

<b>Document Title</b>	Unique Names for Documentation, Measurement and Calibration: Modeling and Naming Aspects including Automatic Generation
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	537
<b>Document Classification</b>	Auxiliary

<b>Document Version</b>	1.0.0
<b>Document Status</b>	Final
<b>Part of Release</b>	4.0
<b>Revision</b>	3

<b>Document Change History</b>			
<b>Date</b>	<b>Version</b>	<b>Changed by</b>	<b>Change Description</b>
03.11.2011	1.0.0	AUTOSAR Administration	Initial Release

## **Disclaimer**

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## **Advice for users**

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

**Content**

1	References .....	6
2	Scope .....	7
3	How to read this document.....	8
3.1	Conventions used .....	8
3.2	Numbering of Rules .....	8
3.3	Acronyms and Abbreviations .....	10
4	Requirements & Requirements Traceability .....	11
5	Methodological Background .....	15
5.1.1	SwCalibrationAccess .....	15
5.2	FlatMap for Measurement and Calibration Data .....	16
5.3	AliasNameSet for Display Names.....	21
5.4	FlatMap for Unique Names of SW Component Prototypes.....	22
5.5	ReferenceBases and Virtual Name Spaces.....	23
5.6	Instance cp- and pb-paths .....	25
6	Overall Modeling Rules .....	26
6.1	Name Part Length.....	27
7	DataPrototypes.....	28
7.1	Increase Readability of Display Names .....	30
7.2	VariableDataPrototypes.....	32
7.3	ParameterDataPrototypes .....	33
7.4	DataPrototypes of ApplicationArrayDataType .....	34
8	Name Space.....	35
9	SwSystemconsts .....	38
10	PortPrototypeBlueprints .....	39
11	ComponentHierarchy .....	42
12	SwComponentPrototypes .....	46
13	Unique SW-Signal Display Names.....	47
14	Appendix: Keywords Phys/Log (P/L-List) for Powertrain Domain .....	50

## List of Figures and Tables

Figure 1 FlatMap [4] .....	16
Figure 2 Example AUTOSAR System EngN .....	17
Figure 3 Example AUTOSAR Blueprints EngN .....	19
Figure 4 AliasNameAssignment [4] .....	21
Figure 5 Example with Multiple Instantiation – Coordinator Wheels .....	44
Figure 6 Example ExtrLi with Multiple Instantiation .....	48
Table 1 Keywords Phys/Log (P/L-List) for Powertrain Domain .....	50

## Known Limitations

- The issue of systematically solving naming conflicts in downstream artifacts is still an open topic (see in chapter 8 Name Space).

## 1 References

- [1] SW-C and System Modeling Guide  
AUTOSAR\_TR\_SWCModelingGuide.pdf
- [2] Table of Application Interfaces  
AUTOSAR\_MOD\_AITable.xls
- [3] XML Specification of Application Interfaces  
AUTOSAR\_MOD\_AISpecification.arxml
- [4] System Template  
AUTOSAR\_TPS\_SystemTemplate.pdf
- [5] Software Component Template  
AUTOSAR\_TPS\_SoftwareComponentTemplate.pdf
- [6] Standardization Template – *upcoming*  
AUTOSAR\_TPS\_StandardizationTemplate.pdf
- [7] Generic Structure Template  
AUTOSAR\_TPS\_GenericStructureTemplate.pdf
- [8] Specification of BSW Module Description Template  
AUTOSAR\_TPS\_BSWModuleDescriptionTemplate.pdf
- [9] Specification of RTE  
AUTOSAR\_SWS\_RTE.pdf
- [10] ASAM MCD-2 MC (ASAP2)  
<http://www.asam.net/index.php?>
- [11] Explanation of Application Interfaces of the Powertrain Domain  
AUTOSAR\_EXP\_AIPowertrain.pdf
- [12] AUTOSAR Predefined Names  
AUTOSAR\_TR\_PredefinedNames.pdf

## 2 Scope

This document is specific for the powertrain domain. For the powertrain domain efficient handling of calibration and measurement is of great importance.

This document gives some methodological background on calibration relevant issues of AUTOSAR independent of any application domain.

Examples are provided. Although they might resemble standardized application interfaces they are simply to be seen as examples<sup>1</sup>.

The main focus of this document, however, is to give a proposal how to automatically generate display names<sup>2</sup> for measurement, calibration and diagnostic tools (MCD). If some assumptions and specific modeling rules are fulfilled then automatic generation leads to adequate display names. Of course, manual definition of display names is still possible and in cases the generated name is not suitable even necessary.

This document does not describe a complete mapping to MCD-2 MC (A2L, [10]). Its current focus is on variable and parameter prototypes of sender receiver and parameter interfaces as well as system constants. Local measurables and calibration parameters as well as other interface types are not (yet) considered.

Documentation issues are not (yet) considered.

This document does **not** contain rules for modeling and naming of AUTOSAR elements in general. This is already covered by AUTOSAR\_TR\_SWCModelingGuide [1].

The XML code which is shown in the document is compliant to the AUTOSAR xsd of Release 4.0.

The algorithms described in this document were applied to the display names part of [11].

---

<sup>1</sup> E.g. the paths are not correct: normally it is /AUTOSAR/AISpecification/Units ([3]) etc. but in this document for better readability the paths were shortened to /AUTOSAR/Units a.s.o..

<sup>2</sup> Please note when using A2L [10]: the display name as used in this document is not identical to the DISPLAY\_IDENTIFIER for A2L but identical to the attribute Name itself.

## 3 How to read this document

### 3.1 Conventions used

In requirements, the following specific semantics are used (taken from Request for Comment RFC 2119 from the Internet Engineering Task Force IETF)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. Note that the requirement level of the document in which they are used modifies the force of these words.

**MUST:** This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

**MUST NOT:** This phrase, or the phrase „SHALL NOT“, means that the definition is an absolute prohibition of the specification.

**SHOULD:** This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

**SHOULD NOT:** This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

**MAY:** This word, or the adjective „OPTIONAL“, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, **MUST** be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, **MUST** be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

### 3.2 Numbering of Rules

All rules or recommendations are identified by an ID. See [6] for name pattern of such identifiers and [12] for the predefined names including the ones used in this document.

The ID starts with “TR\_MCM” for the Modeling Rules for Measurement and Calibration followed by four digits ([TR\_MCM\_xxxx]).

The ID starts with “TR\_MCG” for the Generation Rules for Measurement and Calibration followed by four digits ([TR\_MCG\_xxxx]).



The ID starts with “TR\_MCA” for the Assumptions made for being able to use specific generation rules ([TR\_MCA\_xxxx]). They are introduced for easier stating which assumptions are true for your project (line) and then to decide which rules (TR\_MCR) are applicable.

The ID starts with “TR\_MCR” for the Requirements for Measurement and Calibration followed by four digits ([TR\_MCR\_xxxx]).

Terms taken from the Meta model like e.g. *DataPrototype* are written in italic.

The following table gives an overview of renumbered or deleted rules:

New	Old	Comment
[TR_MCR_9<number>]	[MCR0<number>]	For avoiding same numbers for requirements and rules etc. [6]
[TR_MCM_7<number>]	[MCM0<number>]	For avoiding same numbers for modeling rules from generation rules etc. [6]
[TR_MCA_8<number>]	[MCA0<number>]	For avoiding same numbers for assumptions and rules etc. [6]
[TR_MCA_8789]	[MCA030b]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCG_0788]	[MCG005b]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCR_9781]	[MCR010a]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCR_9782]	[MCR010b]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCR_9783]	[MCR013a]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCR_9784]	[MCR013b]	Renumbered according [TPS_STDT_0042] in [6]
[TR_MCG_0787]	[MCG710]	Because same number as in [TR_MCA_8710]
changed	[TR_MCM_7030]	
changed	[TR_MCG_0035]	
changed	[TR_MCM_7022]	
changed	[TR_MCG_0770]	
--	[TR_MCG_0712]	Implicitly contained in other rules
--	[TR_MCG_0715]	Implicitly contained in other rules

For formulating the rules the used syntax follows the one in [6]. However, some additional placeholder were “invented” to be able to express the intended rules. might be subject to changes in future releases.

The additionally used placeholder are listed in the following:

componentDescriptor

```
componentHierarchy
data
dataInfo
element
port
portBlueprint
portBlueprintDescriptor
systemDescriptor
swSignal
```

The name of a concrete identifiable is presented as (identifiable).name, e.g. PortPrototype.name

### 3.3 Acronyms and Abbreviations

Abbreviation	Explanation
AI	Application interfaces
component	components are the SwComponentPrototypes of a CompositionSwComponentType
cp-path	SwComponentPrototype-Path, see chapter 5.3
dataElement or data element	Element of a port interface of type AutosarDataPrototype, dataElements are VariableDataPrototypes of a PortInterface
DataPrototype or data prototype	An AutosarDataPrototype like VariableDataPrototype or ParameterDataPrototype etc.
Descriptor	<X>Descriptor: Short form for “ShortName FlatInstanceDescriptor in a FlatMap of target element x of element type X”
Element	Element might be a prototype element like SwComponentPrototype, DataPrototype etc. or an ARElement
MCD	Measurement, Calibration and Diagnostic
name	Used as short form for ShortName if not mentioned otherwise
parameter	Element of a ParameterInterface of type ParameterDataPrototype, parameters are the ParameterDataPrototypes of a PortInterface
pb-path	PortPrototypeBlueprint-Path, see chapter 5.3
P/L-list	List with abbreviations for physical and logical types. See chapter 14
port	port prototype or port prototype blueprint – depending on context
SW	Software
SW-C	Software Component
virtual name space	In this document we use the term “ <b>virtual name space for {element} or {prototype}</b> ” to denote that the names of the corresponding elements are unique within the corresponding AR-Package, recursively. See chapter 5.5

## 4 Requirements & Requirements Traceability

In the following table you find a listing of the requirements together with a short description (“Requirement” column) and a short sketch whether and if yes by which rules the requirement is fulfilled in this document (“Satisfied by” column). Sometimes the reference of the chapter is given in which the rules and more details can be found.

