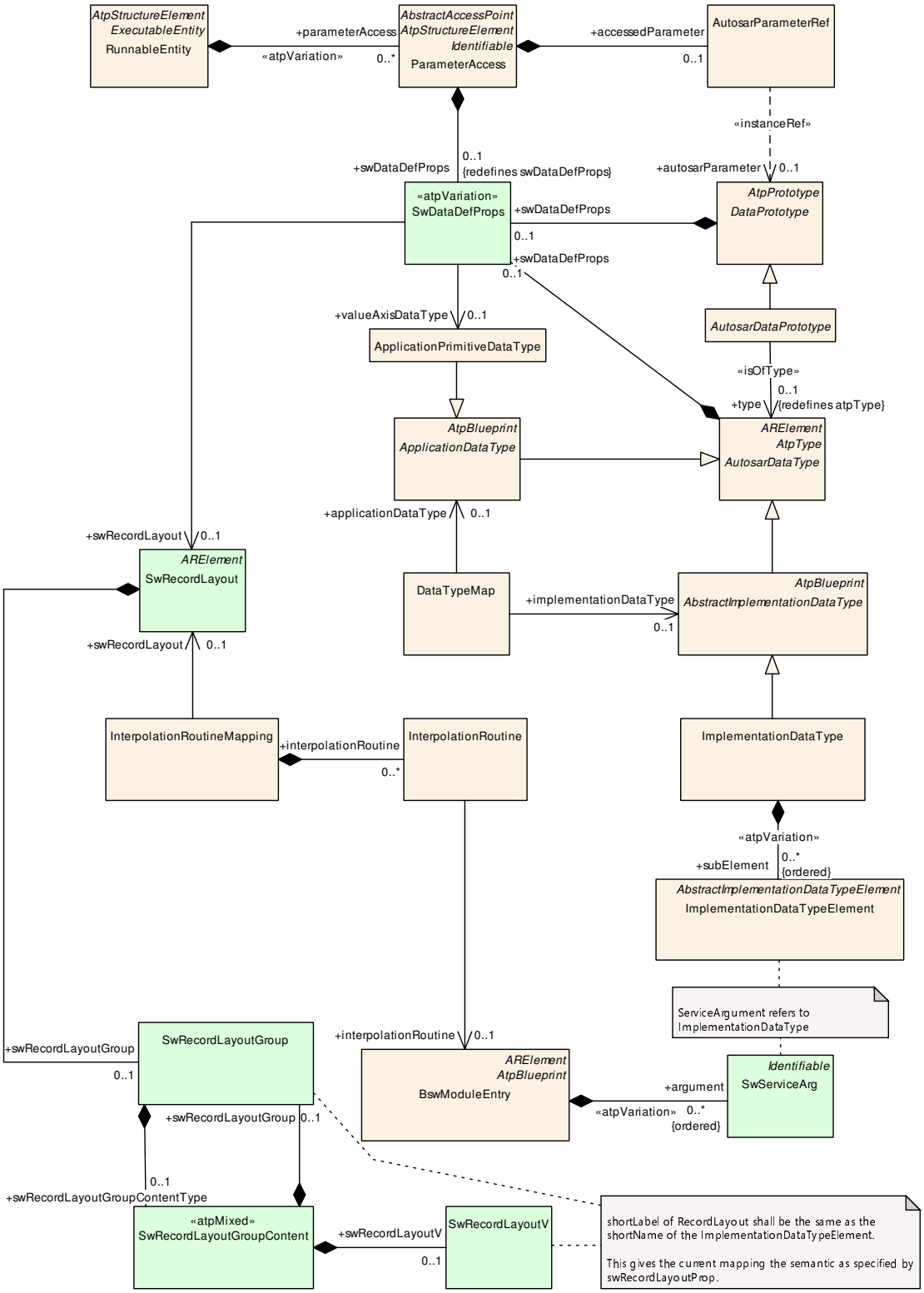

Figure 7.3: Complete MetaModel Overview to Find the Correct BswModuleEntry

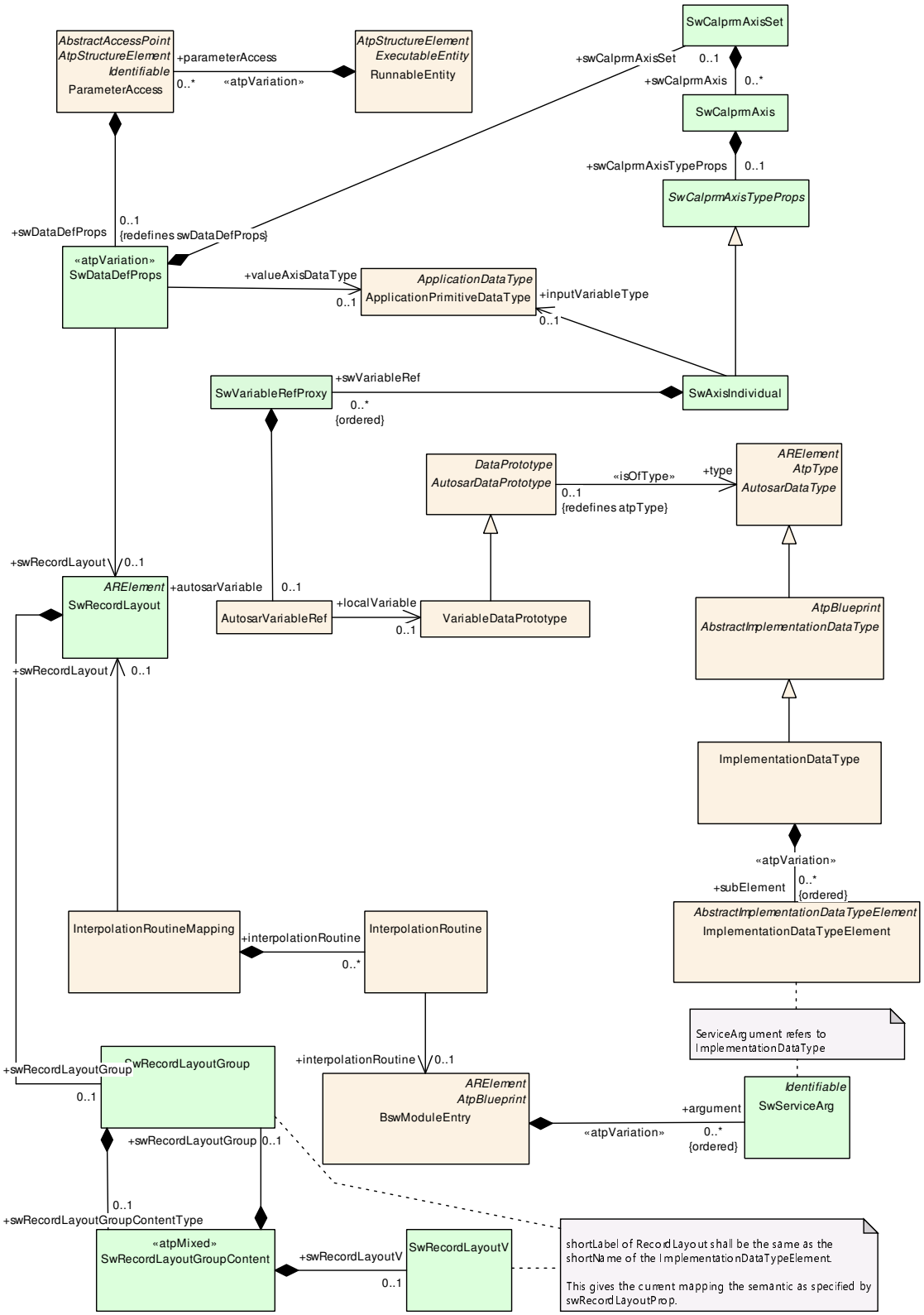

Figure 7.4: Complete MetaModel Overview to Find the Correct BSWModuleEntry with Focus SwCalprms

7.3.1.7 Interpolation Routine and Record layouts

The relationship between record layouts and interpolation routines is specified in `InterpolationRoutineMappingSet`. The interpolation routine is represented as `BswModuleEntry` and implements a particular interpolation method which is denoted in shortLabel of `InterpolationRoutine`. The intended interpolation method is denoted in InterpolationMethod of SwDataDefProps.

Figure 7.5 shows the MetaModel of mapping a Record Layout to a specific interpolation routine (**Note**: This picture is taken from AUTOSAR_TPS_SoftwareComponentTemplate Description, Figure 5.53 [3]).

