ETAS ASCET V6.4

AUTOSAR

User Guide

www.etas.com

ETAS ASCET V6.4 I AUTOSAR User Guide R09 EN I 06.2024

Contents

1 Introduction

In this chapter, you can find information about the intended use, the addressed target group, and information about safety and privacy related topics.

Please adhere to the ETAS Safety Advice (accessible via **Help › Product Disclaimer**) and to the safety information given in the user documentation.

ETAS GmbH cannot be made liable for damage which is caused by incorrect use and not adhering to the safety information.

1.1 Intended Use

The ASCET tools support model-based software development. In model-based development, you construct an executable specification – the model – of your system and establish its properties through simulation and testing in early stages of development. When a model behaves as required, it can be converted automatically to production-quality code.

ASCET-SE is the ASCET tool for generating software for embedded microcontrollers, or AUTOSAR XML code, from an ASCET-MD model. ASCET-SE uses the project to hold configuration information.

1.2 Target Group

This manual addresses qualified personnel working in the fields of automobile control unit development and calibration. Specialized knowledge in the areas of measurement and control unit technology is required, as well as knowledge of ASCET and (at least) basic knowledge of AUTOSAR.

Any user who is not familiar with ASCET should read the ASCET Getting Started manual before reading the AUTOSAR User Guide.

Any user who is not familiar with AUTOSAR should learn the relevant concepts before using the AUTOSAR features of ASCET.

1.3 Classification of Safety Messages

Safety messages warn of dangers that can lead to personal injury or damage to property:

DANGER

DANGER indicates a hazardous situation that, if not avoided, will result in death or serious injury.

WARNING

WARNING indicates a hazardous situation that, if not avoided, could result in death or serious injury.

CAUTION

CAUTION indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.

NOTICE

NOTICE indicates a situation that, if not avoided, could result in damage to property.

1.4 Safety Information

Observe the following safety information when using the NVRAM capabilities of the ASCET-RP or ASCET-SE targets to avoid injury to yourself and others as well as damage to property:

WARNING

Harm or property damage due to unpredictable behavior of vehicle or test bench

Wrongly initialized NVRAM variables can lead to unpredictable behavior of a vehicle or a test bench. This behavior can cause harm or property damage.

ASCET projects that use the NVRAM possibilities of AUTOSAR expect a *user-defined* initialization that checks whether all NV variables are valid for the current project, both individually and in combination with other NV variables. If this is not the case, all NV variables have to be initialized with their (reasonable) default values.

Due to the NVRAM saving concept, this is *absolutely necessary* when projects are used in environments where any harm to people and equipment can happen when unsuitable initialization values are used (e.g., in-vehicle-use or at test benches).

Adhere to the ETAS Safety Advice and the safety information given in the online help and user guides. You can open the ETAS Safety Advice from the main ASCET window with **Help > Product Disclaimer**. A PDF version is available on the installation medium: `Documentation\ETAS Safety Advice.pdf`

In addition, take all information on environmental conditions into consideration before setup and operation (see the documentation of your computer, hardware, etc.).

Further safety advice for this ETAS product is available in the ASCET safety manual, available at ETAS upon request.

1.5 Data Protection

If the product contains functions that process personal data, legal requirements of data protection and data privacy laws shall be complied with by the customer.

As the data controller, the customer usually designs subsequent processing. Therefore, he must check if the protective measures are sufficient.

1.6 Data and Information Security

To securely handle data in the context of this product, see the next sections about data and storage locations as well as technical and organizational measures.

1.6.1 Data and Storage Locations

The following sections give information about data and their respective storage locations for various use cases.

1.6.1.1 License Management

When using the ETAS License Manager in combination with user-based licenses that are managed on the FNP license server within the customer's network, the following data are stored for license management purposes:

Data

- Communication data: IP address
- User data: Windows user ID

Storage location

- FNP license server log files on the customer network

When using the ETAS License Manager in combination with host-based licenses that are provided as FNE machine-based licenses, the following data are stored for license management purposes:

Data

- Activation data: Activation ID

  Used only for license activation, but not continuously during license usage

Storage location

- FNE trusted storage

  `C:\ProgramData\ETAS\FlexNet\fne\license\ts`

1.6.1.2 Problem Report

When an error occurs, ASCET offers to send an error report to ETAS for trouble-shooting. ETAS uses the pe*rs*onal information to have a contact person in case of system errors.

The problem report may contain the following personal data or data category:

Data

- Communication data: IP address
  - User data: Windows user ID, user name

Storage location:

- `EtasLogFiles<index number>.zip` in the ETAS-specific log files directory

Additionally to the problem information that is entered by the users themselves, ASCET collects the available product-related log files in a zip archive to support the bug fixing process at ETAS. The zip file is named according to the pattern `EtasLogFiles<index number>.zip`. See also chapter "Support Function for Feedback to ETAS in Case of Errors" in the ASCET Getting Started manual.

All ETAS-related log files in the ETAS-specific log files directory and the zip archives created by the Problem Report feature can be removed after closing all ETAS applications if they are no longer needed.

1.6.2 Technical and Organizational Measures

We recommend that your IT department takes appropriate technical and organizational measures, such as classic theft protection and access protection to hardware and software.

2 AUTOSAR Overview

Today, special effort is needed when integrating software components from different suppliers in a vehicle project comprising networks, electronic control units (ECUs), and dissimilar software architectures. While clearly limiting the reusability of automotive embedded software in different projects, this effort also calls for extra work in order to provide the required fully functional, tested, and qualified software.

By standardizing, inter alia, basic system functions and functional interfaces, the AUTOSAR partnership aims to simplify the joint development of software for automotive electronics, reduce its costs and time-to-market, enhance its quality, and provides mechanisms required for the design of safety relevant systems.

To reach these goals, AUTOSAR defines an architecture for automotive embedded software. It provides for the easy reuse, exchange, scaling, and integration of those ECU-independent that implement the functions of the respective application software components

The next sections briefly describe the AUTOSAR process for the development of application software components. For more detailed information, refer to the AUTOSAR documents at the AUTOSAR website: www.autosar.org/.

2.1 AUTOSAR Basic Approach

Application software is the name given in AUTOSAR to vehicle functions. Each application is decomposed into one or more *software components* (SWCs), which are designed to be both CPU- and location-neutral. An AUTOSAR application software component can be mapped to any available ECU during system configuration.