*Note: please do not expect to understand the provided solution (“satisfied by”) at this point. It is only a sketch.*

Requirement	Satisfied by
[TR_MCR_9000]「 For display names a global name space within one ECU is sometimes required by Measurement and Calibration Tools or Calibration Data Exchange Formats like e.g. A2L. Therefore this shall be supported.」 ()	See chapter 5.2 A global name space is defined within a <i>FlatMap</i> .
[TR_MCR_9001]「 For one calibration team at OEM or Supplier site: The display name of a software signal shall be stable and identical in all systems to be calibrated by the team independent of the project, the project configuration or the underlying (productline) software architecture.」 ()	See chapter 6 to chapter 13.  For single instantiation the software component type and prototype name are not part of the display name if some assumptions are fulfilled, eg. that there is a virtual name space for all port prototypes: so the requirement is fulfilled.  However, with multiple instantiation this requirement is more difficult to fulfill: the names or at least the number of instances (= prototypes) of a software component type shall be taken into account to generate unique display names.  To fulfill the requirement that there are not only unique names within a single product but for all projects of a product family companies should maintain a display name database even in the cases the original name was specified manually.
[TR_MCR_9002]「 There shall be a default derivable display name.」 ()	See all [TR_MCG*] rules  The default display name is the automatically derived display name.
[TR_MCR_9003]「 Automatic generation	See chapter 6 to chapter 13.

of display names shall be possible.」 ()	
[TR_MCR_9004]「 Automatic generation/derivation of model element names from display names should be possible.」 ()	Automatic generation of model element names would be possible. However automatically re-derivation of the original model element names is only possible if the rules applied and the assumptions made are the same for all derived names.
[TR_MCR_9005]「 It should be possible (but the exception in practice - see requirements for automatic generation and default display names) to manually define a display name for a data prototype.」 ()	Manual definition of display names is not excluded. Manual overwriting of automatically derived names is allowed.
[TR_MCR_9006]「 A display name should be kept as short as possible. It should not exceed 31 characters. In the best case a display name would not exceed 16 characters.  In case of multiple instances the name length can be extended to ensure uniqueness.  The current restrictions of MCD tools shall be considered.」 ()	See chapter 6.1  In case the used MCD tool (version) has stronger length restrictions on the length the display names and the derived display names are too long they have to be manually overwritten.  However, if specific assumptions are fulfilled additional rules are provided to shorten the generated display name (e.g. generation rule [TR_MCG_0010]).  Additionally it is recommended to restrict the length of the short names from the beginning although the meta model allows much longer names ([TR_MCM_7020]).
[TR_MCR_9008]「 In case of multiple instantiation it shall be ensured that instances with the same semantic meaning have the same display names in projects with different numbers of instances.」 ()	This requirement can only be fulfilled if the instance names are identical for all projects. Unfortunately there is no possibility to specify whether several instances are allowed or not. So you might add the instance name in all cases or always omit it and manually overwrite generated display names for the few cases you used multiple instantiation (or vice versa).
[TR_MCR_9009]「 In case the MCD tools do not provide an easy-to-use mechanism for identification of maps, curves etc. the corresponding information should be part of the display name itself.」 ()	See chapter 7.3: add suffix _C, _M etc. to parameters.

<p>[TR_MCR_0781]「 If elements belong together it should be possible to sort them alphabetically.」 ()</p>	<p>See chapter 7.1, [TR_MCM_7070]: it is recommended to start the name with the mean or other semantical information and then to continue with the physical or logical meaning of the sw signal (see requirement [TR_MCR_0782]).</p>
<p>[TR_MCR_0782]「 For elements belonging together it should be possible to sort subgroups according to their physical or logical meaning.」 ()</p>	<p>See chapter 7.1: if a P/L-list is given the names can be improved by physical/logical meaning at one glance. Of course this can only be achieved if the original model element names follow some modeling rules like [TR_MCM_7065].</p>
<p>[TR_MCR_9011]「 The naming convention shall consider a number of about 20000 calibration relevant names per project. Thus readability is an important requirement.」 ()</p>	<p>Main overall goal is to shorten the names (see e.g. requirement [TR_MCR_9006]) for readability and then to further enhance the names for even more readability (see chapter 7.1)</p>
<p>[TR_MCR_0783]「 Modeling Rules or Recommendations for Measurement and Calibration shall not violate the general “shall”-rules for naming as defined in the Meta Model.」 ()</p>	<p>See Chapter 6. [TR_MCM_7005]</p>
<p>[TR_MCR_0784]「 Modeling Rules or Recommendations for Measurement and Calibration shall not violate the general “shall”-rules for naming as defined in AUTOSAR_TR_SWCModelingGuide [1].」 ()</p>	<p>See Chapter 6. [TR_MCM_7010]</p>
<p>[TR_MCR_9014]「 The display names should be useable also for legacy systems to implement global variables.」 ()</p>	<p>Display names of variable data prototypes can be used as names for global variables because C-conformity of display names is ensured (see [TR_MCG_0001]).</p>
<p>[TR_MCR_9015]「 Different data should have different display names.</p> <p>A2L e.g. does not explicitly forbid to have the same display identifier for different data. However, to use this mechanism can lead to confusion and seems not to have any advantages」 ().</p>	<p>See chapter 6 to chapter 13.</p> <p>Fulfilled by [TR_MCG_0005] and [TR_MCG_0788] because complete path is considered. Shortening of the name is only possible if specific assumptions can be made.</p> <p>Of course it is recommended to make a separate check besides generation to ensure that the same name is not gener-</p>

	<p>ated twice.</p> <p>For the display name identifier in A2L the alias name is used.</p>
<p>[TR_MCR_9016]「 One and the same software signal should only have exactly one display name in one calibration project.」 ()</p>	<p>This document does not describe for which ports a display name is specified but describes the general approach. So uniqueness has to be ensured by selecting only the appropriate ports, e.g. only p-ports. In some cases there might be no p-ports so there is no general rule for this.</p>

## 5 Methodological Background

The following chapters give background information on aspects of the AUTOSAR metal model and methodology. They can be skipped except for chapter 5.5 and 5.6 that introduce some new aspects that are important for understanding the following topics of the document.

### 5.1.1 SwCalibrationAccess

Display Names have to be provided for all instances of *DataPrototypes* for which the attribute *SwCalibrationAccess* is set via its *SwDataDefProps* to “readOnly” or “read-Write”.

The information *SwCalibrationAccess* is mandatory for *ApplicationDataTypes* but can be overwritten. For example the *SwCalibrationAccess* can be overwritten in *DataPrototypes* or *FlatInstanceDescriptor.swDataDefProps* a.s.o. (see [constr\_1015] in [5], [rte\_sws\_7196] in [9]) .

In the current standardization of application interfaces ([3]) the information *SwCalibrationAccess* of *ApplicationDataTypes* is set to *ReadOnly*.

Example:

```
...
<VARIABLE-DATA-PROTOTYPE>
  <SHORT-NAME>EngN</SHORT-NAME>
  <TYPE-TREF
    DEST="APPLICATION-DATA-TYPE">
      /OEM1/ApplicationDataTypes/N1</TYPE-TREF>
  <SW-DATA-DEF-PROPS>...
    <SW-CALIBRATION-ACCESS>NOT-ACCESSIBLE</SW-CALIBRATION-ACCESS>
  ...</SW-DATA-DEF-PROPS>
</VARIABLE-DATA-PROTOTYPE>
```

...  
overwrites

```
...
<APPLICATION-PRIMITIVE-DATA-TYPE>
  <SHORT-NAME>N1</SHORT-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL> ...
        <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
      ...</SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS></SW-DATA-DEF-PROPS>
  ...
</APPLICATION-PRIMITIVE-DATA-TYPE>
```



## 5.2 FlatMap for Measurement and Calibration Data

Whereas *SystemSignals* serve for inter-ECU-communication, software signals are implicitly used within ECU-internal communication. There is no model element called software signal because they are represented by instance references (*InstanceRef*) to a *DataPrototype*. However, for calibration engineers there is the need to have unique names for software signals similar as there is the need to have unique names for inter-ECU-communication. These unique names for software signals are identical to the display names that are discussed in this document.

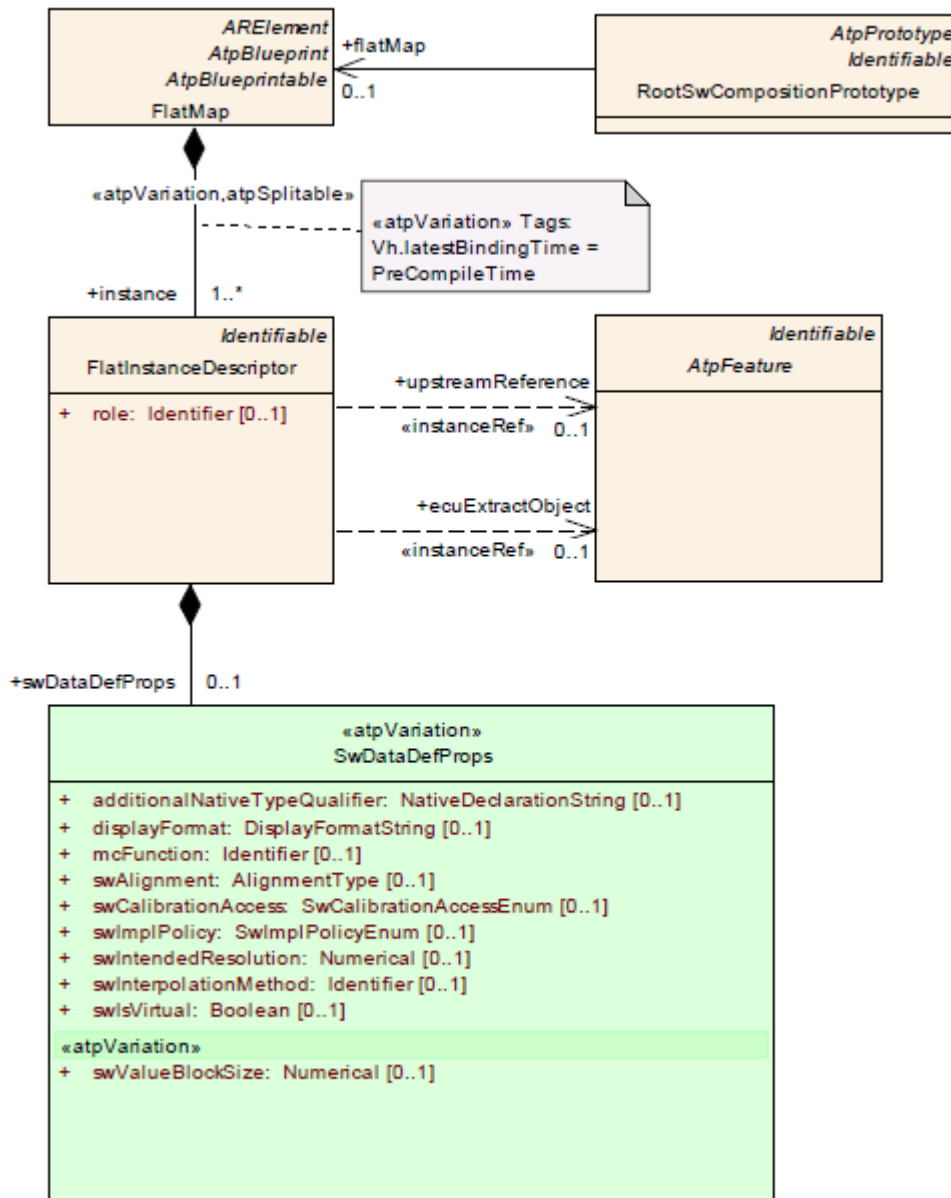


Figure 1 FlatMap [4]

The AUTOSAR methodology provides means to specify unique names e.g. to be used to generate A2L-files out of it [10]. This is done via so-called *FlatMaps* (see Figure 1). The *FlatMap* consists of several *FlatInstanceDescriptors*. In our context a



