Document Title	Macro Encapsulation of Library Calls
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	808

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Change Description
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 4
2 Related documentation 5
 2.1 Input documents 5
 2.2 Related specification 5
3 Introduction 6
4 Motivation 7
5 Disclaimer 8
6 Use Cases 9
 6.1 Generate Encapsulation Macros 9
 6.2 Use Encapsulation Macros 11
7 Solution Proposal 12
 7.1 Definition of Terminology 12
 7.2 Architectural Components 12
  7.2.1 Encapsulation Macros Header File 12
 7.3 Functional Description 13
  7.3.1 Basic Concept Description 13
  7.3.2 Implementation of Macro Encapsulation Concept 26

1 Acronyms and abbreviations

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

2 Related documentation

2.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[4] AUTOSAR Methodology
AUTOSAR_TR_Methodology.pdf

[5] Requirements on Software Component Template
AUTOSAR_RS_SoftwareComponentTemplate.pdf

2.2 Related specification

[1] Specification of Fixed Point Interpolation Routines
AUTOSAR_SWS_IFXLibrary.pdf

[2] Specification of Floating Point Interpolation Routines
AUTOSAR_SWS_IFLLibrary.pdf

[3] Specification of Run Time Environment
AUTOSAR_SWS_RTE.pdf

- AUTOSAR confidential -

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 "MacroEncapsulationofInterpolationCalls " 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.

- AUTOSAR confidential -

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.

- AUTOSAR confidential -

6 Use Cases

6.1 Generate Encapsulation Macros

The document AUTOSAR_TR_Methodology R4.2 illustrates the general approach of generation of atomic software component header files (Figure 1). The proposed encapsulation macros shall be saved in an "Encapsulation Macros Header File" similar to an "Application Header File".