The abstraction of the SWC environment is called the *virtual function bus* (VFB). In each real AUTOSAR ECU, the VFB is mapped by a specific, ECU-dependent implementation of the platform software. The AUTOSAR platform software is split into two major areas of functionality: the runtime environment (RTE) and the *basic software* (BSW).

The BSW provides communications, I/O, and other functionality that all software components are likely to require, e.g., diagnostics and error reporting, or non-volatile memory management.

Application SWCs have no direct access to the BSW. This means that components cannot, for example, directly access operating system or communication services. The *runtime environment* provides the interface between software components, BSW modules, and operating systems (OS). Concerning the interconnection of SWCs, the RTE acts like a telephone switchboard. This is similarly true of components that reside either on single ECUs or on networked ECUs interconnected by vehicle buses.

In AUTOSAR, the OS calls the runnable entities of the SWCs through the RTE. RTE and OS are the key modules of the basic software with respect to controlling application software execution. ETAS offers the *RTA-RTE* AUTOSAR Runtime Environment and the *RTA-OS* AUTOSAR Operating System.

Based on their AUTOSAR interfaces, basic software modules from third-party suppliers can be seamlessly integrated with RTA-RTE and RTA-OS.

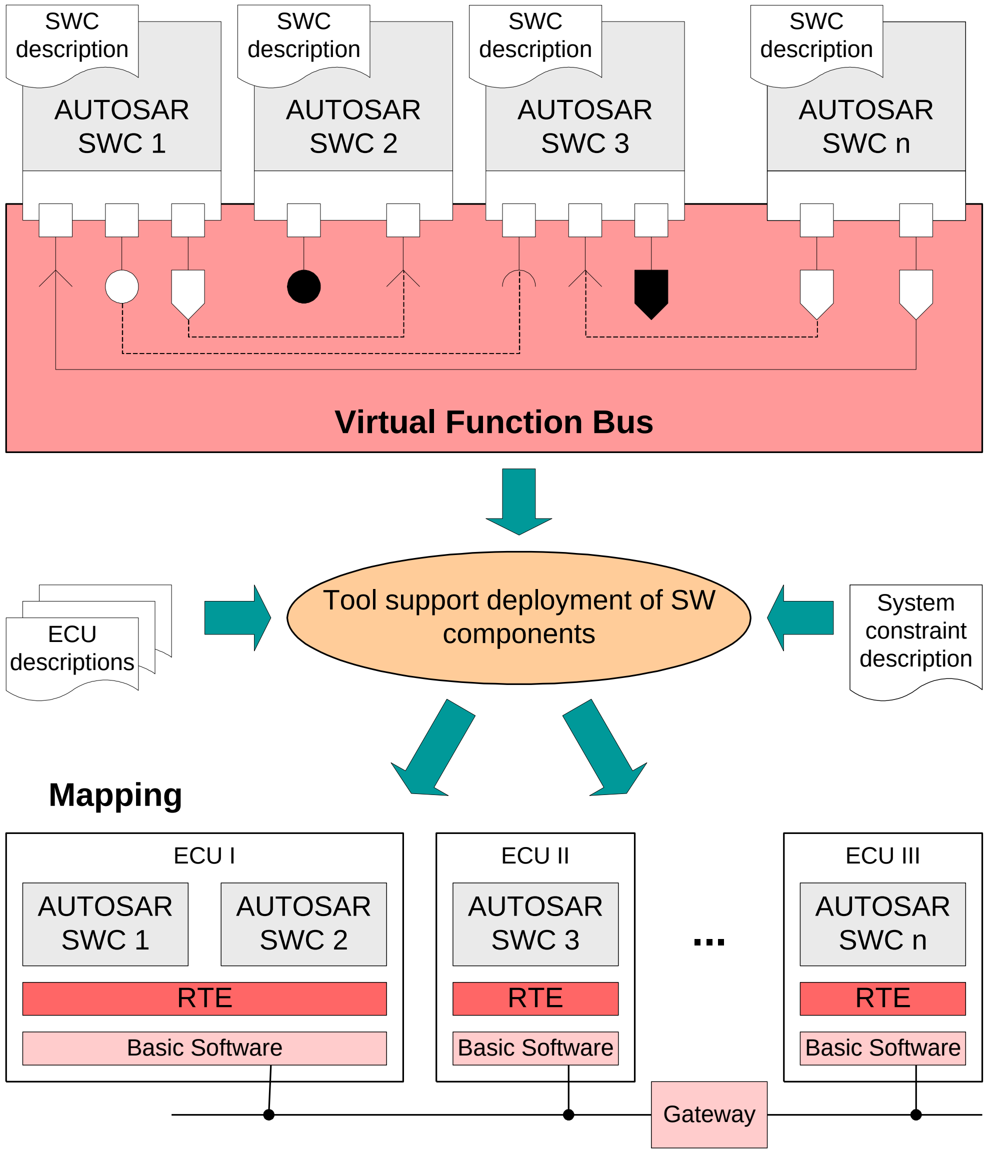


Figure 1: AUTOSAR software component (SWC) communications are represented by a virtual function bus (VFB) implemented using the runtime environment (RTE) and basic software.

2.2 What is an AUTOSAR Authoring Tool?

An AUTOSAR authoring tool is a software tool that supports interpreting, processing and creating of AUTOSAR descriptions:

- *Software Component descriptions* for the following items:
  - the operations and data elements that the software component provides and requires
  - the requirements which the software component has on the infrastructure

- the resources needed by the software component (memory, CPU-time, etc.)
- information regarding the specific implementation of the software component

– *System constraint descriptions* for all system information and the information that must be agreed between different ECUs

– *ECU descriptions* for the resources and configuration of the single ECUs

AUTOSAR SWCs are generic application-level components that are designed to be independent of both CPU and location in the vehicle network. An SWC can be mapped to any available ECU during system configuration, subject to constraints imposed by the system designer. An AUTOSAR software component is therefore the atomic unit of distribution in an AUTOSAR system; it must be mapped completely onto one ECU.