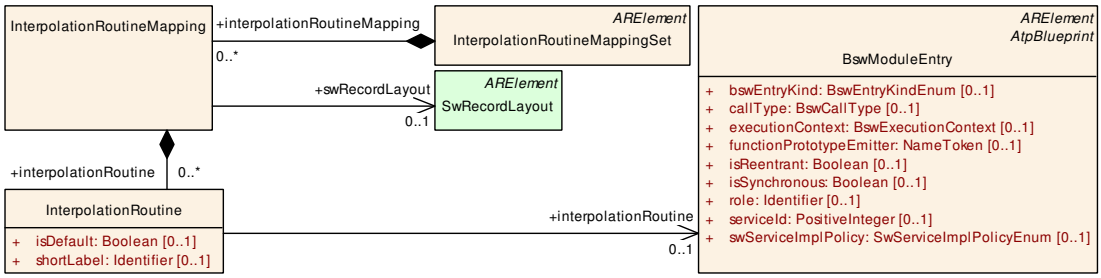

Figure 7.5: Mapping of Record Layouts and Interpolation Routines

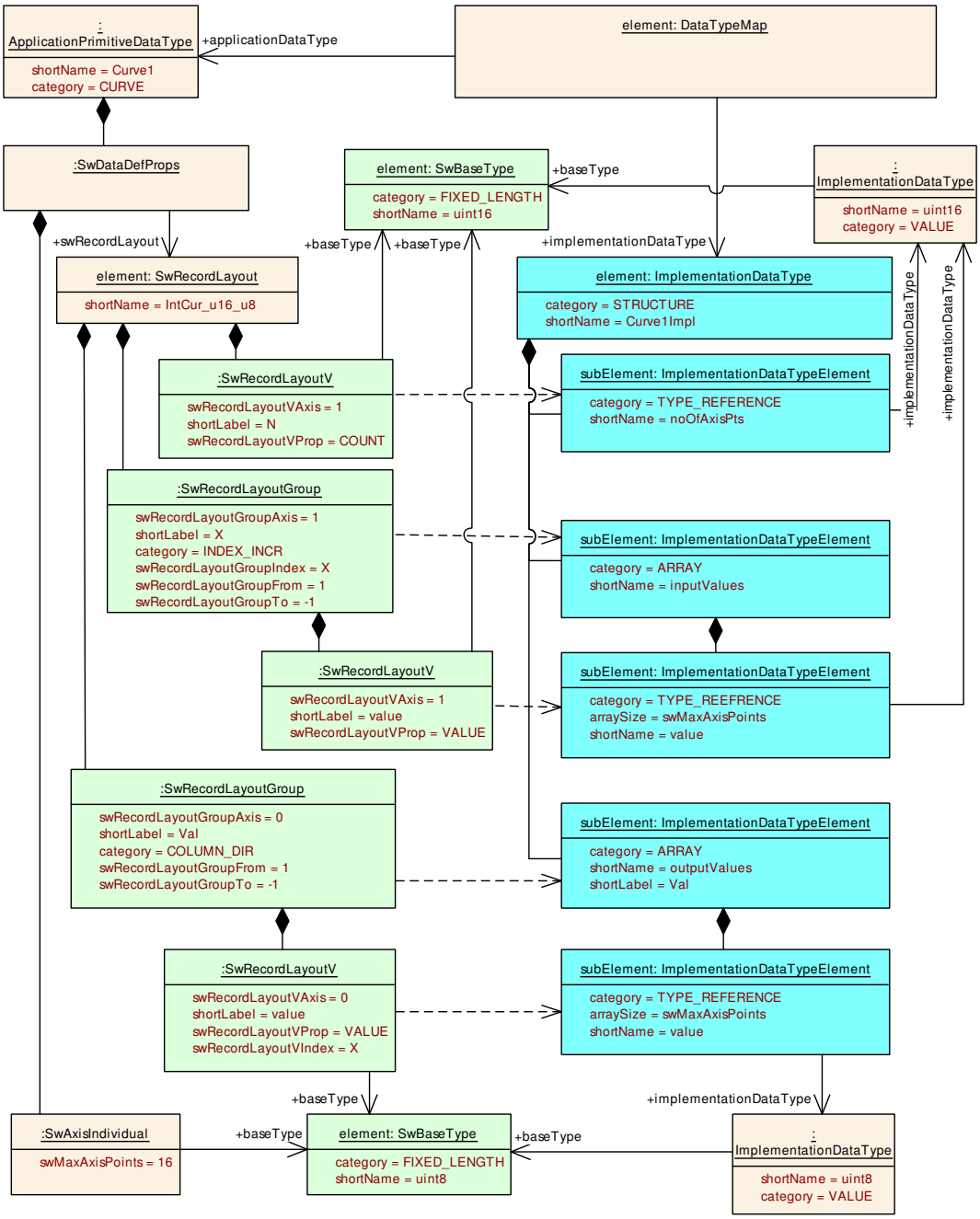

Figure 7.6: Curve implemented as two consecutive arrays

The structure and memory representation of a curve or map is described on data specification level via **RecordLayout**. Figure 7.6 is taken from the AUTOSAR_TPS_SoftwareComponentTemplate, Figure 5.48 [3].

7.3.1.8 Structure of the Name of a Interpolation Routine

The name of the interpolation routine has a defined build convention based on an inherent semantic.

Examples:

- `Ifx_IntIpoCur_u8_s8`
- `Ifl_IntIpoMap_f32f32_f32`

The structure of a name looks as follows:

`{ModuleID}_{Method}{Type}_{InputDataType(s)}_{OutputDataType}`

The single naming parts are described as follows:

{ModuleID} Only two module IDs are possible: "Ifx" for integer interpolations and "Ifl" for float interpolations. A mix of integer and float interpolations is not intended.

{Method} There are different methods available. A translation map is suggested to get a mapping between a specific method and the method part of the name of the interpolation routine. The method is described within `ApplicationDataType`.interpolationMethod. E.g. Linear IntIpo, Lookup IntLkUp

{Type} If the interpolation has to be done for a curve or map can be chosen via category of the `ApplicationDataType`.category. Category CURVE Cur, MAP Map

{InputDataType(s)} With the help of the `ImplementationDataTypeElements` the data types for the inputs are identified. Additionally the types of the axis can be derived via `DataTypeMap` from the DataTypes of the `ApplicationDataTypes`.valueAxisDataType. Figure 7.7 visualizes the dependency between DataTypes and `SwRecordLayouts` and is taken from AUTOSAR_TPS_SoftwareComponentTemplate figure 5.33 [3].