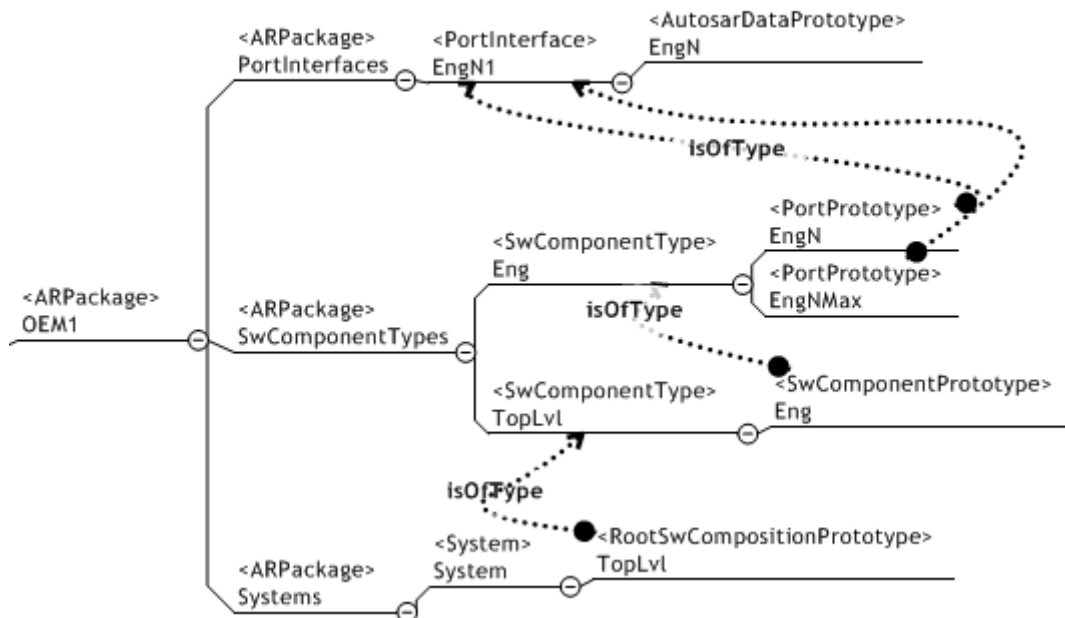
*FlatInstanceDescriptor* represents exactly one *VariableDataPrototype* or a *ParameterDataPrototype* and attaches a unique name (*shortName*) to it. This *shortName* can later be used as the display name of the software signal. In [4] the following mapping is recommended to A2L:

*FlatInstanceDescriptor.shortName* ->  
MEASUREMENT Name for *VariableDataPrototypes*  
CHARACTERISTIC Name for *ParameterDataPrototypes*

In [9], chapter 4.2.8, it is described how the RTE deals with Measurement and Calibration.

An instance of a *FlatMap* containing the definition of display names for a set of software signals is an XML-artifact which can be represented by a single file. Depending on its scope, it is attached (via reference) to the top-level-composition of a system description or to the top-level-composition (*RootSwCompositionPrototype*) of an ECU-extract.

The mapping table (*FlatMap*) can be maintained manually or the display names may be automatically generated. Of course, also a mix of both approaches is possible. This mix is supported by `<<atpSplitable>>`.



**Figure 2 Example AUTOSAR System EngN**

Example:

Figure 2 shows an example of an AUTOSAR system. In the following a *FlatMap* for the software signal with the display name “Eng\_n” is described in arxml:

```
<AR-PACKAGE>
  <SHORT-NAME>OEM1</SHORT-NAME>
  <AR-PACKAGES><AR-PACKAGE>
    <SHORT-NAME>FlatMaps</SHORT-NAME><ELEMENTS>
      <FLAT-MAP>
        <SHORT-NAME>OEMMap</SHORT-NAME>
        <INSTANCES>
          <FLAT-INSTANCE-DESCRIPTOR>
            <SHORT-NAME>Eng_n</SHORT-NAME>
            <UPSTREAM-REFERENCE-IREF>
              <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
                /OEM1/Systems/System/TopLvl
              </CONTEXT-ELEMENT-REF>
              <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">
                /OEM1/SwComponentTypes/TopLvl/Eng
              </CONTEXT-ELEMENT-REF>
              <CONTEXT-ELEMENT-REF DEST="PORT-PROTOTYPE">
                /OEM1/SwComponentTypes/Eng/EngN
              </CONTEXT-ELEMENT-REF>
              <TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /OEM1/PortInterfaces/EngN1/EngN
              </TARGET-REF>
            </UPSTREAM-REFERENCE-IREF>
          </FLAT-INSTANCE-DESCRIPTOR>
        </INSTANCES>
      </FLAT-MAP></ELEMENTS>
    </ELEMENTS>
  </AR-PACKAGE>...
```

For array types an index has to be added if only a single element out of the array is intended to be referenced. Example (no figure) for the first array element (INDEX : "0") of an array:

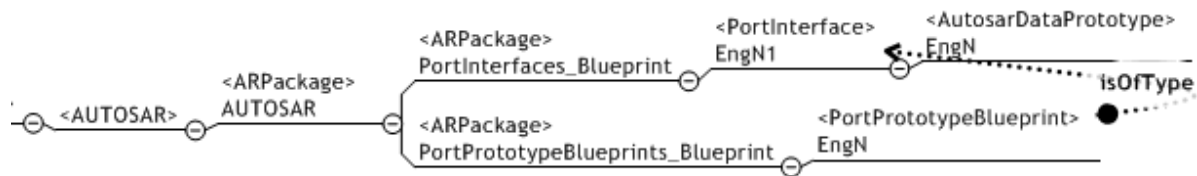
```
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>Esc_vWhlInd_0</SHORT-NAME>
  <UPSTREAM-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
      /OEM1/Systems/System/TopLvl
    </CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">
      /OEM1/SwComponentTypes/TopLvl/Pt
    </CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="PORT-PROTOTYPE">
      /OEM1/SwComponentTypes/Pt/EscVWhlInd
    </CONTEXT-ELEMENT-REF>
    <CONTEXT-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
      /OEM1/PortInterfaces/WhlSpdCircuml1/WhlSpdCircuml
    </CONTEXT-ELEMENT-REF>
    <TARGET-REF DEST="APPLICATION-ARRAY-ELEMENT" INDEX="0">
      /OEM1/ApplicationDataTypes/WhlSpdCircumlPerWhl1
    </TARGET-REF>
  </UPSTREAM-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
```

When only standardizing *PortPrototypeBlueprints* and not the software components themselves the display names can only be standardized by using the reference to the

*PortPrototypeBlueprint*. In this case the display names are themselves blueprints and are used as basis for deriving display names of port prototypes that were derived from the port prototype blueprint.

For multiple instantiated components the display names cannot be completely standardized because in general neither the number of instances nor the instance names are known in advance but only on ECU level.

Due to [constr\_2528] in [6] an element referencing blueprints has to be a blueprint itself or a *BlueprintMap* a *FlatMap* defining display names for blueprints is also a blueprint (i.e. it must be element of an *ARPackage* of Category *Blueprint*).



**Figure 3 Example AUTOSAR Blueprints EngN**

Example, see Figure 3:

Assume the port prototype blueprint “EngN” refers to port interface “EngN1”. “EngN1” might be a blueprint itself. For a blueprint the *NamePattern* has to be added. This is done for the complete *FlatMap* but also for its single *FlatInstanceDescriptors*. In this example we know that there is no multiple instantiation for this blueprint. Therefore the *NamePattern* is equal to the *ShortName* of the descriptor (*{blueprintName}*).

Additionally a *BlueprintCondition* is introduced to denote that e.g. not all standardized application interfaces are present in every system. Thus, it is allowed to derive a *FlatMap* that contains less elements than the blueprint *FlatMap* plus additional ones.

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
  <AR-PACKAGES><AR-PACKAGE>
    <SHORT-NAME>FlatMaps_Blueprint</SHORT-NAME>
    <CATEGORY>BLUEPRINT</CATEGORY>
    <ELEMENTS>
      <FLAT-MAP>
        <SHORT-NAME NAME-PATTERN="{anyName}">ARMap</SHORT-NAME>
        <INSTANCES>
          <FLAT-INSTANCE-DESCRIPTOR>
            <SHORT-NAME NAME-PATTERN="{blueprintName}">
              AR_Eng_n
            </SHORT-NAME>
            <LONG-NAME><L-4 L="EN">Actual Engine Speed</L-4></LONG-NAME>
            <UPSTREAM-REFERENCE-IREF>
              <CONTEXT-ELEMENT-REF DEST="PORT-PROTOTYPE-BLUEPRINT">
                /AUTOSAR/PortPrototypeBlueprints_Blueprint/EngN
              </CONTEXT-ELEMENT-REF>
            </UPSTREAM-REFERENCE-IREF>
          </FLAT-INSTANCE-DESCRIPTOR>
        </INSTANCES>
      </FLAT-MAP>
    </ELEMENTS>
  </AR-PACKAGE>
</AR-PACKAGES>
</AR-PACKAGE>
```

```
</CONTEXT-ELEMENT-REF>
<TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">
  /AUTOSAR/PortInterfaces_Blueprint/EngN1/EngN
</TARGET-REF>
</UPSTREAM-REFERENCE-IREF>
<VARIATION-POINT>
  <SHORT-LABEL NAME-PATTERN="{blueprintName}">VP1</SHORT-LABEL>
  <BLUEPRINT-CONDITION>
    <P><L-1 L="EN">used only if port prototypes are derived from
      this blueprint</L-1></P>
    <P><L-1 L="EN">The condition swSyscond has to be implemented
      in the derived element (Upcoming)</L-1></P>
  </BLUEPRINT-CONDITION>
  <SW-SYSCOND>undefined</SW-SYSCOND></VARIATION-POINT>
</FLAT-INSTANCE-DESCRIPTOR>
</INSTANCES>
</FLAT-MAP>
</ELEMENTS>
</AR-PACKAGE>...
```

The FlatMap presented earlier might so have been derived from this blueprint *Flat-Map*, formally described as:

```
<AR-PACKAGE>
  <SHORT-NAME>BlueprintMappingSets</SHORT-NAME>
  <ELEMENTS>
    <BLUEPRINT-MAPPING-SET>
      <SHORT-NAME>OEM1BlueprintMapSet</SHORT-NAME>
      <BLUEPRINT-MAPS>
        <BLUEPRINT-MAPPING>
          <BLUEPRINT-REF DEST="FLATMAP">
            /AUTOSAR/Flatmaps_Blueprint/ARMap
          </BLUEPRINT-REF>
          <DERIVED-OBJECT-REF DEST="FLAT-MAP">
            /OEM1/Flatmaps/OEMMap
          </DERIVED-OBJECT-REF>
        </BLUEPRINT-MAPPING>
        <BLUEPRINT-MAPPING>
          <BLUEPRINT-REF
            DEST="PORT-PROTOTYPE-BLUEPRINT">
            /AUTOSAR/PortPrototypeBlueprints_Blueprint/EngN
          </BLUEPRINT-REF>
          <DERIVED-OBJECT-REF DEST="P-PORT-PROTOTYPE">
            /OEM1/SwComponentTypes/Eng/EngN
          </DERIVED-OBJECT-REF>
        </BLUEPRINT-MAPPING>
        <BLUEPRINT-MAPPING>
          <BLUEPRINT-REF
            DEST="SENDER-RECEIVER-INTERFACE">
            /AUTOSAR/PortInterfaces_Blueprint/EngN1
          </BLUEPRINT-REF>
          <DERIVED-OBJECT-REF
            DEST="SENDER-RECEIVER-INTERFACE">
            /OEM1/PortInterfaces/EngN1
          </DERIVED-OBJECT-REF>
        </BLUEPRINT-MAPPING>
      </BLUEPRINT-MAPPING-SET>
```