Before an SWC can be created, its component type (SWC type) must be defined. The SWC type identifies fixed characteristics of an SWC, i.e. port names, how ports are typed by interfaces, how the SWC behaves, etc. The SWC type is named, and the name must be unique within the system. Thus, an SWC consists of the following components:

– a complete formal SWC description that indicates how the infrastructure of the component must be configured,

– an SWC implementation that contains the functionality (in the form of C code)

To allow an SWC to be used, it needs to be instantiated at configuration time. The distinction between type and instance is analogous to types and variables in conventional programming languages. You define an application-wide unique type name (SWC type), and declare one uniquely named variable of that type (one or more SWC instance).

In the VFB model, software components interact through ports, which are typed by interfaces. The interface controls what can be communicated, as well as the semantics of communication. The port provides the SWC access to the interface. The combination of port and port interface is named *AUTOSAR interface*.

A *runnable entity* is a piece of code in an SWC that is triggered by the RTE (see section 2.3, *What is a Runtime Environment?*, on page 14) at runtime.

A software component comprises one or more runnable entities the RTE can access at runtime. Runnable entities are triggered, among others, by the following events:

– *Timing events* represent some periodic scheduling event, e.g., a periodic timer tick. The runnable entity provides the entry point for regular execution.

– Events triggered by the reception of data at an Rport (*DataReceive events*).

AUTOSAR runnable entities can be sorted in several categories. ASCET supports runnable entities of category 1.

In order to be executed, runnable entities must be assigned to the tasks of an AUTOSAR operating system.

AUTOSAR elements reference each other in a standardized XML file format, the so-called *ARXML format*. The ARXML format can slightly differ depending on the AUTOSAR release version. AUTOSAR authoring tools are required to be able to interpret, create, or modify ARXML descriptions.

> **NOTE**
>
> By default, the ARXML examples provided in this user guide are generated using the AUTOSAR release version 4.0.2.
>
> Exceptions are labeled explicitly.

2.3 What is a Runtime Environment?

The VFB provides the abstraction that allows components to be reusable. The *runtime environment* (RTE) provides the mechanisms required to make the VFB abstraction work at runtime. The RTE is, therefore, in the simplest case, an implementation of the VFB. However, the RTE must provide the necessary interfacing and infrastructure to allow software components to

A. be implemented without reference to an ECU (the VFB model); and
B. be integrated with the ECU and the wider vehicle network once this is known (the Systems Integration model) without changing the application software itself.

More specifically, the RTE must do the following:

- Provide a communication infrastructure for software components.

  This includes both communication between software components on the same ECU (intra-ECU) and communication between software components on different ECUs (inter-ECU).

- Arrange for real-time scheduling of software components.

  This typically means that the runnable entities of the SWCs are mapped, according to time constraints specified at design time, onto tasks provided by an operating system.

Application software components have no direct access to the basic software below the abstraction implemented by the RTE. This means that components cannot, for example, directly access operating system or communication services. So, the RTE must present an abstraction over such services. It is essential that this abstraction remains unchanged, irrespective of the software components' location. All interaction between software components therefore happens through standardized RTE interface calls.

In addition, the RTE is used for the specific realization of a previously specified architecture consisting of SWCs on one or more ECUs. To make the RTE implementation efficient, the RTE implementation required for the architecture is determined at build time for each ECU. The standardized RTE interfaces are automatically implemented by an RTE generation tool that makes sure that the interface behaves in the correct way for the specified component interaction and the specified component allocation.

For example, if two software components reside on the same ECU, they can use internal ECU communication, but if one is moved to a different ECU, communication now needs to occur across the vehicle network.

From the application software component perspective, the generated RTE therefore encapsulates the differences in the basic software of the various ECUs by:

- Presenting a consistent interface to the software components so they can be reused – they can be designed and written once but used multiple times.
- Binding that interface onto the underlying AUTOSAR basic software implemented in the VFB design abstraction.

2.4 What is a Behavior Modeling Tool?

An AUTOSAR Behavior Modeling Tool is a software tool that allows defining and implementing the functional behavior of AUTOSAR-compliant vehicle functions using a behavior modeling language.

A behavior modeling language is a notation primarily used to capture a functional behavior specification or design of a function or system. Usually, a functional behavior modeling language has a graphical notation and is regarded to be executable, i.e. its semantics is sufficiently precise to execute functional behavior models by means of a simulation engine. Furthermore, the precision in its semantics then allows the transformation of the functional model into a source code in a programming language like C.

When ASCET is used as a behavioral modeling tool, the internal behavior of the application software components is specified by means of the block diagram editor. The internal behavior can consist of variables, parameters, class instances and modules. AUTOSAR runnable entities can be seamlessly implemented by means of sequences of methods calls and processes.

Existing ASCET models can be easily adapted to AUTOSAR because many AUTOSAR concepts can be mapped to interface specifications in ASCET in a similar form. On the whole, it suffices to rework the interface of the respective application to make an existing software module AUTOSAR-compliant. In terms of time, the expenditure of reworking an existing application is relatively minor.

3 Developing Software Components in ASCET

The following products are required to use the AUTOSAR features of the current ASCET version:

- ASCET-MD
- ASCET-SE
- RTA-RTE (not part of the ASCET product family, see www.etas.com/en/products/rta_rte.php for further information)

> **NOTE**
>
> ASCET V6.4 supports the AUTOSAR releases R4.0.2, R4.0.3, R4.2.2, R4.3.0, R4.3.1.

Sample Database

The database `AUTOSAR_UG_Tutorial`[1] is provided with the ASCET installation. The examples depicted in this document are modeled in the `Solutions` folder. The corresponding ASCET-generated code can be found on the Windows file system, in the subdirectory `generated code_Solutions` contained in the database.

Finding Out More

This user guide is available electronically and can be viewed on the screen at any time. Using the index, full-text search, and hypertext links, you can find references fast and conveniently.

More detailed information on the AUTOSAR features of ASCET is given in the ASCET online help, sections "Software Component Editor" and "AUTOSAR Interfaces".

The following related documents are installed with the respective software:

- ASCET Getting Started manual (`ASCET V6.4 Getting Started.pdf`)
- ASCET-SE User Guide (`ASCET-SE V6.4 Users Guide.pdf`)
- RTA-RTE User Guide and other RTA-RTE documentation (available via the Windows **Start** menu, **E > ETAS RTA-RTE *<x.y>* > *<document>***)

These documents are also available in the Download Center of the ETAS website.

3.1 Configuring ASCET

This section briefly describes how to configure ASCET for developing AUTOSAR software components. For a more detailed description on how to work with ASCET, please refer to the ASCET Getting Started manual and the ASCET online help.