Hint: The data type of the axis values may be different from the data type of the input value of the curve.

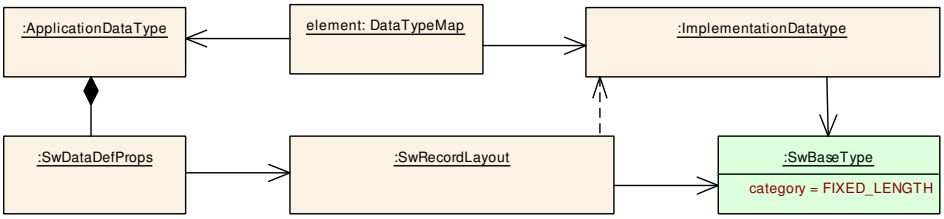

Figure 7.7: Dependency of DataTypes and SwRecordLayouts

{OutputDataType} The output data type depends on the data type of the access point.

With that principle the `BswModuleEntry` can be filled inside the `InterpolationRoutineMapping`. The macro encapsulation generator tool can assume that a name of an interpolation routine exists inside the `BswModuleEntry`.

7.3.1.9 Data Type of the Number of Axis Points

The macro encapsulation concept does not need this data type explicitly but the interpolation routine applies a special data type for the parameter for the number of axis

points. Additionally the number of axis point is an element which is located in memory as well as the axis and values of a curve or map. Therefore the data type for the number of axis points has to be defined when the ImplementationDataType is derived from an ApplicationDataType.

The rule to determine the data type for the number of axis points is quite easy:

The number of axis points gets the same data type as the first axis.

Impacts for curves:

A curve has only one axis. Therefore the number of axis points gets the same data type as the x axis. If the x axis is a **sint8** axis the number of axis points will be of data type **sint8** too. It is clear that negative numbers of axis points makes no sense but 127 axis points should be sufficient. If the axis is from **uint8**, **sint16** or **uin16** type the number of axis points use the same data types too.

Impacts for maps:

A map has two axes. Here the number of axis points of the x and y axes gets the data type of the x axis. The reason for this is to avoid fill bytes within definition of ImplementationDataType. To understand this point further a definition has to be made. The order of elements within an ImplementationDataType has a well defined sequence. First the elements with the number of axis points have to be defined, than the axis/axes and finally the values of the curve or map are defined. The implementation of an ImplementationDataType can be done as structure or array. As example:

```
Struct
{
  uint8 Nx;
  uint8 Ny;
  uint8 AxisX[];
  uint16 AxisY[];
  sint8 Values[];
} Map;
```

Assuming a processor with natural alignment ("**naturally aligned**" means that any element is aligned to at least a multiple of its own size. For example, a 4-byte object is aligned to an address that's a multiple of 4, an 8-byte object is aligned to an address that's a multiple of 8, etc.) of memory elements no gap byte is needed between Nx and Ny. If Ny has the same type as the Y axis between Nx and Ny is a fill byte.

7.3.2 Implementation of Macro Encapsulation Concept

This chapter describes how the encapsulation macros will be generated and the needed information is picked up. This chapter refers to chapter 7.3.1.3 where the needed information for the macro encapsulations is described.

Three parts have to be generated:

- Name of the encapsulation macro
- Name of the interpolation routine
- Parameters of the interpolation routine

Abstract form of the generated macro:

```
#define {NameOfMacro} {RoutineName}((X),{Parameters})
```

Details of the generated macro (Example using a curve):

```
#define Elc_Get_{Runnable}_{NameOfAcessPoint} {RoutineName}(X)((X), {
   RteAccess}.{N}, {RteAccess}.{Axis}, {RteAccess}.{Values}
```

7.3.2.1 Generation of the Name of the Encapsulation Macro

The name of the encapsulation macro is derived from the name of the access point and a suffix according to the pattern:

```
Elc_Get_{NameOfRunnable}_{NameOfAcessPoint}
```

In this context Figure 7.8 shows the runnable access to a calibration port. This picture is taken from AUTOSAR_TPS_SoftwareComponentTemplate, 7.29 [3].