```
</ELEMENTS>
</AR-PACKAGE>
```

### 5.3 AliasNameSet for Display Names

An *AliasNameAssignment* (see Figure 4) can be used to associate an alternative name to a flat instance descriptor or an *Identifiable* [4].

In [4] the following mapping is recommended to A2L [10]:

*AliasNameAssignment.shortLabel* ->  
MEASUREMENT -> DISPLAY\_IDENTIFIER for *VariableDataPrototypes*  
CHARACTERISTIC -> DISPLAY\_IDENTIFIER for *ParameterDataPrototypes*  
but  
SYSTEM\_CONSTANT -> Name for *SwSystemconsts*

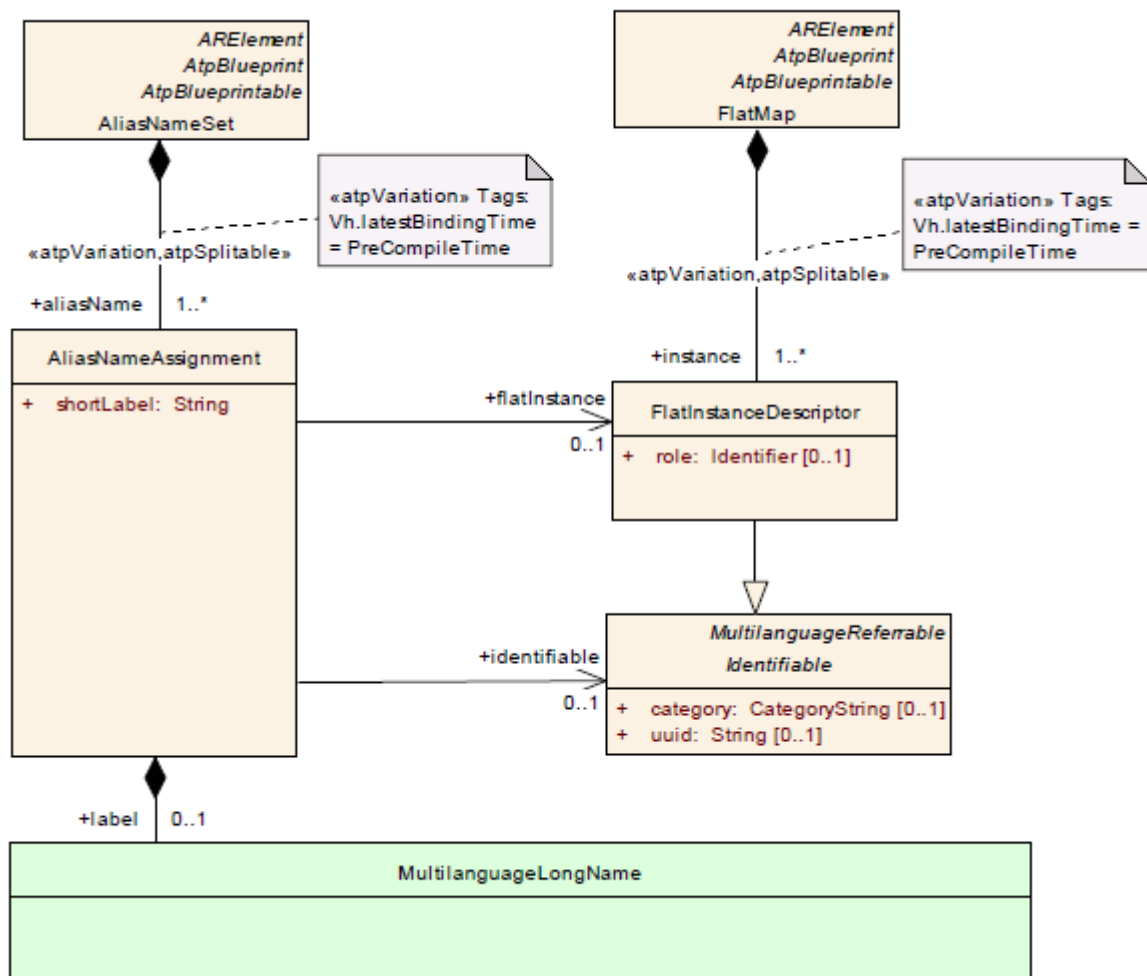


Figure 4 AliasNameAssignment [4]

Example for system constants:

```
<ALIAS-NAME-ASSIGNMENT>
  <SHORT-LABEL>ENGNRCYL_SC</SHORT-LABEL>
  <IDENTIFIABLE-REF DEST="SW-SYSTEMCONST">
    /OEM1/SwSystemconsts/EngNrCyl
  </IDENTIFIABLE-REF>
</ALIAS-NAME-ASSIGNMENT>
```

Since *FlatMaps* do not allow to reference *SwSystemconsts* the alias mechanism is used to specify unique display names for *SwSystemconsts*.

Aliases or `DISPLAY_IDENTIFIER` are not supported for *SwSystemconsts*, neither in AUTOSAR nor in ASAM MCD-2MC [10].

Additionally this mechanism is used if no unique display names can be generated but only a part of a display name that is to be extended for uniqueness. This might be the case for *PortPrototypeBlueprints*.

Example for blueprinted alias of an already existing flat instance descriptor that was generated automatically e.g. and now is intended to be “overwritten”:

```
<ALIAS-NAME-ASSIGNMENT>
  <SHORT-LABEL NAME-PATTERN="{blueprintName}">Esc_vWhlIndFrntLe
</SHORT-LABEL>
  <FLAT-INSTANCE-REF DEST="FLAT-INSTANCE-DESCRIPTOR">
    /AUTOSAR/Flatmaps/ARMap/Esc_vWhlInd_0
  </FLAT-INSTANCE-REF>...
</ALIAS-NAME-ASSIGNMENT>
```

This document does not contain rules for display names of *Units*. Nevertheless here some hints w.r.t. units. The units are standardized in [2] or [3], resp. For *Units* the meta model provides an explicit *DisplayName* Tag [5]. We have ([4] and [5])

*AliasNameAssignment.shortLabel* ->  
UNIT -> Name for Units  
*Unit.DisplayName* ->  
UNIT -> Display for Units

Note that the usage of the *AliasNameSet* is not defined within AUTOSAR. It is intended to be used by the A2L - generator which is not standardized by AUTOSAR. Nevertheless [4] gives some recommendations how to make the mapping.

## 5.4 FlatMap for Unique Names of SW Component Prototypes

Within the ECU extract of a system the *FlatMap* entries are needed to specify unique names for component prototypes [4].

Example:

The example specifies a unique name for the component prototype “Eng”. Please note that the example from Figure 2 was extended by an additional component hierarchy “Pt” to make example more real.

```
<FLAT-MAP>
  <SHORT-NAME>ARMap</SHORT-NAME>
  <INSTANCES>
    <FLAT-INSTANCE-DESCRIPTOR>
      <SHORT-NAME>Eng</SHORT-NAME>
      <UPSTREAM-REFERENCE-IREF>
        <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
          /OEM1/Systems/System/TopLvl
        </CONTEXT-ELEMENT-REF>
        <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">
          /OEM1/SwComponentTypes/TopLvl/Pt
        </CONTEXT-ELEMENT-REF>
        <TARGET-REF DEST="SW-COMPONENT-PROTOTYPE">
          /OEM1/SwComponentTypes/Pt/Eng
        </TARGET-REF>
      </UPSTREAM-REFERENCE-IREF>
    </FLAT-INSTANCE-DESCRIPTOR>
  </INSTANCES>
</FLAT-MAP>
```

## 5.5 ReferenceBases and Virtual Name Spaces

AUTOSAR allows to define so called reference bases for the usage of relative references. “Reference bases are identified by the *shortLabel* which must be unique in the current package [7].”

Example:

```
<REFERENCE-BASE>
  <SHORT-LABEL>AR</SHORT-LABEL>
  <IS-DEFAULT>>false</IS-DEFAULT>
  <IS-GLOBAL>>false</IS-GLOBAL>
  <BASE-IS-THIS-PACKAGE>>false</BASE-IS-THIS-PACKAGE>
  <PACKAGE-REF DEST="AR-PACKAGE">/AUTOSAR</PACKAGE-REF>
</REFERENCE-BASE>
```

Usage e.g. in

```
<TARGET-REF BASE="AR"
  DEST="SW-COMPONENT-PROTOTYPE">
  SwComponentTypes/Pt/Eng
</TARGET-REF>
```

In a way a reference base defines an abbreviation for a specific path. However, their usage should be restricted to define relative references and not to define some kind of alias for path.

In this document we use the term “virtual name space for {element}” to denote that the name of all elements of kind {element} are unique within a specific *ARPackage*.

This means that for all *ARPackages* being subpackages of this package as well as other elements of the meta model like *SwComponentTypes* that define their own name spaces according to the meta model this property of being able to define its own name space is ignored.

It also should be taken into account that there are restrictions to the AUTOSAR meta model specified for the RTE [9], i.e. the meta model allows to define more name spaces than the actual implementations of the RTE would support. E.g. [rte\_sws\_7190] requires unique names for *SwComponentTypes* regardless of the *ARPackage* hierarchy.

AUTOSAR allows to define so called global elements [7]. Global elements can be referenced by using this reference base. It means that the *ShortNames* of these elements are unique within the current package (if the base is the current package) or assumed to be unique in the referenced *ARPackage*. However, their usage is restricted and shall not be misused for defining virtual name spaces.

Example (ignoring [constr\_2538]) – do not realize like this!:

```
<REFERENCE-BASE>
  <SHORT-LABEL>globals</SHORT-LABEL>
  <IS-DEFAULT>false</IS-DEFAULT>
  <IS-GLOBAL>true</IS-GLOBAL>
  <BASE-IS-THIS-PACKAGE>true</BASE-IS-THIS-PACKAGE>
  <GLOBAL-ELEMENTS>
    <GLOBAL-ELEMENT>DATA-PROTOTYPE</GLOBAL-ELEMENT>
    <GLOBAL-ELEMENT>PORT-PROTOTYPE</GLOBAL-ELEMENT>
    <GLOBAL-ELEMENT>SW-COMPONENT-PROTOTYPE</GLOBAL-ELEMENT>
  </GLOBAL-ELEMENTS>
</REFERENCE-BASE>
```

Usage e.g. in

```
<FLAT-MAP>
  <SHORT-NAME>OEMMap</SHORT-NAME>
  <INSTANCES>
    <FLAT-INSTANCE-DESCRIPTOR>
      <SHORT-NAME>Eng_n</SHORT-NAME>
      <UPSTREAM-REFERENCE-IREF>
        <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
          /OEM1/Systems/System/TopLvl
        </CONTEXT-ELEMENT-REF>
        <CONTEXT-ELEMENT-REF BASE="globals"
          DEST="SW-COMPONENT-PROTOTYPE">
          Eng
        </CONTEXT-ELEMENT-REF>
        <CONTEXT-ELEMENT-REF BASE="globals"
          DEST="PORT-PROTOTYPE">
          EngN
        </CONTEXT-ELEMENT-REF>
        <TARGET-REF BASE="globals"
          DEST="VARIABLE-DATA-PROTOTYPE">
          EngN
        </TARGET-REF>
      </UPSTREAM-REFERENCE-IREF>
    </FLAT-INSTANCE-DESCRIPTOR>
  </INSTANCES>
</FLAT-MAP>
```



```

        </UPSTREAM-REFERENCE-IREF>
    </FLAT-INSTANCE-DESCRIPTOR>
</INSTANCES>
</FLAT-MAP>

```

[constr\_2538] in [7] restricts the usage of global references to certain elements like chapter etc.

## 5.6 Instance cp- and pb-paths

In this chapter new terms are introduced that allow a shortened description of the flat instance descriptors. In the context of calibration and measurement the data prototypes are of relevance.

**Definition.** The instance cp-path (*SwComponentPrototype-Path*) is the instance path to the software component prototype + one of its ports prototypes + one of the data prototypes (recursively) of the port interface of the port prototype:

{instance path to sw component prototype}/{port prototype}/{data prototype}<sub>1..n</sub>  
e.g.