[1] located in the `Database` folder of the ASCET data path (selected during installation)

3.1.1 Configuring the Creation of AUTOSAR Components

ASCET offers the possibility to configure user profiles. In the context of AUTOSAR, ASCET provides a configuration option for the creation of AUTOSAR components.

To enable the creation of AUTOSAR components:

1. In the ASCET component manager, select **Tools › Options**.

   The "Options" dialog window opens.

2. In the "Modeling" node, make sure that the **Enable Creation of AUTOSAR components** option is activated.
3. Click **OK**.

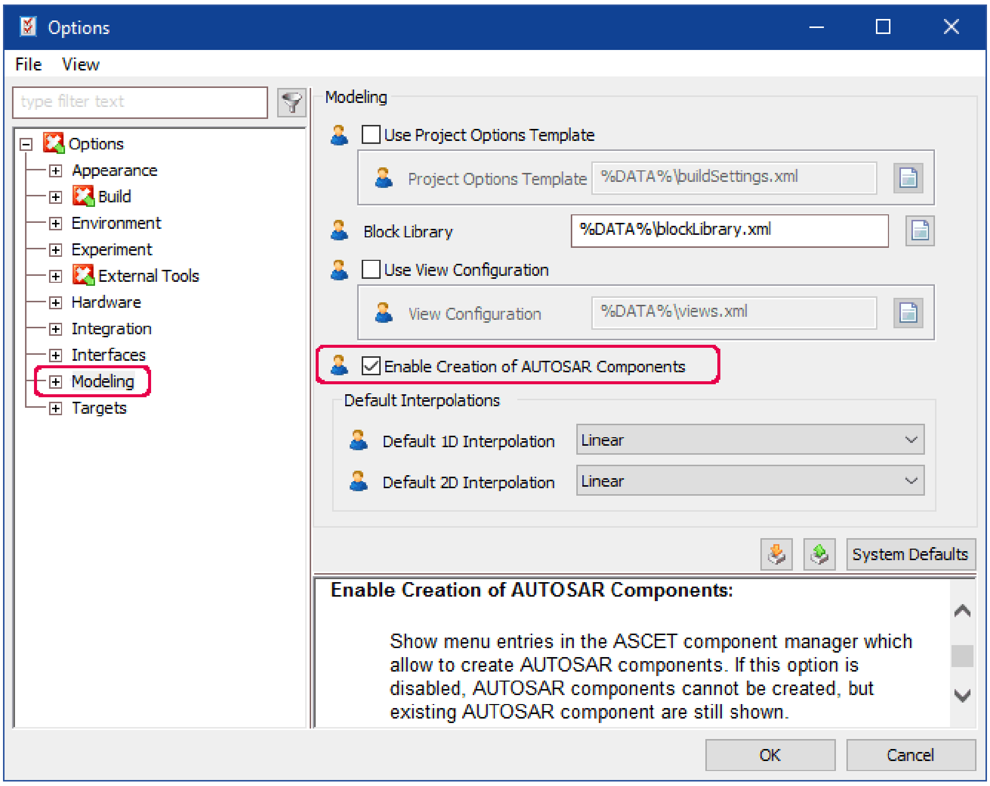


Figure 2: Enable creation of AUTOSAR components

3.1.2 Code Generation Settings for AUTOSAR

A project is the main unit in ASCET representing a complete software system. Formulas, implementation types etc. are defined within the context of a project.

To create a project

1. In the component manager, select **Insert › Project** or click the **Insert Project** button to add a new project.
2. Name the project `ARProject`.
3. Select **Edit › Open Component** or double-click the project.

   The project editor opens.

To set the code generation settings for AUTOSAR

1. In the project editor, select **File › Properties** or click the **Project Properties** button.

   The "Project Properties" dialog window opens.

2. In the "Build" node, select the following options:

   - Target: `ANSI-C`

     > **NOTE**
     >
     > Since ASCET V6.3, the `ANSI-C` target is the only target that can be used for AUTOSAR code generation.

   - Operating System: `RTE-AUTOSAR x.y.z`

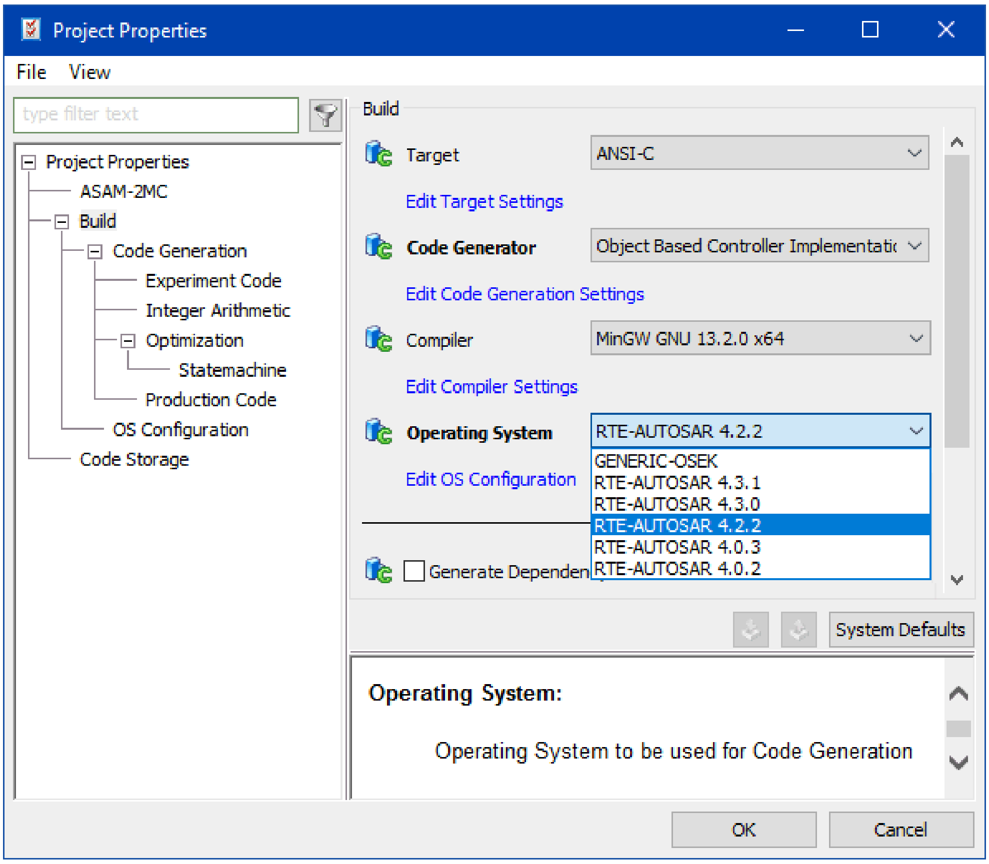


Figure 3: Project settings for AUTOSAR projects

> **NOTE**
>
> ASCET V6.4 supports the AUTOSAR releases 4.0.2, 4.0.3, R4.2.2, R4.3.0, R4.3.1.