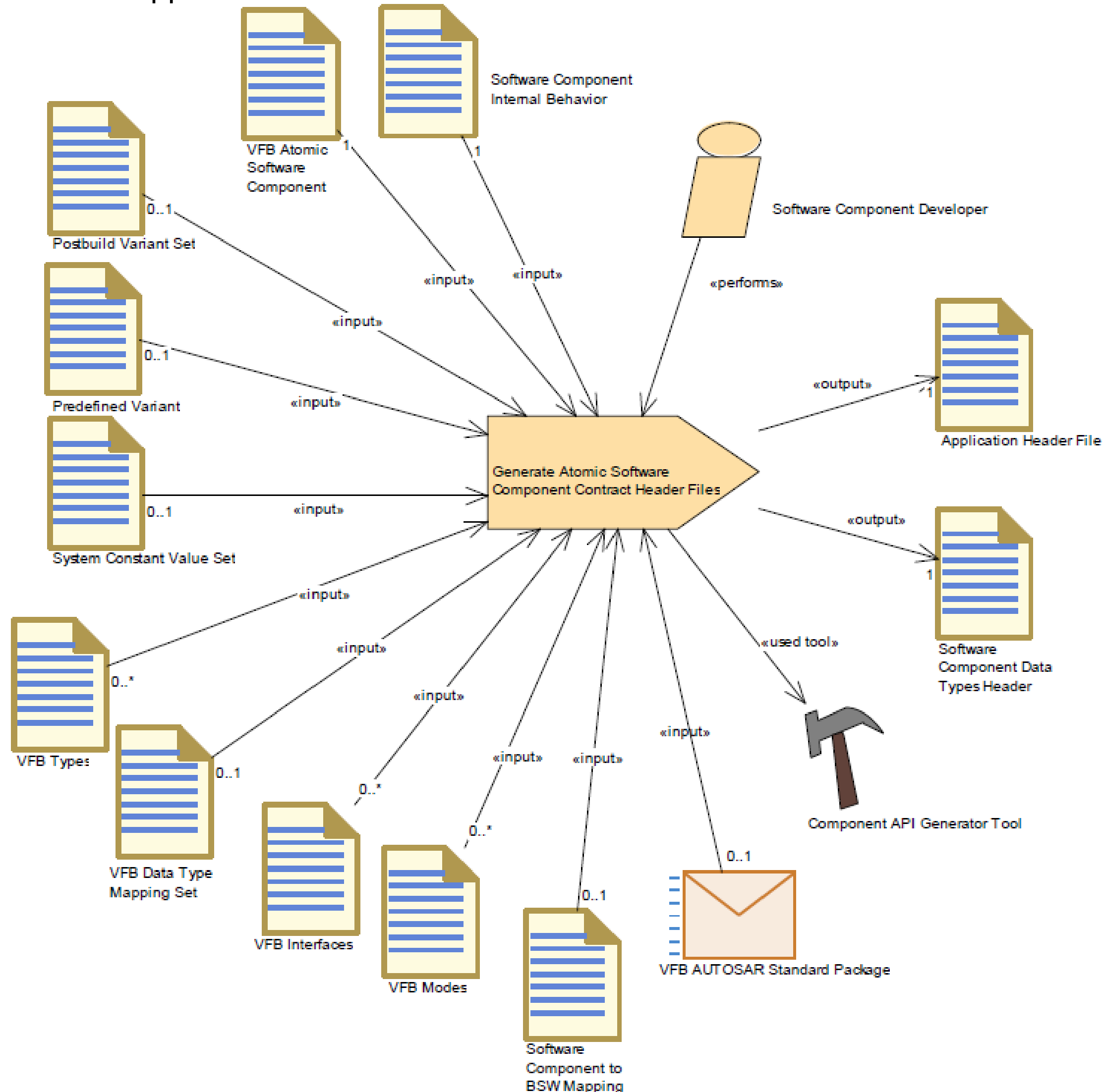

Figure 1: Generate Atomic Software Component Contract Header Files

Figure 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). 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 generated interfaces.
The proposed encapsulation macros shall be saved in an "Encapsulation Macros Header File" (see figure 2) similar to an "Application Header File