```
/OEM1/Systems/System/TopLvl/Pt/Eng/EngN/EngN
```

i.e. in Backus Naur Form:

```

/({ARPackage of System})1..n
{System.name}/
{RootSwCompositionPrototype.name}/
({SwComponentPrototype
  - part of referenced SwComponentType of
    RootSwCompositionPrototype or
    parent SwComponentPrototype resp.})0..n
{PortPrototype of referenced SwComponentType}
(/{DataPrototype of referenced PortInterface or
  within parent data prototype})1..n

```

Parent sw component prototype means the *SwComponentPrototype* referring to the *SwComponentType* the *SwComponentPrototype* is part of. E.g. in the cp-path

```
/OEM1/Systems/System/TopLvl/Pt/Eng/EngN/EngN
```

“Pt” is the parent sw component prototype of “Eng” (see example later in document).

**Definition.** The instance pb-path (*PortPrototypeBlueprint-Path*) is the instance path to the port prototype blueprint + one of the data prototypes of the port interface of the port prototype blueprint:

{instance path to port prototype blueprint}/{data prototype}<sub>1..n</sub>  
e.g.

```
/AUTOSAR/PortPrototypeBlueprints_Blueprint/EngN/EngN
```

i.e. in Backus Naur Form:

```

/({ARPackage of PortPrototypeBlueprint})1..n
{PortPrototypeBlueprint}
(/{DataPrototype of referenced PortInterface or
  within parent data prototype})1..n

```

## 6 Overall Modeling Rules

[[TR\_MCM\_7005]] The model and the generated model elements shall be compliant to the Meta Model. ] ()

The meta model parts relevant for this document are documented in [4], [5], [6] and [7].

[TR\_MCM\_7010]] The model shall be compliant to the Shall-Rules defined in AUTOSAR\_TR\_SWCModelingGuide [1]. Recommendations and optional rules, however, may be violated. ] ()

[TR\_MCG\_0001]] The generated display names shall be conformant to the C programming language. ] ()

For explanation of “shall” and “recommendation/should” please see chapter 3.1.

There are two separate display name schemas, one for prototypes and one for *ARElements*.

[TR\_MCG\_0785]] For *ARElements* we have the following name parts:

```
element : {nameSpace}_ {data}
with
  nameSpace : {ARPackage.name} (_ {ARPackage.name}) _ 0..n
  data : {ARElement.name} (_ {typeId})
] ()
```

The {nameSpace} corresponds to the package hierarchy since *ARPackages* define name spaces. The {typeId} corresponds to additional information, e.g. for calibration parameters.

Elements discussed in the following are *PortPrototypeBlueprints*, *SwSystemconsts*, *Units* and *Systems*.

[TR\_MCG\_0786]] For prototypes we have the following name parts:

```
prototype : {nameSpace}_ {componentHierarchy}_ {data}
with
  nameSpace : ({ARPackage.name}_ ) _ 1..n {System.name}
  componentHierarchy : ({SwComponentPrototype.name}_ ) _ 0..n
or
  componentHierarchy : {RootSwCompositionPrototype.name}
```

The {data} differs depending on the kind of the prototype.

] ()

The {namespace} corresponds to path to the relevant system. The {componentHierarchy} considers the *SwComponentPrototypes* hierarchy including multiple instantiation. The *RootSwCompositionPrototype* can typically be ignored in the {componentHierarchy} because there is exactly one within a system. Only if the port directly belongs to the root component then its name is relevant.

The {data} refers to the port and one of the data prototypes of its port interface plus an optional suffix. The details on {data} see next chapters.

Prototypes discussed in the following are *SwComponentPrototypes* and *DataPrototypes* of a *PortInterface* or of a complex data type like e.g. an array. Note: *PortPrototypeBlueprints* are elements, no prototypes. *PortPrototype* names themselves are only relevant as part of display names but not for themselves in the context of measurement and calibration.

In the following we will have a closer look at every name part and will discuss how to shorten it or discuss in which cases the name part can be omitted completely.

## 6.1 Name Part Length

A display name of a sw signal should not exceed 31 characters. Therefore we recommend to fulfill the following rules. We are aware that this will not always be possible.

[TR\_MCM\_7022]「 The {nameSpace} of a sw signal should not exceed 3 characters. 」 ()

[TR\_MCM\_7020]「 It is recommended to further restrict the length of model element names relevant for generating the display names. *DataPrototype* names and *PortPrototype* names should not exceed 23 characters. 」 ()

[TR\_MCM\_7390]「 The suffixes added to distinguish different type identifiers ({typeId}) of *DataPrototypes* should not exceed 3 characters. 」 ()

## 7 DataPrototypes

In this chapter we deal with the {data} name part. The general schema is (see [TR\_MCG\_0786]):

```
data : {port}_{dataInfo}_{typeId}
```

[TR\_MCG\_0005] For {data} of data prototypes we have the following name parts:

```
data : {port}_{dataInfo}(_{typeId})0..1
```

{port} may refer to *PortPrototype* or *PortPrototypeBlueprint*, i.e.

```
port : (PortPrototype.name)
```

or

```
port : {PortPrototypeBlueprint.name}
```

{dataInfo} refers to one data element within the port interface of the port. In case of complex data types however the data element of the port interface is not sufficient but additionally the type hierarchy to the final data element has to be considered.

In case of primitive types:

```
dataInfo : {DataPrototype.name}
```

In case of complex types:

```
dataInfo :  
{DataPrototype.name}(_{parentData})0..n{DataPrototype.name}
```

with

```
parentData : ({DataPrototype.name}|{index})
```

{typeId} depends on the type of the final data prototype. {index} is necessary in case of array elements.

┘ ()

Example:

Assume we have the port prototype LockgCenSts referencing *PortInterface* LockgCenSts with data element LockgCenSts with the following *ApplicationRecord-DataType*:

```
ShortName: LockgCenSts1  
recordElement 1: LockgSt  
recordElement 2: TrigSrc
```

Then we would have

```
data : LockgCenSts_LockgCenSts_LockgSt
```

or

```
data : LockgCenSts_LockgCenSts_TrigSrc
```

depending which of the two sw signals you are interested in.

Since *ArrayElements* do not have names only the index number can be used for automatic generation. So in general it is better to define display names manually in this case.

Arrays are handled in chapter 7.4, they will need the {index} name part. Note that the {parentData} above may be an *ArrayElement*.

The {data} name part is often quite long, contains a lot of redundancy and does not always fulfill all the requirements. Depending on specific additional rules and assumptions this name can be shortened.

These rules and assumptions are documented in the following.

Only one of the following four rules [TR\_MCG\_0010], [TR\_MCG\_0015], [TR\_MCG\_0020] and [TR\_MCG\_0030] can be applied.

[TR\_MCG\_0010] If the data element name is identical to the port name then the port prototype or port prototype blueprint name can be ignored within the display name of the element. We have

```
data : {dataInfo}(_{typeId})0..1  
] ()
```

[TR\_MCG\_0015] If the data element name is identical to “Val” (for “Value”) then the data prototype name can be ignored within the display name.

In case of primitive types we have:

```
data : {port}(_{typeId})0..1
```

In case of complex types we have:

```
data : {port}_{dataInfo}(_{typeId})0..1
```

with

```
dataInfo :  
  ({parentData}_)0..n{DataPrototype.name}
```

```
] ()
```

[TR\_MCG\_0020] If the *PortInterface* contains exactly one data element then the data element name can be ignored within the display name.

In case of primitive types we have:

```
data : {port}(_{typeId})0..1
```

In case of complex types we have:

```
data : {port}_{dataInfo}(_{typeId})0..1
```

with

```
dataInfo :  
  ({parentData}_)0..n{DataPrototype.name}
```

```
] ()
```

[TR\_MCG\_0030] If

1) the *PortInterface* contains more than one data element and

2) the *PortPrototype* or *PortPrototypeBlueprint* name is a substring of the data element name

then the port name can be ignored in the display name. We have

```
data : {dataInfo}(_{typeId})0..1  
] ()
```

Example 1:

Take the following examples for {port}\_{data} from Figure 2 in chapter 5.2 (with primitive data type):

EngN\_EngN  
EngNMax\_EngN

and the following example:

EngN\_Val

And finally the following example assuming the port interface has two data prototypes EngNMax and EngNMin:

EngN\_EngNMax

Rules [TR\_MCG\_0010], [TR\_MCG\_0015], [TR\_MCG\_0020] and [TR\_MCG\_0030] then would lead to

EngN or EngN  
EngNMax  
EngN  
EngNMax

Example 2:

See above, our record example LockgCenSts would then lead to ([TR\_MCG\_0010]):

data : LockgCenSts\_LockgSt

or

data : LockgCenSts\_TrigSrc

depending which of the two sw signals you are interested in.

Similar rules can be added for data prototypes within complex data types.

## 7.1 Increase Readability of Display Names

To further increase the readability of display names the following modeling rules are recommended.

Our examples from the previous chapter were very short, e.g.:

EngN

However, the generated name could also look like this:

TrsmCluStTar

and may even be much longer.

In this case it would improve readability to point out the mean/source as well as the physical or logical meaning of the *DataPrototype* and add an underscore before it:

TrsmClu\_**St**Tar

To even improve readability more the physical or logical types should be written in small letters:

TrsmClu\_**st**Tar

To be able to do so additional generating and modeling rules are necessary:

[TR\_MCM\_7050]「 There shall be a small and manageable list of physical and logical types for generating display names. This list is called the P/L-list.」 ()

In chapter 14 a proposal for such a P/L-List for the powertrain domain is presented.

[TR\_MCM\_7060]「 The P/L-List is a subset of the keyword abbreviations from AUTOSAR\_MOD\_AISpecification [3].」 ()

However, not all physical types or actions of AUTOSAR\_MOD\_AISpecification [3] are elements of the P/L-list; it would be too large. Also other keywords not belonging to the semantic field “Physical Type/Action” might be part of the P/L-list. For example no logical types are contained in the field “Physical type/Action” in the modeling guideline but these are very useful for distinguishing control flow and functional software signals.

[TR\_MCM\_7065]「 Assuming the P/L-list is not empty every *DataPrototype* name should contain at least one keyword from the P/L-list.」 ()

The P/L-list can be empty. Then there is no effect on display name generation and modeling.

[TR\_MCG\_0070]「 For generating the display name only the first occurrence of a keyword out of the P/L-list within the port or data element name is relevant: an underscore is added before and the keyword is written in small letters. If there are both, port prototype name and data prototype name then the underscore between them is deleted.」 ()

Example:

Assume we have {port}\_{DataPrototype.name} : EngN\_Max. Then rule [TR\_MCG\_0070] leads to  
Eng\_nMax.

[TR\_MCG\_0080]「 If a name starts with a keyword out of the P/L-list then no “\_” is added before the keyword from the P/L-list.」 ()

Reason: Within the *FlatMap Identifiables* are used for the display name. For *Identifiables* it is not allowed to start the name with “\_” or to contain “\_” [7].

[TR\_MCM\_7070]「 A *PortPrototype* or *PortPrototypeBlueprint* name should not start with a keyword out of the P/L-list. I.e. it is recommended to first add a mean or other semantic information to it.」 ()

The rules above lead to the following schema in syntax of name pattern language of [6]:

```
display Name   :   ({part1}_){0..1} ({physLog}){0..1} {part2}(_{typeId}){0..1}
with
part1          :   ( ({keyword}) 1..n ({index}) 0..n )1..n
physLog        :   {keyword from P/L-List, written in small letters}
part2          :   ({keyword} | {index} | '_' )0..n
typeId         :   { 'MP' | 'SC' | 'C' | 'T' | 'M' | 'CA' | 'Ax' }
```

The kind of suffix allowed are described in the following.

## 7.2 VariableDataPrototypes

To distinguish between variables that are needed for inter-runnable exchange and variables that are only defined for measurement purposes, the following rule is defined:

[TR\_MCG\_0090]「 Add a suffix “\_MP” for the display name of *VariableDataPrototypes* that are measurement points. We have  
     data : {port}\_{dataInfo }\_MP  
 」 ()

Whether a data prototype is a measurement point or not is specified within *SwImplPolicy* of *SwDataDefProps*.

Example:

Assume the following software signal to be defined only for measurement purposes:

TrsmClu\_stTar

With rule [TR\_MCG\_0090] we then have

TrsmClu\_stTar\_MP



### 7.3 ParameterDataPrototypes

The type of a parameter is characterized by the category of its data type, e.g. MAP, CURVE [5]. A *DataPrototype* is in the role of a parameter if it is an *ApplicationCompositeElementDataPrototype* being part of a *ParameterDataPrototype* or it is a *ParameterDataPrototype* itself.

It increases readability if it can be seen at one glance whether it is a variable or a parameter. Thus the following rules are added to distinguish parameters from variables:

[TR\_MCG\_0310] ⌈ Add a suffix “\_C” for “constant” for the display name of a *DataPrototype* in the role of a parameter being a single data point (*Category of ApplicationDataType* equal to VALUE, STRING or BOOLEAN). We have

data : {port}\_{dataInfo}\_C

⌋ ()

[TR\_MCG\_0320] ⌈ Add a suffix “\_T” for “table” or “curve” for the display name of a *DataPrototype* in the role of a parameter being a set of data points with one axis (dependency) (*Category of ApplicationDataType* equal to CURVE). We have

data : {port}\_{dataInfo}\_T

⌋ ()

[TR\_MCG\_0330] ⌈ Add a suffix “\_M” for “map”, i.e. a *DataPrototype* in the role of a parameter with a two dimensional set of data points with two axes (dependencies) (*Category of ApplicationDataType* equal to MAP). We have

data : {port}\_{dataInfo}\_M

⌋ ()

[TR\_MCG\_0340] ⌈ Add a suffix “\_CA” for “array of calibration parameters” for the display name of a *DataPrototype* in the role of a parameter of type *ApplicationArrayDataType* with type of *ApplicationArrayElements* of Category CURVE or VAL\_BLK. We have

data : {port}\_{dataInfo}\_CA

⌋ ()

[TR\_MCG\_0350] ⌈ Add a suffix “\_Ax” for “axis” for the display name of a *DataPrototype* in the role of a parameter with axis elements (Category COM\_AXIS and RES\_AXIS). We have

data : {port}\_{dataInfo}\_Ax

⌋ ()

For the other categories of *ParameterDataPrototypes* additional suffixes should be added.

Example:

CtryNrCod

would lead to (assuming CtryNrCod is a parameter of Category VALUE, “Nr” is part of the P/L-list and applying rule [TR\_MCG\_0310]):

Ctry\_nrCod\_C

## 7.4 DataPrototypes of ApplicationArrayType

For arrays there are two possibilities: Define a display name for the complete array and use the display name as derived for the *DataPrototype*, i.e.

```
data : {port}_{dataInfo}_{typeId}
```

or add information about the single *ArrayElements*

```
data : {port}_{dataInfo}_{arrayElement}_{typeId}
```

Since *ArrayElements* do not have names only the index number can be used for automatic generation. We have

```
arrayElement : {index}
```

So in general it is better to define display names manually in this case.

[TR\_MCG\_0400] In case of *DataPrototypes* of an *ApplicationArrayType* the display name of the *ArrayElement* can be generated by using the display name of the *ApplicationArrayType* as basis and adding an array index. We have

```
data : {dataInfo}{index}(_{typeId})_{0..1}
```

」()

The {typeId} is added depending on type of elements.

Example:

Display name of the single elements of the 4-element array

EscVWhlInd

could be generated by using [TR\_MCG\_0400]

Esc\_vWhlInd\_0, Esc\_vWhlInd\_1, Esc\_vWhlInd\_2 and Esc\_vWhlInd\_3

or – manually extending the generated display name of the array itself –

Esc\_vWhlIndFrntLe, Esc\_vWhlIndFrntRi,

Esc\_vWhlIndReLe and Esc\_vWhlIndReRi

for front left wheel etc.

## 8 Name Space

In this chapter we deal with the name space identifier {nameSpace}. The general schema for the name space for *ARElements* ({element}) is (see [TR\_MCG\_0785])

```
nameSpace : ( {ARPackage.name} / )1..n
```

The general schema of the name space for prototypes ({prototype}) is (see [TR\_MCG\_0786]):

```
nameSpace : ( {ARPackage.name} / )1..n {System.name}
```

Details on the name space concept of AUTOSAR can be found in [7]. Virtual name spaces are defined in chapter 5.5.

Every element is part of an *ARPackage*.

Some prototype elements like *SwComponentPrototypes* are element of a different name space, for *SwComponentPrototypes* it is the *SwComponentType*. For those the {componentHierarchy} is more important than the *ARPackage* the component type is part of. But for *ARElements* like *PortPrototypeBlueprints*, *Units*, *System* etc. the {nameSpace} is very important.

In the following some rules are given how to deal with the {nameSpace}.

[TR\_MCM\_0030]「 An arbitrary abbreviation for the {nameSpace} may be defined for the elements within an *ARPackage*.」 ()

[TR\_MCM\_7040]「 For elements within the predefined *ARPackage* “AUTOSAR” the abbreviation “AR” should be used as part of the {nameSpace}.」 ()

[TR\_MCG\_0035]「 Package paths within generated display names should be shortened by using defined abbreviations」 ().

Example:

Assume we define the following abbreviations for the paths

AUTOSAR/SwComponentTypes → [AR\\_Types](#)

Then rule [TR\_MCG\_0035] would lead to

nameSpace : [AR\\_Types](#)

instead of AUTOSAR\_SwComponentTypes

Please note too that the short label was chosen as “AR\_Types” and not “Types” or “hugo”: This is because [TR\_MCM\_7040] to have a name space for AUTOSAR display names still should be fulfilled.

[TR\_MCG\_0120] If an *ARPackage* belongs to a virtual name space for the element class then the package name can be ignored in the display name of this element. Only the name of the *ARPackage* defining the virtual name space and the {nameSpace} of this *ARPackage* itself are to be considered.

Example:  
In

/AUTOSAR/AISpecification/PortPrototypesBlueprints\_Blueprint/  
/AUTOSAR/ currently defines a virtual name space for all port prototypes blueprints. Therefore with [TR\_MCG\_0120] the display name of a blueprint can be shortened to

element : {portBlueprint} : AUTOSAR\_{data}

With [TR\_MCG\_0035] and [TR\_MCM\_7040] the name can further be shortened to  
AR\_{data}

If we could not ensure uniqueness for /AUTOSAR/ then we could only say that /AUTOSAR/AISpecification defines a virtual name space for all port prototypes blueprints. Therefore with [TR\_MCG\_0120] the display name of a blueprint can only be shortened to

portBlueprint : AUTOSAR\_AISpecification\_{data}  
AUTOSAR is the {nameSpace} of *ARPackage* AISpecification.

[TR\_MCA\_8710]「 Assumption: There is only one *System* relevant for display name generation.」()

This assumption is typically fulfilled because there is no need to have display names consistent over different systems: a *FlatMap* always covers only one system or one ECU but in different variations.

[TR\_MCG\_0713]「 If there is only one system the {nameSpace}, i.e. the names of the *ARPackages* as well as the system name, can be ignored in the display name of a prototype element. We have

prototype : {componentHierarchy}\_{data}

Reason: the starting point is the *RootSwCompositionPrototype* for the *System*.

」()

In rule [TR\_MCG\_0713] assumption [TR\_MCA\_8710] is assumed to be fulfilled.

If there is a *FlatMap* defining a unique name for a *System* also this name can be used for {nameSpace}.

[TR\_MCG\_0770] 「 The display names specified in a *FlatMap* for *System* can be used for specifying the {nameSpace} name part of the display name of a sw signal.  
General rule:

nameSpace : {systemDescriptor}

」 ()

## 9 SwSystemconsts

Even constants values already evaluated at compile time can be visible (*readOnly*) within a MCD system. For system constants (element : {systemconst}) the general schema is defined as follows (see [TR\_MCG\_0785]):

```
systemconst : {nameSpace}_{SwSystemconst.name}_{typeId}
```

[TR\_MCG\_0510] 「 Add a suffix “\_SC” for “software configuration” for the display name of a *SwSystemconst*. We have

```
systemconst : {nameSpace}_{data}
with
data : {SwSystemconst.name}_SC
」 ()
```

[TR\_MCG\_0520]「 For a display name of a *SwSystemconst* use only capital letters.  
」 ()

Please note: [TR\_MCG\_0070] does not hold for system constants. This is because system constants typically do not contain physical or logical information but represent a feature or functionality.

[TR\_MCA\_8530]「 Assumption: There is exactly one virtual name space sw for system constants.」 ()

[TR\_MCG\_0512] 「 If there is only one system and exactly one virtual name space for *SwSystemconsts* we have