3. If your SWC directly or indirectly contains a matrix, set up the ANSI-C target as follows:

   i. Follow the **Edit Target Settings** link.

      The ASCET options window opens in the “Targets\ANSI-C\Build” node.

ii. In that node, set the “Number of dimensions for fixed matrices” option to `Two-dimensional`.

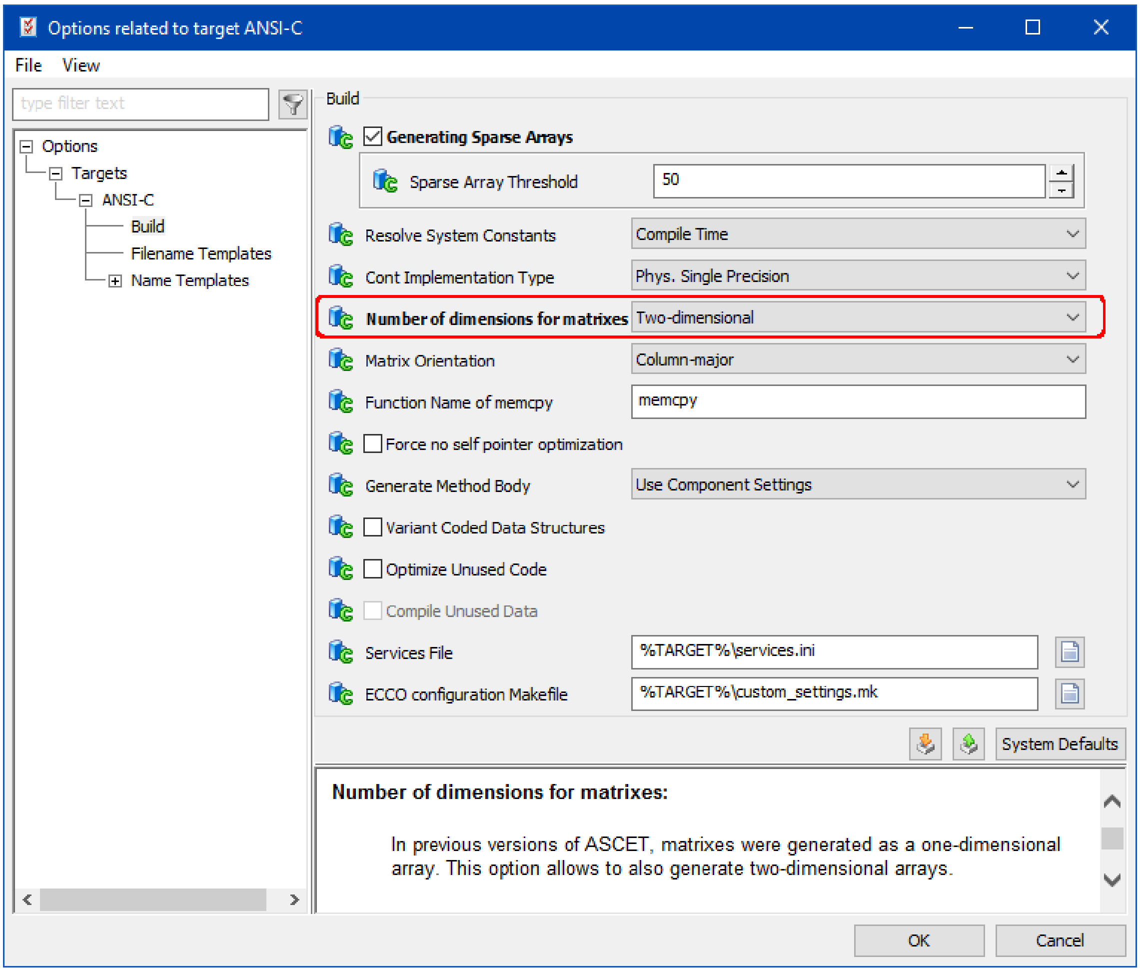


Figure 4: Matrices in AUTOSAR: target settings for the ANSI-C target

iii. Close the ASCET options window with **OK**.

4. In the "Code Generation" node, select the `MISRA compliant` casting strategy from the **Casting** combo box.

   Other casting strategies are not recommended.