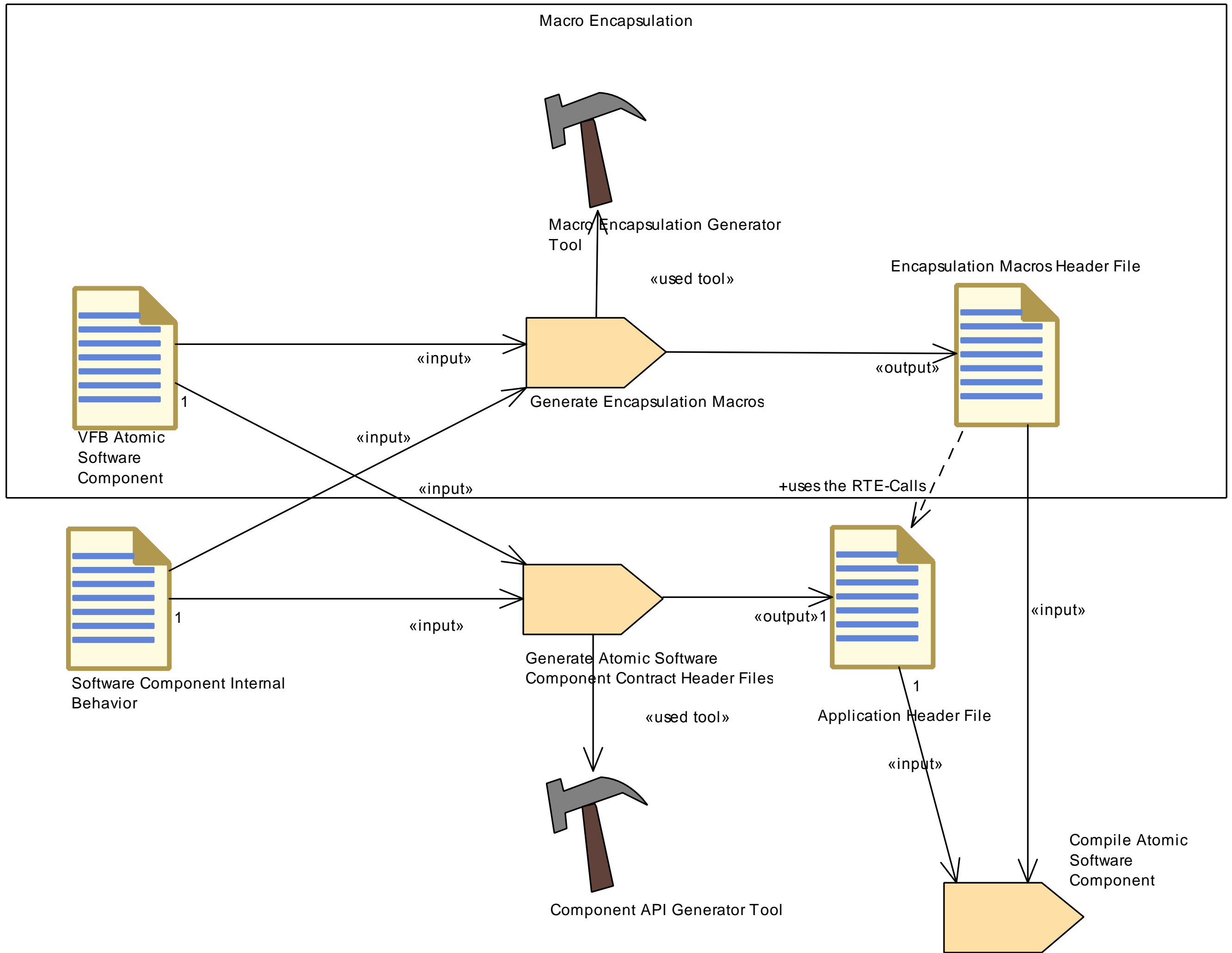

Figure 2: Generation Process of Encapsulation Macros Header File

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::MethodologyLibrary::Component::Work Products		
Brief Description	Header generated for an AtomicSoftwareComponentType from Macro Encapsulation Generator Tool after the RTE contract phase.		
Description	Header generated for an AtomicSoftwareComponentType from Macro Encapsulation Generator Tool after the RTE contract phase. 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***	***Mul.***	***Note***
AggregatedBy	Delivered Software Component	1	
ParameterOut	Generate Atomic Software Component Contract Header Files	1	Meth.bindingTime = CodeGenerationTime
ParameterIn	Compile Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime

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(X_input, 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(X_input, \
                                 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 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 preprocessed before calling the interpolation routine. In this case the interpolation routine is called with local variables which are not passed through RTE 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 make no problem. These parameters 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_IntlpoCur_U8_U8
  Ifx_IntlpoCur_U8_U16
  Ifx_IntlpoCur_U8_S8
  Ifx_IntlpoCur_U8_S16
  Ifx_IntlpoCur_U16_U8
  Ifx_IntlpoCur_U16_U16
  Ifx_IntlpoCur_U16_S8
  Ifx_IntlpoCur_U16_S16
  Ifx_IntlpoCur_S8_U8
  Ifx_IntlpoCur_S8_U16
  Ifx_IntlpoCur_S8_S8
  Ifx_IntlpoCur_S8_S16
  Ifx_IntlpoCur_S16_U8
  Ifx_IntlpoCur_S16_U16
  Ifx_IntlpoCur_S16_S8
  Ifx_IntlpoCur_S16_S16

- ***Parameters of the Interpolation Routine: {ImplTypeStruct}.{N}, …***
  Provision of the parameter of the interpolation routine. RTE 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 3 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.