```
systemconst : {SwSystemconst.name}_SC
」 ()
```

In rule [TR\_MCG\_0512] assumption [TR\_MCA\_8530] and [TR\_MCA\_8710] are assumed to be fulfilled.

Example:

/OEM1/SwSystemconsts/EngNrCyl  
would lead to (applying [TR\_MCG\_0510], [TR\_MCG\_0520], assuming uniqueness of names ( TR\_MCA\_8530)), even assuming “Nr” is part of the P/L-list):  
ENGNR CYL **SC**

## 10 PortPrototypeBlueprints

In this chapter we deal with unique names for port prototype blueprints. Since *PortPrototypeBlueprints* are *ARElements* ( $\{\text{element}\} : \{\text{portBlueprint}\}$ ) the general schema is (see [TR\_MCG\_0785]):

```
portBlueprint : {nameSpace}_{data}
with
  nameSpace : ( {ARPackage.name}_ )1..n
  data : {port}_{dataInfo}
```

$\{\text{typeId}\}$  is not relevant for port prototype blueprints because they cannot be measured or calibrated themselves but may serve only as part of display names of sw signals.

[TR\_MCG\_0788] 「 For port prototype blueprints the following basic display name schema

```
portBlueprint : {nameSpace}_ {data}
```

is used for generating the display name of a single data element.

」 ()

The corresponding path of a port prototype blueprint is

```
( {ARPackage.name}/ )1..n {PortPrototypeBlueprint.name}
```

The port interface names are ignored.

For the data prototype EngN in example in Figure 3 in chapter 5.2 the complete instance pb-path would be

```
AUTOSAR/PortPrototypeBlueprints_Blueprint/EngN/EngN
```

and the generated display name according to [TR\_MCG\_0788] without any shortening would be

```
AUTOSAR_PortPrototypeBlueprints_Blueprint_EngN_EngN
```

This name is quite long, contains a lot of redundancy and does not fulfill all the requirements. Depending on specific additional rules and assumptions this name can be shortened. The rules for the port and data prototypes can be found in chapter 7. The rules for shortening the name space identifier can be found in chapter 8. The overall rules can be found in chapter 6.

In the following we will make some assumptions about virtual name spaces. If these assumptions are fulfilled in a system then the display names can be further shortened.

Virtual name spaces are defined in chapter 5.5.

Example (continue Figure 2 and Figure 3 in chapter 5.2) :

Assume the *ARPackages* “AUTOSAR” and “OEM1” define a virtual name space for port prototype blueprints. Within this virtual name space the port prototype blueprint

names are unique. Given this additional information the generated display name for data prototype “EngN” will be ([TR\_MCM\_7040], [TR\_MCG\_0120]):

AR\_EngN

If there is only one virtual name space ([TR\_MCA\_8030]) covering the packages “AUTOSAR” and “OEM1” the name can even be reduced to

EngN

In the following we will express these assumptions and rules formally:

Port prototype blueprint names are unique within a given package, no additional assumption w.r.t. uniqueness of port prototype blueprint names within a given *ARPackage* is needed.

For being able to apply [TR\_MCG\_0120] (see chapter 8) the following assumption should be fulfilled for port prototype blueprints:

[TR\_MCA\_8020]「 Assumption: Each Top-Level *ARPackages* defines a virtual name space for port prototype blueprints.」 ()

The following rule is subsumed in rule [TR\_MCG\_0120] and thus does not define anything new. For better understandability it is stated explicitly.

[TR\_MCG\_0040]「 Only packages explicitly defining virtual name spaces for port prototype blueprints within a product or product family shall be considered in the display name. 」 ()

Example:

With rule [TR\_MCG\_0040] and [TR\_MCM\_7040] and assumption [TR\_MCA\_8020]

AR\_PortPrototypeBlueprints\_Blueprint\_EngNMax  
could be shortened to  
AR\_EngNMax

[TR\_MCA\_8030]「 Assumption: There is exactly one virtual name space for port prototype blueprints.」 ()

[TR\_MCG\_0045]「 If there is only one virtual name space for port prototype blueprints the {nameSpace}, i.e. all *ARPackage* names, can be completely ignored in the display name. We have

portBlueprint : {data}

」 ()



In [TR\_MCG\_0045] assumption [TR\_MCA\_8030] is assumed to be fulfilled.

Example:

With rule [TR\_MCG\_0045] and assumption [TR\_MCA\_8030]

AR\_EngNMax

could be further shortened to

EngNMax

## 11 ComponentHierarchy

In this chapter we deal with the component hierarchy {componentHierarchy}. The {componentHierarchy} is relevant for prototypes ({prototype}). The general schema is:

```
componentHierarchy : {RootSwCompositionPrototype.name}  
                    (_{SwComponentPrototype.name}_)0..n
```

Component hierarchies are relevant for sw signals because they are always instantiated in the context of a *SwComponentPrototype*. They are not relevant for port prototype blueprints and other first class elements of an *ARPackage*.

[TR\_MCG\_0705] 「 The {componentHierarchy} is defined as follows:

```
componentHierarchy : {RootSwCompositionPrototype.name}  
                    (_{SwComponentPrototype.name}_)0..n
```

」 ()

In rule [TR\_MCG\_0705] assumption [TR\_MCA\_8710] (see chapter 8) is assumed to be fulfilled.

[TR\_MCG\_0706] 「 If there is only one System relevant then the {componentHierarchy} is defined as follows:

```
componentHierarchy : ( {SwComponentPrototype.name}_ )1..n
```

or

```
componentHierarchy : {RootSwCompositionPrototype.name}()
```

For being able to apply [TR\_MCG\_0120] (see chapter 8) one of the following assumptions should be fulfilled for sw component prototypes:

Similar to [TR\_MCA\_8020] we have:

[TR\_MCA\_8715]「 Assumption: Sw component prototype names are unique within a given *ARPackage*, i.e. this *ARPackage* defines a virtual name space for *SwComponentPrototypes*.」 ()

[TR\_MCA\_8720]] 「 Assumption: Each Top-Level *ARPackage* defines a separate virtual name space for sw component prototypes.

」 ()

[TR\_MCG\_0740]⌈ If an *ARPackage* defines a virtual name space for *SwComponentPrototypes* then the parent sw component prototype names of the sw component prototypes defined within this package can be ignored in the display name of the sw signal or the sw component prototype, resp. - except for the first one and the ones that have the same *SwComponentType*, i.e. that are multiple instances of the same *SwComponentType*.⌋ ()

In rule [TR\_MCG\_0740] assumption [TR\_MCA\_8715] is assumed to be fulfilled.

The first sw component prototype name is necessary because it cannot be excluded that there are sw component prototypes with the same name in different packages.

[TR\_MCA\_8730]⌈ Assumption: There is exactly one virtual name space for sw component prototypes. ⌋ ()

[TR\_MCG\_0750]⌈ If there is only one virtual name space for all sw component prototypes then the parent component names can be ignored in the display name except the ones that have the same *SwComponentType*, i.e. that are multiple instances of the same *SwComponentType*.⌋ ()

In [TR\_MCG\_0750] assumption [TR\_MCA\_8730] is assumed to be fulfilled.

Generation rule [TR\_MCG\_0750] shorten the display name. However, strange names might be the result. So they should be checked and if necessary changed manually.

.

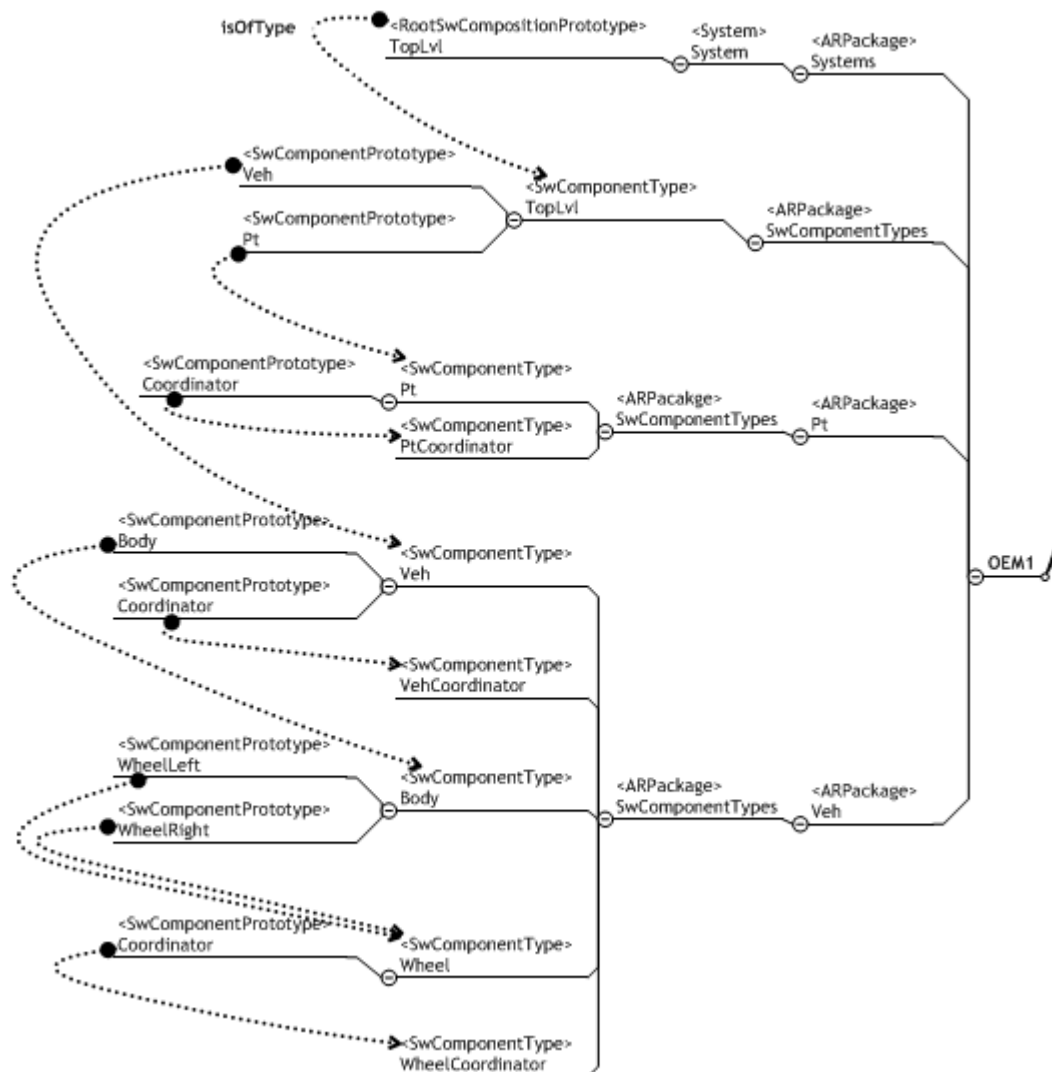


Figure 5 Example with Multiple Instantiation – Coordinator Wheels

Example (see Figure 5):

For *SwComponentType* **Wheel** we have two instances: **WheelLeft** and **WheelRight**.

For Coordinator we have the following paths:

/OEM1/Systems/System/TopLvl/Pt/Coordinator

and

/OEM1/Systems/System/TopLvl/Veh/Coordinator

/OEM1/Systems/System/TopLvl/Veh/Body/WheelLeft/Coordinator

/OEM1/Systems/System/TopLvl/Veh/Body/WheelRight/Coordinator

We have only one system, so [TR\_MCA\_8710] is fulfilled and [TR\_MCG\_0713] can be applied. Every top-level package defines a virtual name space for *SwComponent-Prototypes*, i.e. [TR\_MCA\_8720] is fulfilled. However, there is no global virtual name space for sw component prototypes, i.e. [TR\_MCA\_8730] is not fulfilled.

Assume we want to create display names for all sw component prototypes with name "Coordinator".

With rule [TR\_MCG\_0706] we would have the following display names for sw component prototype hierarchies ({componentHierarchy}):

Pt\_Coordinator  
Veh\_Coordinator  
Veh\_Body\_WheelLeft\_Coordinator  
and  
Veh\_Body\_WheelRight\_Coordinator

With rule [TR\_MCG\_0740] we have:

Pt\_Coordinator  
Veh\_Coordinator  
Veh\_WheelLeft\_Coordinator  
and  
Veh\_WheelRight\_Coordinator

## 12 SwComponentPrototypes

In this chapter we deal with the display name of a single *SwComponentPrototype*. The general schema for sw component prototypes (prototype : {component}) is