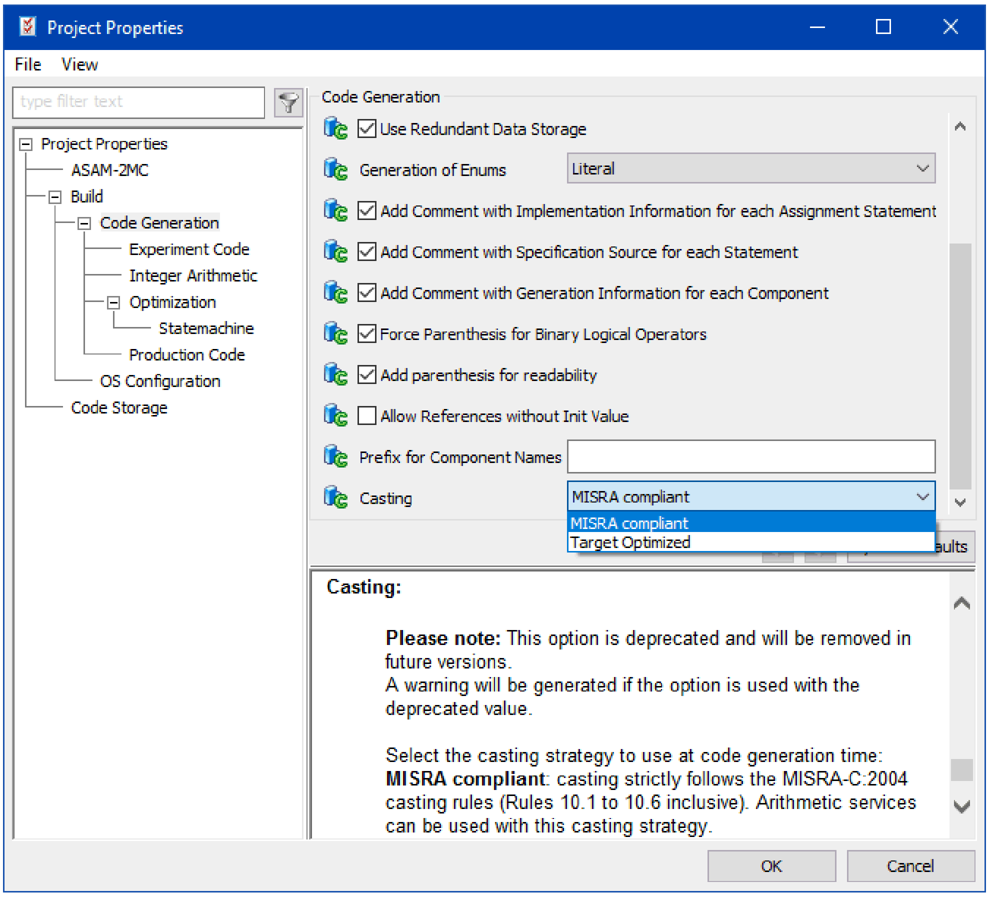


Figure 5: MISRA compliant casting for AUTOSAR projects

To define a memory sections definition file

When generating code in an AUTOSAR project, ASCET loads the memory sections from an XML configuration file. This file is defined in the project properties, "OS Configuration" node; see Figure 6.

1. Go to the "OS Configuration" node of the "Project Properties" dialog window.
2. In the "Memory Sections Configuration File" field, enter or select (via the button) path and name of the XML file that contains your memory sections definition.

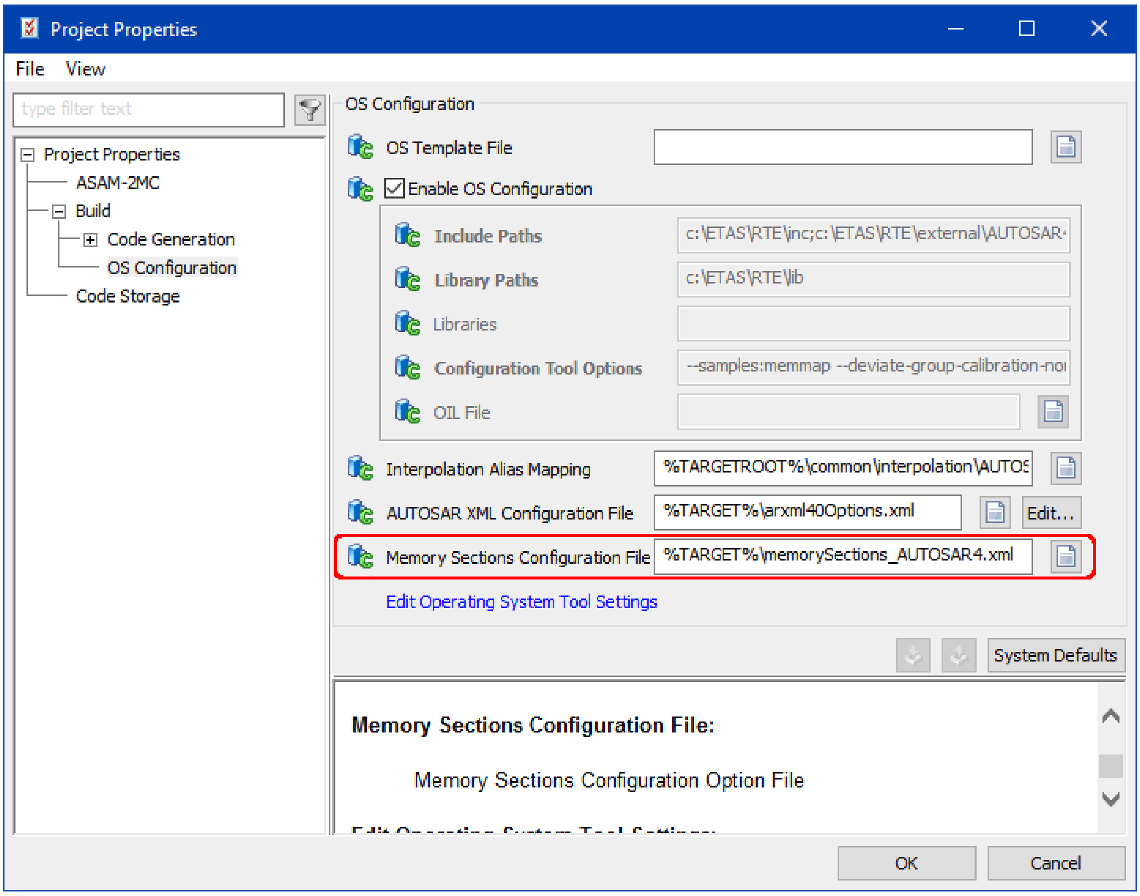


Figure 6: OS Configuration settings for an AUTOSAR R4.* project

ASCET provides a sample file, `memorySections_Autosar4.xml` (AUTOSAR R4.*). This file is preselected.

3.1.3 Settings for the AUTOSAR XML Output

The "Project Properties" window offers a possibility to configure the AUTOSAR XML output, i.e. to set package names or short names, to specify output files, etc.

To configure the AUTOSAR XML (ARXML) output

1. In the "Project Properties" dialog window, go to the "OS Configuration" node.
2. In the "AUTOSAR XML Configuration File" field, enter or select the configuration file.

   By default, each AUTOSAR Rx.y version uses a separate configuration file. It is recommended that you do not change this behavior because different AUTOSAR versions allow different ARXML settings.
3. Click the **Edit** button to open the "ARXML Configuration Settings" dialog window.

   This window provides a set of options to configure the AUTOSAR XML generation. The options are grouped in several categories; see Table 1.