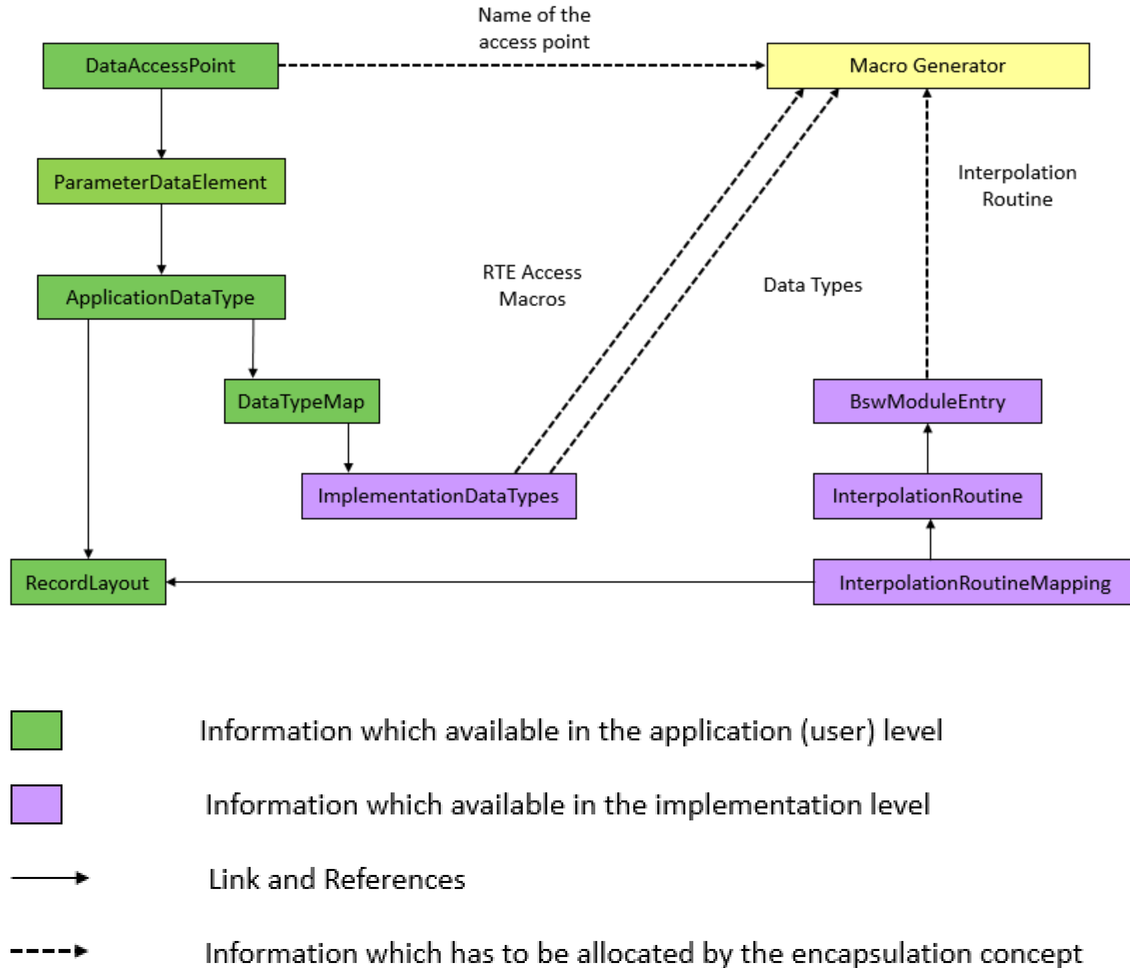

Figure 3: Overview of Workflow of Encapsulation Concept based on Meta Model

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 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 4 **and** Figure 5 describes the complete overview with different focus.