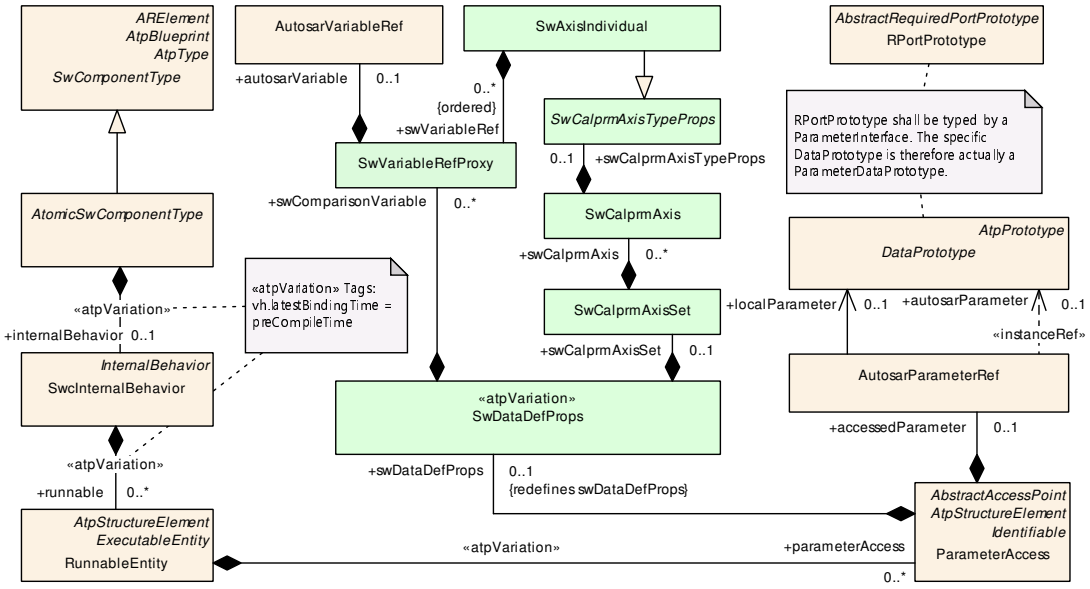

Figure 7.8: Runnable Access to a Calibration Port

7.3.2.2 Generation of the Name of the Interpolation Routine

The name of the interpolation routine is defined in the MetaModel as `BswModuleEntry`. The Macro Encapsulation Generator Tool has to parse the MetaModel in following sequence to get the name of the interpolation routine:

1. Start at **DataAccess** -> `RunnableEntity` -> `ParameterAccess`
2. Via `AutosarParameterRef` the `DataPrototype` can be found
3. Via `AutosarDataPrototype` the `AutosarDataType` can be found
4. The `AutosarDataType` has a relation to `SwDataDefProps`
5. Via `SwDataDefProps` a `SwRecordLayout` is chosen
6. Via `SwRecordLayout` and `InterpolationRoutineMapping` and `InterpolationRoutine` the needed interpolation routine candidate's call can be found in `BswModuleEntry`.
7. Finally the appropriate `InterpolationRoutine` is then determined by matching the data types of the `ImplementationDataType`.

The structure of a name looks as follows:

{ModuleID}_{Method}{Type}_{InputDataType(s)}_{OutputDataType}
———— 6 ———- ———————- 7 ————

7.3.2.3 Generation of the Parameters of the Interpolation Routine for ImplementationDataType of Category STRUCTURE

As decided in the concept decision in chapter 7.3.1.2 input variables for the curve or map interpolation are not encapsulated. In general they are available over **DataAccess.dataDefProperties.swCalprmAxisSet.variableRef**.

Only the parameters for the number of axis points, pointer to the axis and pointer to the curve or map values are generated. To get these parameters RTE [2] generated information is used.

The RTE generates typedefs and structures depending on `ImplementationDataTypes` which are the based on `SwRecordLayouts` of the corresponding curves or maps. The Macro Encapsulation Generator Tool has to know the same methods like the RTE to derive a typedef and structure from an `ImplementationDataType` to be able to use that information.

By default the RTE generates for each `ImplementationDataType` with category attribute set to "STRUCTURE" following typedef in the RTE Data Type header file "Rte_Type.h". This is done in the "RTE Contract" and "RTE Generation" phase.

```
typedef struct { <elements> } <name>;
```

where **<elements>** is the record element specification and **<name>** is the **shortName** of the **Structure Implementation Data Type**. For each record element defined by one `ImplementationDataTypeElement` one record element specification **<elements>** is defined. The record element specifications are ordered according the order of the related `ImplementationDataTypeElements` in the input configuration. Sequent record elements are separated with a semicolon. It is ensured by RTE that the names of the structure and their elements are unique. The prefix Rte_ is not used because the type names representing AUTOSAR Data Types.

Based on such a typedef a located structure is generated in the Rte.c file. Standard RTE access is used to address the elements of the structure.

One point to clarify is the issue how to map the elements of the `ImplementationDataType` to the associated parameter of interpolation routine. On the one hand the elements of the `ImplementationDataType` could be defined in an arbitrary order and on the other hand the sequence of parameters of the interpolation routines is fixed. There must be a mapping that the element of the `ImplementationDataType` fits to the correct parameter of the interpolation routine. E.g. the element which describes the number of axis points must fit to the parameter of the interpolation routine with same denotation.

To handle this relation two proceedings are possible:

- Either a new map in MetaModel is needed to define the parameter sequence order regarding the corresponding elements of the `ImplementationDataTypes`
- Or a naming convention has to be defined to have well defined names for specific element behaviours.

The naming convention will be chosen because it is easier to define and to implement and the MetaModel need not be expanded. The below table shows the naming convention for the concatenation of `ImplementationDataTypes` and parameters of interpolation routines.