```
component : {nameSpace}_{componentHierarchy}_{data}
```

{data} corresponds to the name of the sw component prototype or root composition. I.e. the last component prototype is not part of the {componentHierarchy}. We have

```
component :  
    {nameSpace}_{componentHierarchy}_{SwComponentPrototype.name}  
or  
component :  
    {nameSpace}_{RootSwCompositionPrototype.name}
```

Component hierarchies are relevant for sw signals because they are always instantiated in the context of a *SwComponentPrototype*. They are not relevant for port prototype blueprints and other first class elements of an *ARPackage* ({element}).

[TR\_MCG\_0610] For sw component prototypes the following basic display name schema

```
component :  
    {nameSpace}_{componentHierarchy}_{SwComponentPrototype.name}
```

is used for generating the display names.

In case the *SwComponentPrototype* corresponds to the *RootSwCompositionPrototype* we have

```
component : {nameSpace}_{RootSwCompositionPrototype.name}
```

」()

The corresponding path of a component is

```
(({ARPackage.name})/)1..n ({System.name})/ ({RootSwCompositionPrototype.name})/  
 ({SwComponentPrototype.name})/)0..n ({SwComponentPrototype.name})0..1
```

## 13 Unique SW-Signal Display Names

The general schema for sw signals looks like this ((prototype) : {swSignal}):

```
swSignal : {nameSpace}_{componentHierarchy}_{data}
with
    data : {port}_{dataInfo}_{typeId}
```

[TR\_MCG\_0787] is derived from [TR\_MCG\_0001]:

[TR\_MCG\_0787] 「 For multiple instantiation and in case several systems have to be considered the following display name schema for a sw signal

```
swSignal : {nameSpace}_{componentHierarchy}_{data}
```

is used for generating the display names of sw signals.

」 ()

The instance cp-path of a sw signal is

```
(({ARPackage.name})/){1..n}{System.name}/
{RootSwCompositionPrototype.name}/{(SwComponentPrototype.name)/}{0..n}
{PortPrototype.name}/{(DataPrototype.name)}{1..n}
```

Example: (see Figure 6 ExtrLi)

Several systems to be considered.

SwComponentType: **ExtrLiAdpr**

SwComponentPrototypes referencing Type **ExtrLiAdpr**: **ExtrLiAdprFrntLe** and **ExtrLiAdprFrntRi**

Given the instance cp-path

**OEM1/Systems/System/TopLvl/ExtrLiAdprFrntRi/ActvnOfIndcFrntRi/Cmd**

this would lead to the following display name with rule [TR\_MCG\_0787]:

**OEM1\_Systems\_System\_ ExtrLiAdprFrntRi\_ActvnOfIndcFrntRi\_Cmd**

The following rule is derived from [TR\_MCG\_0713]. Additionally assumption [TR\_MCA\_8710] is assumed to be fulfilled.

[TR\_MCG\_0716] 「 For multiple instantiation for a single system the following basic display name schema

```
swSignal : {componentHierarchy}_{data}
```

is used for generating the display name of a sw signal

The {nameSpace}, the sw component type names, the package names and port interface names are ignored.

」 ()

In rule [TR\_MCG\_0716] assumption [TR\_MCA\_8710] is assumed to be fulfilled.

Example: (see Figure 6 ExtrLi)  
Assumption: only one System

SwComponentType: **ExtrLiAdpr**

SwComponentPrototypes referencing Type **ExtrLiAdpr**: **ExtrLiAdprFrntLe** and **ExtrLiAdprFrntRi**

Given the instance cp-path

**OEM1/Systems/System/TopLvl/ExtrLiAdprFrntRi/ActvnOfIndcrFrntRi/Cmd**

this would lead to the following display name applying [TR\_MCG\_0716]:

**ExtrLiAdprFrntRi\_ActvnOfIndcrFrntRi\_Cmd**

And after applying [TR\_MCG\_0020], chapter 8, we would have

**ExtrLiAdprFrntRi\_ActvnOfIndcrFrntRi**

Since there has to be a *FlatMap* for the ECU-Extract specifying unique names for component prototypes the rules in chapter 13 can be applied to generate these unique component prototype names. Rule [TR\_MCG\_0760] can substitute rule [TR\_MCG\_0787].

[TR\_MCG\_0760] The unique component prototype names specified in a *FlatMap* can be used for specifying part of the display name of a sw signal. General Rule:

swSignal : {componentDescriptor}\_{data}

」 ()

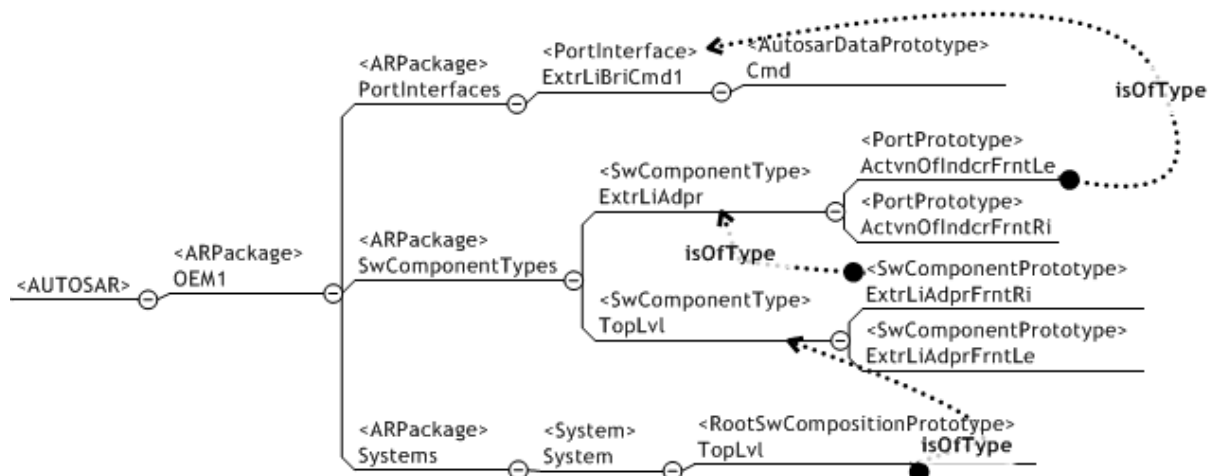


Figure 6 Example ExtrLi with Multiple Instantiation

Example (Figure 6 ExtrLi):

Assume we have the following *FlatMap* defining names for sw component prototypes:

```
<FLAT-MAP>
  <SHORT-NAME>ARMap< / SHORT-NAME>
  <INSTANCES>
```



```
<FLAT-INSTANCE-DESCRIPTOR>
  <SHORT-NAME>I1</SHORT-NAME>
  <UPSTREAM-REFERENCE-IREF>
    <CONTEXT-ELEMENT-REF DEST="ROOT-SW-COMPOSITION-PROTOTYPE">
      /OEM1/Systems/System/TopLvl
    </CONTEXT-ELEMENT-REF>
    <TARGET-REF DEST="SW-COMPONENT-PROTOTYPE">
      /OEM1/SwComponentTypes/TopLvl/ExtrLiAdprFrntRi
    </TARGET-REF>
  </UPSTREAM-REFERENCE-IREF>
</FLAT-INSTANCE-DESCRIPTOR>
</INSTANCES>
</FLAT-MAP>
```

Instead of

**TopLvl\_ExtrLiAdprFrntRi\_ActvnOfIndcFrntRi\_Cmd**  
with rule [TR\_MCG\_0760] and [TR\_MCG\_0020] and **I1** being the unique short name defined in the *FlatMap* for **TopLvl\_ExtrLiAdprFrntRi** we have:  
**I1\_ActvnOfIndcFrntRi**

[TR\_MCA\_0789]「 Assumption: There is exactly one virtual name space for port prototypes.」 ()

Similar to [TR\_MCG\_0045] we have the following rule:

[TR\_MCG\_0752]「 If there is only one virtual name space for port prototypes and the sw component types are only instantiated once then the component prototype names (the {componentHierarchy}) and the {nameSpace} can be ignored in the display name of a sw signal. We simply have:

```
swSignal : {data}
```

」 ()

If the port prototypes are based on port prototype blueprints the following rule holds:

[TR\_MCG\_0780]「 The display names specified in a *FlatMap* for the data elements within port prototype blueprints can be used for specifying the display name of a sw signal derived from this blueprint:

```
swSignal : {portBlueprintDescriptor}
```

It has to be ensured that the display names of signals not derived from blueprints or for signals no flatmap with display names is available are different from those already normalized in the flatmaps for the blueprints.

」 ()

The {componentHierarchy} can be ignored because there is no multiple instantiation. The {nameSpace} can be ignored because it is already contained in the blueprint descriptor name.

## 14 Appendix: Keywords Phys/Log (P/L-List) for Powertrain Domain

The following table defines a P/L-list for the Powertrain Domain and can be used as input for the rules described in chapter 7.1.

The keyword abbreviations are a subset of the ones defined in AUTOSAR\_MOD\_AISpecification [3] and cover physical as well as logical aspects.

**Table 1 Keywords Phys/Log (P/L-List) for Powertrain Domain**

<b>Keyword Abbreviation</b>	<b>Description</b>
A	acceleration
Adr	address
Ag	angle
Ar	area
Cntr	counter
Fac	factor
Flg	flag, bit, boolean, binary signal
Frq	frequency
I	electric current
Idx	index
N	(rotational) speed
Nr	number
P	pressure
Posn	position
Pwr	power
R	resistance
Rat	ratio, duty cycle
St	state, status
T	temperature
Ti	time, duration
Tq	torque
U	voltage
V	velocity
Vol	volume
W	work