- AUTOSAR confidential -

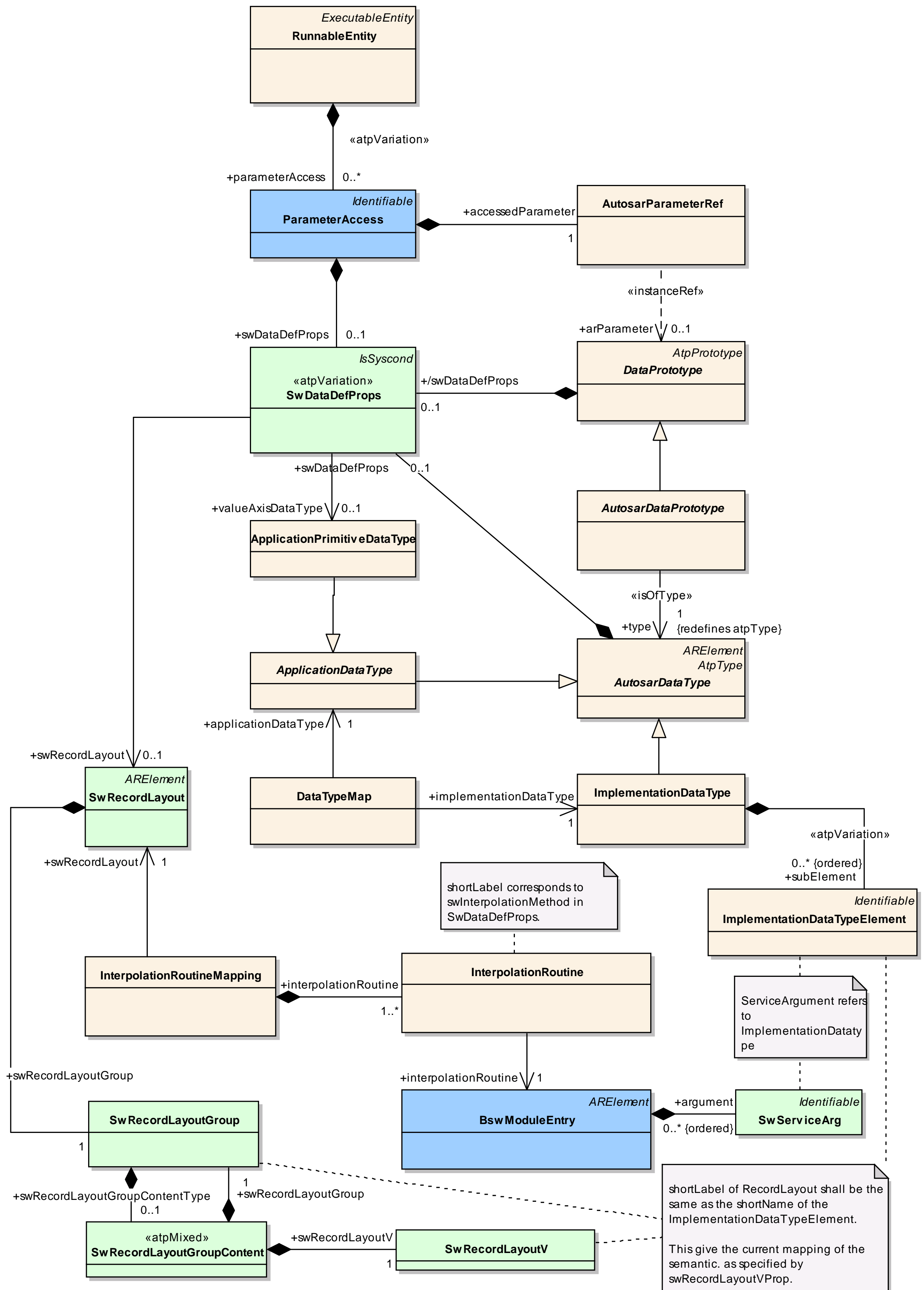

Figure 4: Complete MetaModel Overview to Find the Correct BswModuleEntry

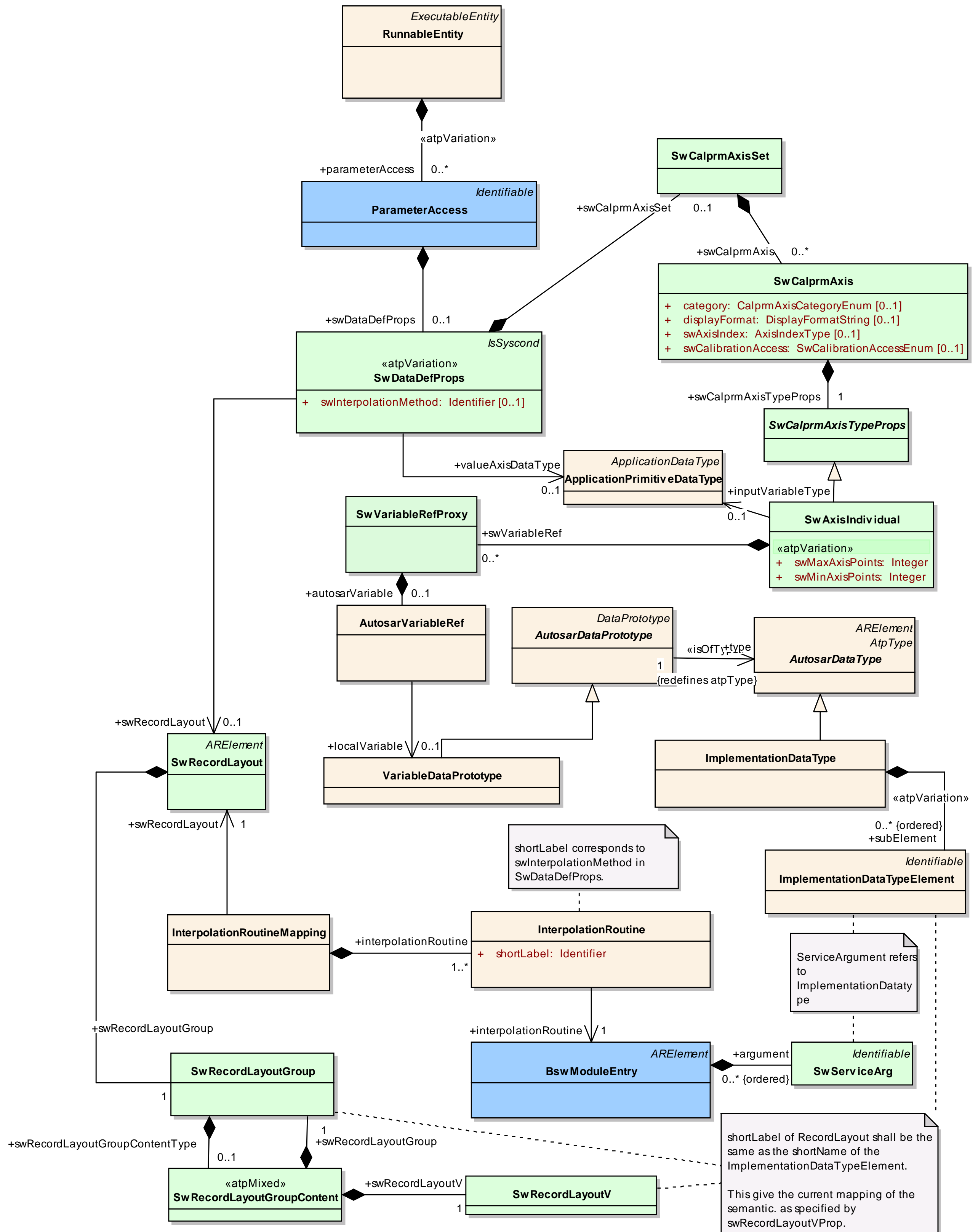

Figure 5: Complete MetaModel Overview to Find the Correct BSWModuleEntry with Focus SwCalprms