Parameter	Defined name
Number of x axis points	Nx
Number of y axis points	Ny
X axis	AxisX
Y axis	AxisY
Values of the curve or map	Values

7.3.2.4 Generation of the Parameters of the Interpolation Routine for ImplementationDataType of Category ARRAY

There are approaches where the `ImplementationDataType` for e.g. a Curve is not a STRUCTURE but an ARRAY. Obviously this requires that the same primitive data types are used for number of Axispoints, Axis points, Values.

Nevertheless, in this case the naming convention described in chapter 7.3.2.3 is not fully applicable. Therefore the required positions in the implementation array need to be determined by a kind of "address calculation" based on the `SwRecordLayout` and the current size of the corresponding curve / map. The location of the size element can be found according to the naming conventions in chapter 7.3.2.3 and the record layout.

A Mentioned Class Tables

Class	***ApplicationDataType*** (abstract)			
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 `ApplicationDataType`s only.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*ApplicationCompositeDataType*, ApplicationPrimitiveDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: ApplicationDataType

Class	***AutosarDataPrototype*** (abstract)			
Note	Base class for prototypical roles of an `AutosarDataType`.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ArgumentDataPrototype, ParameterDataPrototype, VariableDataPrototype			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	AutosarDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.2: AutosarDataPrototype

Class	***AutosarDataType*** (abstract)			
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. **Tags:** atp.Splitkey=swDataDefProps

Table A.3: AutosarDataType

Class	**AutosarParameterRef**			
Note	This class represents a reference to a parameter within AUTOSAR which can be one of the following use cases: localParameter: • localParameter which is used as whole (e.g. sharedAxis for curve) autosarVariable: • a parameter provided via `PortPrototype` which is used as whole (e.g. parameterAccess) • an element inside of a composite local parameter typed by `ApplicationDataType` (e.g. sharedAxis for a curve) • an element inside of a composite parameter provided via Port and typed by `ApplicationDataType` (e.g. sharedAxis for a curve) autosarParameterInImplDatatype: • an element inside of a composite local parameter typed by ImplementationDatatype • an element inside of a composite parameter provided via PortPrototype and typed by Implementation Datatype			
Base	*ARObject*			
Aggregated by	InstantiationDataDefProps.parameterInstance, ParameterAccess.accessedParameter, RoleBasedData Assignment.usedParameterElement, SwCalprmRefProxy.arParameter			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
autosar Parameter	DataPrototype	0..1	iref	This instance reference is used if the calibration parameter is either imported via a port or is part of a composite data structure. **Tags:** atp.Splitkey=autosarParameter, autosar Parameter.portPrototype, autosarParameter.root ParameterDataPrototype, autosarParameter.contextData Prototype, autosarParameter.targetDataPrototype **InstanceRef implemented by:** ParameterInAtomic SWCTypeInstanceRef
localParameter	DataPrototype	0..1	ref	In the majority of cases this reference goes to Parameter DataPrototypes rather than VariableDataPrototypes. Pointing the reference to a VariableDataPrototype is limited to special use cases, e.g. if the AutosarParameter Ref is used in the context of an SwAxisGrouped. This reference is used if the arParameter is local to the current component. Of course, it would technically also be feasible to use an InstanceRef for this case. However, the InstanceRef would not have a contextElement (because the current instance is the context). Hence, the local instance is a special case which may provide further optimization. Therefore an explicit reference is provided for this case. **Tags:** atp.Splitkey=localParameter

Table A.4: AutosarParameterRef

Class	**BswModuleEntry**
Note	This class represents a single API entry (C-function prototype) into the BSW module or cluster. The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation. **Tags:** atp.recommendedPackage=BswModuleEntrys This Class is only used by the AUTOSAR Classic Platform.

▽

△

Class	**BswModuleEntry**			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45
bswEntryKind	BswEntryKindEnum	0..1	attr	This describes whether the entry is concrete or abstract. If the attribute is missing the entry is considered as concrete. **Tags:** xml.sequenceOffset=40
callType	BswCallType	0..1	attr	The type of call associated with this service. **Tags:** xml.sequenceOffset=25
execution Context	BswExecutionContext	0..1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:** xml.sequenceOffset=30
function Prototype Emitter	NameToken	0..1	attr	This attribute is used to control the generation of function prototypes. If set to "RTE", the RTE generates the function prototypes in the Module Interlink Header File.
isReentrant	Boolean	0..1	attr	Reentrancy from the viewpoint of function callers: • true: Enables the service to be invoked again, before the service has finished. • false: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=15
isSynchronous	Boolean	0..1	attr	Synchronicity from the viewpoint of function callers: • true: This calls a synchronous service, i.e. the service is completed when the call returns. • false: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:** atp.Splitkey=returnType, returnType.shortName xml.sequenceOffset=40
role	Identifier	0..1	attr	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:** xml.sequenceOffset=10
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:** xml.sequenceOffset=5

▽

△

Class	**BswModuleEntry**			
swServiceImpl Policy	SwServiceImplPolicy Enum	0..1	attr	Denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. **Tags:** xml.sequenceOffset=35

Table A.5: BswModuleEntry

Class	***DataPrototype*** (abstract)			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level. **Tags:** atp.Splitkey=swDataDefProps

Table A.6: DataPrototype

Class	**DataTypeMap**			
Note	This class represents the relationship between `ApplicationDataType` and its implementing `AbstractImplementationDataType`.			
Base	*ARObject*			
Aggregated by	DataTypeMappingSet.dataTypeMap			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationData Type	ApplicationDataType	0..1	ref	This is the corresponding `ApplicationDataType` **Tags:** atp.Splitkey=applicationDataType
implementation DataType	AbstractImplementation DataType	0..1	ref	This is the corresponding `AbstractImplementationDataType`. **Tags:** atp.Splitkey=implementationDataType

Table A.7: DataTypeMap

Class	**ImplementationDataType**			
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*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *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.