Category	Content
Package Templates	Each package template allows the specification of an ARXML package name following the scheme: `/<Root-Package>/<Sub-Package>/.../<Short-Name>` Specific template parameters can be used.
Short Name Templates	Each short name template allows the specification of an ARXML short name. Specific template parameters can be used.
Filename Templates	Each filename template allows the specification of a filename where the associated package will be generated into. Specific template parameters can be used.
Miscellaneous	Each miscellaneous template or option represents an additional option somehow relevant for the ARXML generation. Among these options are the following: • **Generate System Constants**, which is required for variant handling (see section 7.11 on page 156). ▪ **Use Imported ARXML Info**, which is used to determine the way ASCET uses the usage information from imported ARXML files (see section 3.2.1.2 on page 25). The miscellaneous options and templates are described in the ASCET online help.

Table 1: Categories for the configuration of generated ARXML code. The content of the categories depends on the selected AUTOSAR version.

4. Adjust the options in the different categories according to your needs.

   The descriptions in the "ARXML Configuration Settings" dialog window contain detailed information on each option.

5. Click **OK** to confirm the settings and close the "ARXML Configuration Settings" dialog window.

> **NOTE**
>
> The changes in the "ARXML Configuration Settings" window are kept even if you leave the "Project Properties" window with **Cancel**.

3.1.4 Code Generation

An AUTOSAR project shall contain an AUTOSAR software component and requires the project settings described in the previous section. When generating code for the project, ASCET creates the AUTOSAR XML description files (`*.arxml` files) and the corresponding C code. The generated C code uses the AUTOSAR API macros, which are implemented in the RTE.

To create an AUTOSAR software component

1. In the component manager, select **Insert > AUTOSAR > Software Component Block Diagram** or **Software Component ESDL**.
2. Name the software component `Swc`.

To insert an AUTOSAR software component in a project

1. Open the project `ARProject` in the project editor.
2. In the project editor, select **Insert > Component**.
3. The "Select Item..." window opens.
4. In the „Database" or "Workspace" field of the "Select Item..." window, select the component `Swc`.

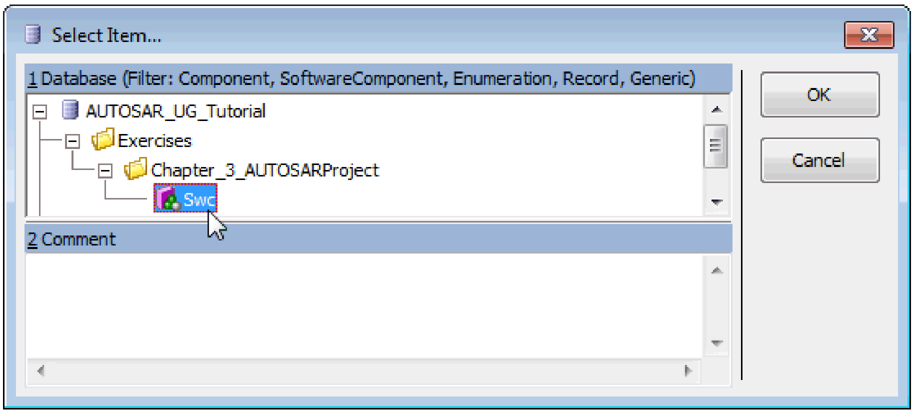


Figure 7: Select item `Swc` in the project `ARProject`

5. Click **OK** to close the "Select Item..." window and insert `Swc` into the project.

   The "Properties for Complex Element" window opens. You can enter a name and a comment for the `Swc` instance.
6. Click **OK** to use the default name and comment.

To generate code in a project

1. In the project editor, first select **Build > Touch > Recursive**, then select **File > Export > Generated Code > Recursive**.

   The "Path to export Items" window opens. The ASCET code generation directory, `Cgen`, is preselected.

   NOTE

   The `CGen` directory in the ASCET installation is a temporary directory that contains intermediate results from the code generator. It is not recommended to store code in this directory.
2. Select a destination folder to export the generated code.

   You may use, e.g., a subdirectory of the current ASCET database `C:\ETASData\ASCET6.4\Database\AUTOSAR_UG_Tutorial`.

   For the `ARProject` containing the empty AUTOSAR software component `Swc`, the following files are generated.

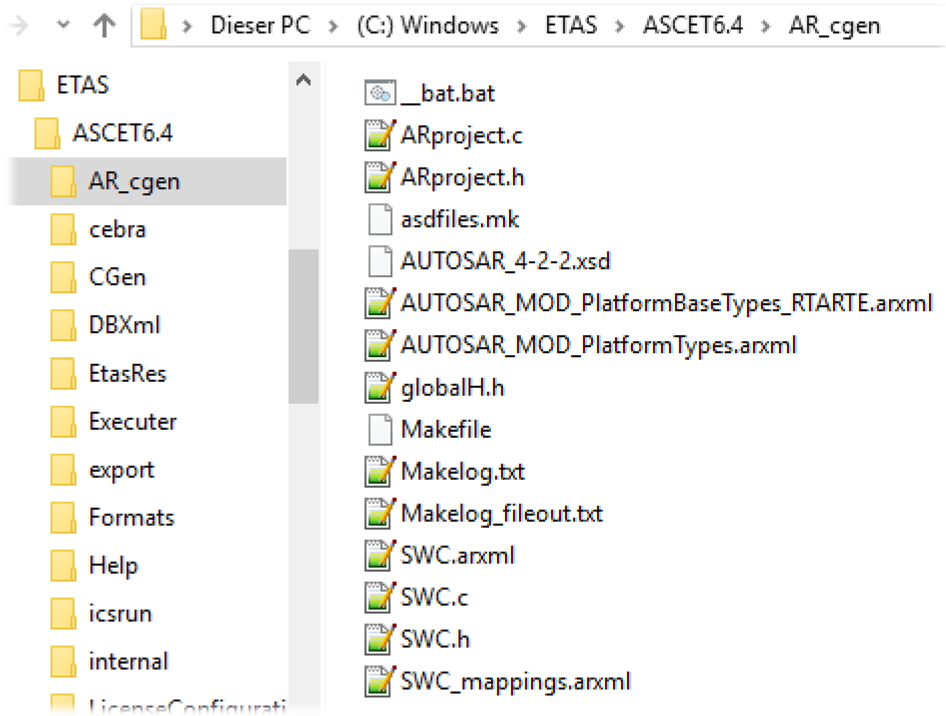


Figure 8: ASCET-generated AUTOSAR code for the project `ARProject` (`*.arxml`, `*.c`, and `*.h` files) in a non-default directory; AUTOSAR R4.2.2

3.2 Approaches for Creating Software Components

The development of AUTOSAR software components in ASCET can be done using two approaches: the top-down approach and the bottom-up approach.