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

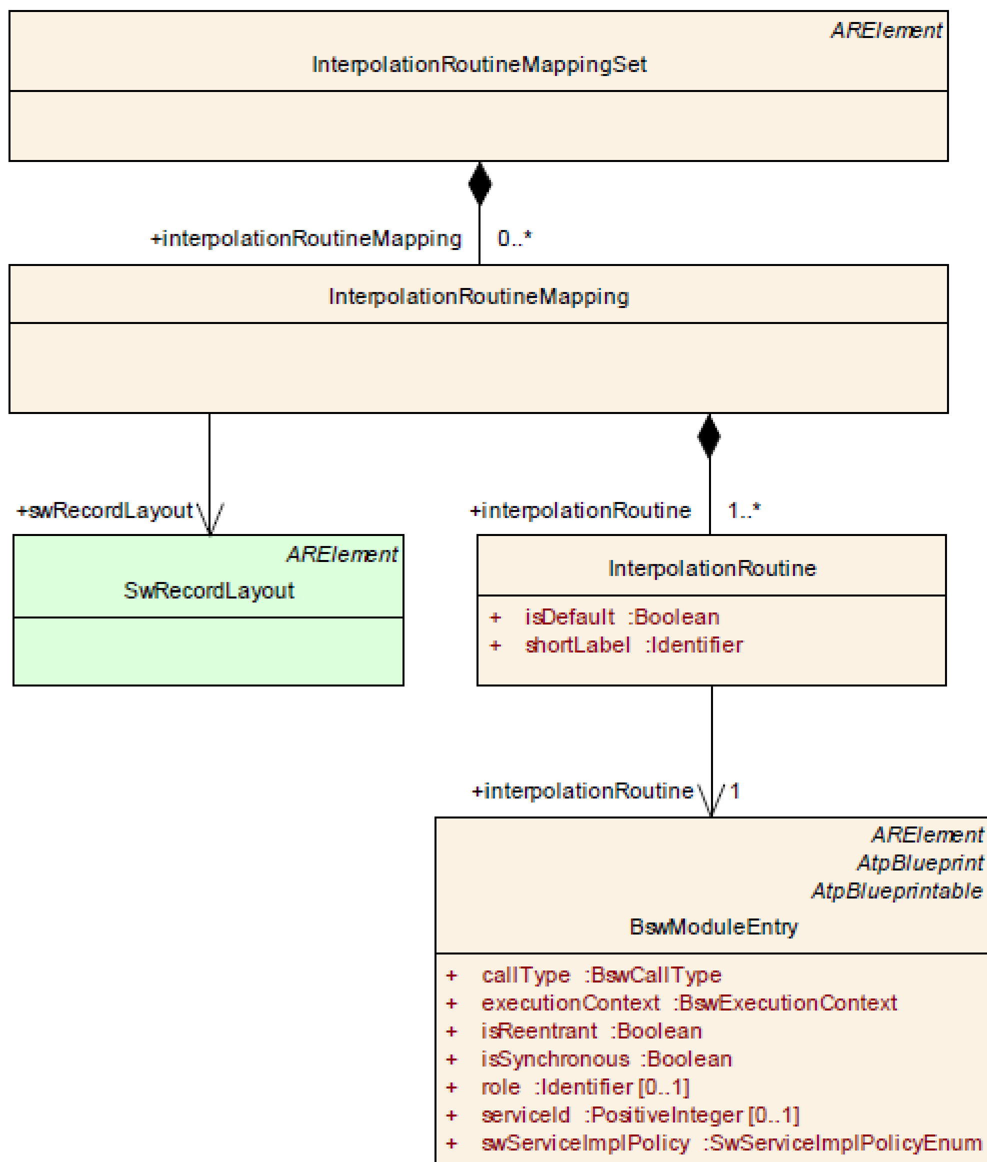

Figure 6: Mapping of Record Layouts and Interpolation Routines

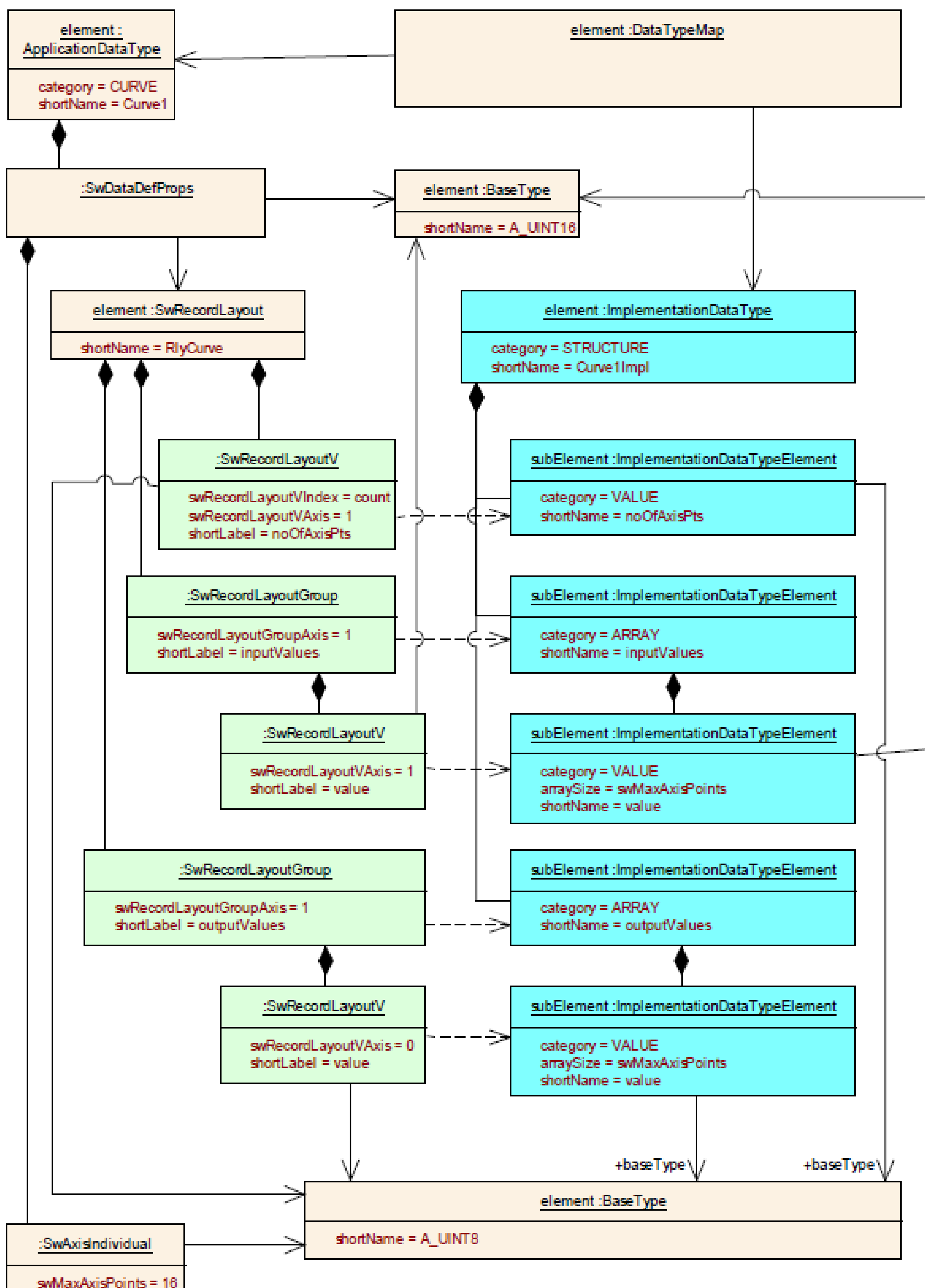

Figure 7: Curve implemented as two consecutive arrays

The structure and memory representation of a curve or map is described on data specification level via ***RecordLayout***. Figure 7 is taken from the AUTOSAR_TPS_SoftwareComponentTemplate, figure 5.48.