▽

△

Class	**ImplementationDataType**			
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 `ImplementationDataTypeElement` 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
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name
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.8: ImplementationDataType

Class	**ImplementationDataTypeElement**			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further `subElement`s or it is further defined via its `swDataDefProps`. There are several use cases within the system of `ImplementationDataType`s for such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ImplementationDataType.subElement, ImplementationDataTypeElement.sub Element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arrayImplPolicy	ArrayImplPolicyEnum	0..1	attr	This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing `ImplementationDataType` constitutes an array.
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this `ImplementationDataTypeElement` represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.

▽

△

Class	**ImplementationDataTypeElement**			
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing `ImplementationDataTypeElement` as optional. This means that, at runtime, the `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.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of `ImplementationDataTypeElement` 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
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement. **Tags:** atp.Splitkey=swDataDefProps

Table A.9: ImplementationDataTypeElement

Class	**InterpolationRoutine**			
Note	This represents an interpolation routine taken to evaluate the contents of a curve or map against a specific input value.			
Base	*ARObject*			
Aggregated by	InterpolationRoutineMapping.interpolationRoutine			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
interpolation Routine	BswModuleEntry	0..1	ref	This specifies a BswModuleEntry which implements the current interpolation method for the given record layout. **Tags:** atp.Splitkey=interpolationRoutine xml.sequenceOffset=30 This Attribute is only used by the AUTOSAR Classic Platform.
isDefault	Boolean	0..1	attr	This attribute specifies whether the enclosing InterpolationRoutine is considered the default in the context (defined by the System Template) of a given collection InterpolationRoutineMapping that owns the enclosing InterpolationRoutine. **Tags:** xml.sequenceOffset=20
shortLabel	Identifier	0..1	attr	This is the name of the interpolation method which is implemented by the referenced bswModuleEntry. It corresponds to swInterpolationMethod in SwDataDef Props. **Tags:** xml.sequenceOffset=10

Table A.10: InterpolationRoutine

Class	**InterpolationRoutineMapping**			
Note	This meta-class provides a mapping between one record layout and its matching interpolation routines. This allows to formally specify the semantics of the interpolation routines. The use case is such that the curves/Maps define an interpolation method. This mapping table specifies which interpolation routine implements methods for a particular record layout. Using this information, the implementer of a software-component can select the appropriate interpolation routine.			
Base	*ARObject*			
Aggregated by	InterpolationRoutineMappingSet.interpolationRoutineMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
interpolation Routine	InterpolationRoutine	*	aggr	This is one particular interpolation routine which is mapped to the record layout. **Tags:** atp.Splitkey=interpolationRoutine
swRecord Layout	SwRecordLayout	0..1	ref	This refers to the record layout which is mapped to interpolation routines. **Tags:** atp.Splitkey=swRecordLayout

Table A.11: InterpolationRoutineMapping

Class	**InterpolationRoutineMappingSet**			
Note	This meta-class specifies a set of interpolation routine mappings. **Tags:** atp.recommendedPackage=InterpolationRoutineMappingSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
interpolation Routine Mapping	InterpolationRoutine Mapping	*	aggr	This specifies one particular mapping of recordlayout and its matching interpolationRoutines. **Tags:** atp.Splitkey=interpolationRoutineMapping

Table A.12: InterpolationRoutineMappingSet

Class	**ParameterAccess**			
Note	The presence of a `ParameterAccess` implies that a `RunnableEntity` needs access to a `ParameterDataPrototype`.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, RunnableEntity.parameterAccess			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
accessed Parameter	AutosarParameterRef	0..1	aggr	Reference to the accessed calibration parameter. **Tags:** atp.Splitkey=accessedParameter
swDataDef Props	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis. **Tags:** atp.Splitkey=swDataDefProps

Table A.13: ParameterAccess

Class	**RunnableEntity**
Note	A `RunnableEntity` represents the smallest code-fragment that is provided by an `AtomicSwComponentType` and are executed under control of the RTE. `RunnableEntity`s are for instance set up to respond to data reception or operation invocation on a server.