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 8 visualizes the dependency between **DataTypes** and **SwRecordLayouts** and is taken from AUTOSAR_TPS_SoftwareComponentTemplate figure 5.33.

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

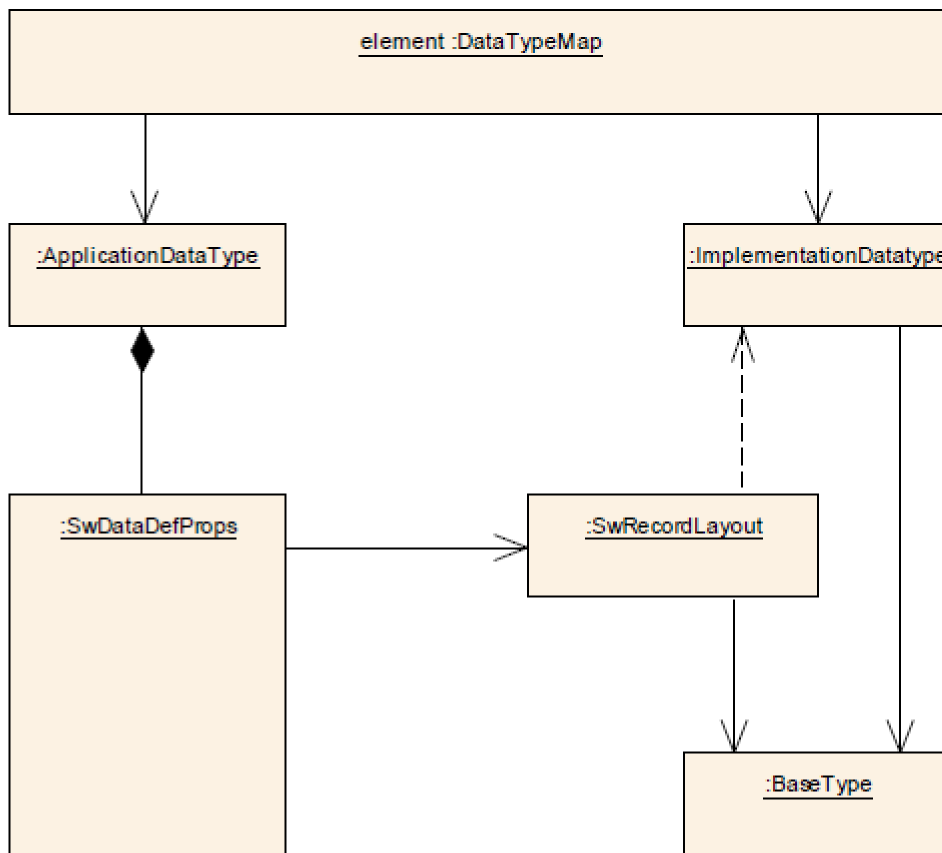

Figure 8: Dependency of DataTypes and SwRecordLayouts

- **{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 9 shows the runnable access to a calibration port. This picture is taken from AUTOSAR_TPS_SoftwareComponentTemplate, 7.29.

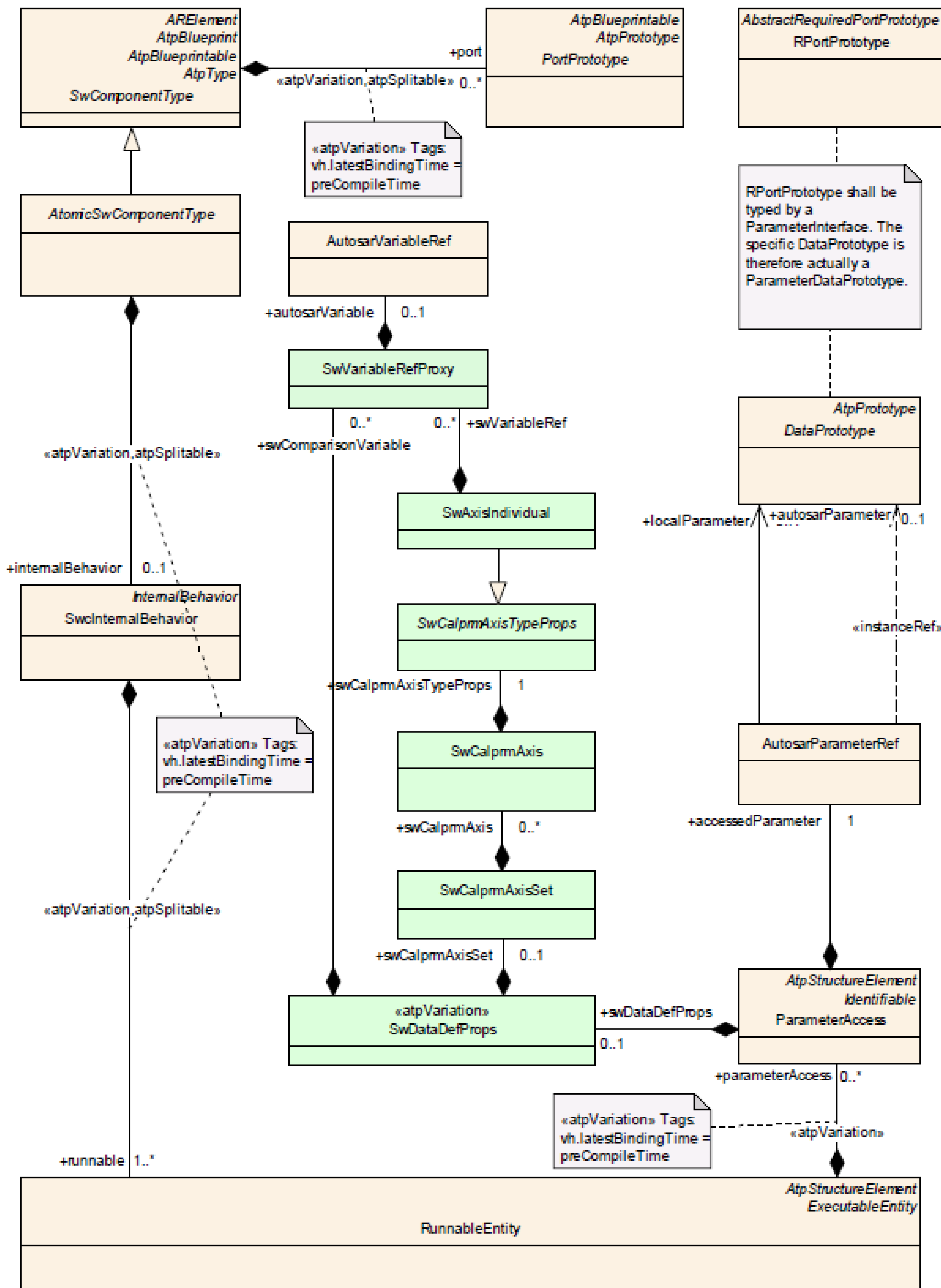

Figure 9: Runnable Access to a Calibration Port

7.3.2.2 Generation of the Name of the Interpolation Routine

The name of the interpolation routine is defined in the MetaModel as **BSWModule-Entry**. 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 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

- AUTOSAR confidential -

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

- AUTOSAR confidential -

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.