▽

△

Class	**RunnableEntity**			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.runnable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of an argument to a RunnableEntity.
asynchronous ServerCall ResultPoint	AsynchronousServer CallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint, asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
canBeInvoked Concurrently	Boolean	0..1	attr	If the value of this attribute is set to "true" the enclosing `RunnableEntity` can be invoked concurrently (even for one instance of the corresponding `AtomicSwComponentType`). This implies that it is the responsibility of the implementation of the `RunnableEntity` to take care of this form of concurrency.
dataRead Access	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReadAccess, dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceive PointBy Argument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument, dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
dataReceive PointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointBy Value is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue, dataReceivePoint ByValue.shortName, dataReceivePointByValue.variation Point.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataSendPoint, dataSendPoint.shortName, dataSendPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWrite Access	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataWriteAccess, dataWriteAccess.short Name, dataWriteAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
external TriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint, externalTriggering Point.ident.shortName, externalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
internal TriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint, internalTriggering Point.shortName, internalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime
modeAccess Point	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=modeAccessPoint, modeAccess Point.ident.shortName, modeAccessPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch Point	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=modeSwitchPoint, modeSwitchPoint.short Name, modeSwitchPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameter Access	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a Parameter DataPrototype which may either be local or within a Port Prototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of Parameter Access (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=parameterAccess, parameterAccess.short Name, parameterAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=readLocalVariable, readLocalVariable.short Name, readLocalVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=serverCallPoint, serverCallPoint.shortName, serverCallPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
symbol	CIdentifier	0..1	attr	The symbol describing this `RunnableEntity`'s entry point. This is considered the API of the `RunnableEntity` and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The `WaitPoint` associated with the `RunnableEntity`. **Tags:** atp.Splitkey=waitPoint, waitPoint.shortName
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=writtenLocalVariable, writtenLocal Variable.shortName, writtenLocalVariable.variation Point.shortLabel vh.latestBindingTime=preCompileTime

Table A.14: RunnableEntity

Class	«atpVariation» **SwDataDefProps**
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, swAddrMethod, 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, invalid Value • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime
Base	*ARObject*
Aggregated by	*AutosarDataType*.swDataDefProps, CompositeNetworkRepresentation.networkRepresentation, Cpp ImplementationDataTypeElement.swDataDefProps, *DataPrototype*.swDataDefProps, DataPrototype TransformationProps.networkRepresentationProps, DiagnosticDataElement.swDataDefProps, Diagnostic EnvDataElementCondition.swDataDefProps, DiagnosticExtendedDataRecordElement.swDataDefProps, DiagnosticSovdPrimitiveContentElement.swDataDefProps, DltArgumentProps.networkRep, FlatInstance Descriptor.swDataDefProps, ImplementationDataTypeElement.swDataDefProps, InstantiationDataDef Props.swDataDefProps, ISignal.networkRepresentationProps, McDataInstance.resultingProperties, ParameterAccess.swDataDefProps, PerInstanceMemory.swDataDefProps, *ReceiverComSpec*.network Representation, SecurityEventContextDataElement.networkRepresentation, *SenderComSpec*.network Representation, SomeipDataPrototypeTransformationProps.networkRepresentation, SwPointerTarget Props.swDataDefProps, SwServiceArg.swDataDefProps, SwSystemconst.swDataDefProps, System Signal.physicalProps

Attribute	**Type**	**Mult.**	**Kind**	**Note**
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** atp.Splitkey=annotation 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:** atp.Splitkey=baseType xml.sequenceOffset=50

△

Class	«atpVariation» **SwDataDefProps**			
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** atp.Splitkey=compuMethod xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** atp.Splitkey=dataConstr xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** atp.Splitkey=implementationDataType xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** atp.Splitkey=invalidValue 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:** atp.Splitkey=swAddrMethod xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod. **Tags:** xml.sequenceOffset=33

▽

△

Class	«atpVariation» **SwDataDefProps**			
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** atp.Splitkey=swBitRepresentation xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** atp.Splitkey=swCalprmAxisSet xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** atp.Splitkey=swComparisonVariable xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** atp.Splitkey=swDataDependency 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:** atp.Splitkey=swHostVariable xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250

▽

△

Class	«atpVariation» **SwDataDefProps**			
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** atp.Splitkey=swPointerTargetProps xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** atp.Splitkey=swRecordLayout xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** atp.Splitkey=swRefreshTiming xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** atp.Splitkey=swTextProps xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** atp.Splitkey=unit xml.sequenceOffset=350

▽

△

Class	«atpVariation» **SwDataDefProps**			
valueAxisData Type	ApplicationPrimitive DataType	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:** atp.Splitkey=valueAxisDataType xml.sequenceOffset=355

Table A.15: SwDataDefProps

Class	**SwRecordLayout**			
Note	Defines how the data objects (variables, calibration parameters etc.) are to be stored in the ECU memory. As an example, this definition specifies the sequence of axis points in the ECU memory. Iterations through axis values are stored within the sub-elements swRecordLayoutGroup. **Tags:** atp.recommendedPackage=SwRecordLayouts			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swRecord LayoutGroup	SwRecordLayoutGroup	0..1	aggr	This is the top level record layout group. **Tags:** atp.Splitkey=swRecordLayoutGroup xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table A.16: SwRecordLayout