In the top-down approach, the software architecture is described in an authoring tool. In this case, ASCET is used as a behavior modeling tool for the implementation of the software components.

In the bottom-up approach, ASCET is not only used as a behavior modeling tool, but as an authoring tool for the description of the AUTOSAR software components as well.

3.2.1 Top-Down Approach

In the top-down approach, the creation of an AUTOSAR software component is done in two steps:

A. In the first step, the interface of the component is defined. The interface is specified in an authoring tool and exchanged via ARXML. The ARXML files are then given to a component API generator, which transforms the interface description into a header file. As a rule, the component API generator is the contract phase part of an RTE generator (see section 3.3.1, *Contract Phase*, on page 29).

B. In a second step, the ARXML files are imported in ASCET, and the application software component developer provides the internal behavior in terms of C files respecting and using the interfaces as defined in the header file. Now the `*.h` and the `*.c` files of the software components are defined and can be compiled.

In the top-down approach, a key feature is the ARXML importer, which is described in the next subsections.

3.2.1.1 ARXML Importer

The ARXML description of a software component can be imported into ASCET with the "*AUTOSAR to ASCET Importer*". This tool transforms the ARXML file(s) containing all necessary information to describe a software component (i.e. AUTOSAR types, interfaces, software component type) into the proprietary ASCET XML format, the *AMD format*. Afterwards, ASCET imports the AMD files into the currently open database or workspace.

The AUTOSAR to ASCET Importer is started from the component manager, with the **Tools > AUTOSAR to ASCET Converter** menu option. See the AUTOSAR to ASCET Importer User Guide for details.

This is especially useful if the ARXML files contain multiple software components, and only one of them shall be imported.

In addition, ARXML file(s) can be imported using the standard import menu option.

To import an ARXML file into ASCET

1. In the component manager, select **File > Import**.

   The "Select Import File" dialog opens.

2. Select the ARXML file(s) to be imported and click **OK**.

   ASCET imports the selected files in the currently open database or workspace.

3.2.1.2 Usage Information from Imported ARXML Files

The ARXML files contain usage information for the interface elements in the runnables. Usage information means, e.g., access information for data elements and operation usage information.

Access information for a data element is stored using one of the following keywords, with additional information on the port and the data element name.

`DATA-READ-ACCESS`, `DATA-RECEIVE-POINT-BY-ARGUMENT`, `DATA-RECEIVE-POINT-BY-VALUE`, `DATA-SEND-POINT`, `DATA-WRITE-ACCESS`, `PARAMETER-ACCESS`

Operation usage information is stored using one of the following keywords, with additional information on the port and the operation name.

`ASYNCHRONOUS-SERVER-CALL-RESULT-POINT`, `ASYNCHRONOUS-SERVER-CALL-POINT`, `SYNCHRONOUS-SERVER-CALL-POINT`

Prior to V6.4.7, this information was ignored by the ARXML importer. Beginning with V6.4.7, this information is imported into ASCET. You can use the **Use Imported ARXML Info** option to determine the way ASCET uses the usage information from the ARXML file; see Table 2.

Use Imported ARXML Info value	Action
`Ignore`	Usage information from the ARXML file is ignored.
`Match`	Usage information from the ARXML file is compared with access information derived from the model.
`DeriveAndMatch`	Usage information from the ARXML file decides about access semantics prior to comparing it to the information derived from the model.

Table 2: Possible settings for the **Use Imported ARXML Info** ARXML configuration option

If you set the ARXML configuration option **Use Imported ARXML Info** to `Match`, usage information from the imported ARXML files is compared on a per-runnable basis with usage information derived from the model.

If you set **Use Imported ARXML Info** to `DeriveAndMatch`, the ARXML usage information is used to determine the access semantics, and an information message `IMdl533` is issued. The result is then compared in the same way as for `Match`.

If **Use Imported ARXML Info** is set to `Match` or `DeriveAndMatch`, and **Generate System Constants** is activated, too, usage conditions from the imported ARXML files are compared with usage conditions derived from the model.

If no match is detected, the following happens:

- If data access information is found in the ARXML usage information, but the corresponding information in the model-derived usage information is different, or completely missing, a warning `WIle533` is issued.
- If operation usage information is found in the ARXML usage information, but the corresponding information in the model-derived usage information is different, or completely missing, a warning `WIle534` is issued.
- If data access information is found in the model-derived usage information, but the corresponding information in the ARXML usage information is different, an error `MIle533` is issued.
- If operation usage information is found in the model-derived usage information, but the corresponding information in the ARXML usage information is different, an error `MIle534` is issued.
- If a usage condition is found in the ARXML file, but the corresponding usage condition derived from the model is different, a warning `WIle5351` is issued.
- If a usage condition is completely missing either in the imported ARXML file or in the model-derived information, `TRUE` is used instead, i.e. the element is used unconditionally.
- If no usage information was imported from the ARXML file, but **Use Imported ARXML Info** is set to `Match` or `DeriveAndMatch`, an error `MIle5331` or `MIle5341` is issued.

3.2.1.3 Using the Attribute UUID in the ARXML Import

UUIDs (Universally Unique Identifiers) are optional fields in the ARXML specification, and most authoring tools support them. ASCET also supports UUIDs in the AMD format, and this enables ASCET to be easily integrated in AUTOSAR toolchains. At present, the ASCET-generated ARXML provides a UUID for those elements that were imported with this attribute; otherwise, the attribute is empty.

UUIDs are mainly used for the identification of existing elements in the ASCET database or workspace when importing ARXML files. The use of the `UUID` attribute needs to be explicitly enabled.

To use UUIDs for identification

1. In the component manager, select **Tools > Options**.

   The ASCET options dialog window opens.
2. Go to the "Interfaces\Import" node.
3. Enable the option Use UUIDs for Identification.
4. Click **OK** to close the ASCET options window and accept the setting.

   The **Use UUIDs for Identification** option is also available in the "Select Import File" window, see Figure 9.

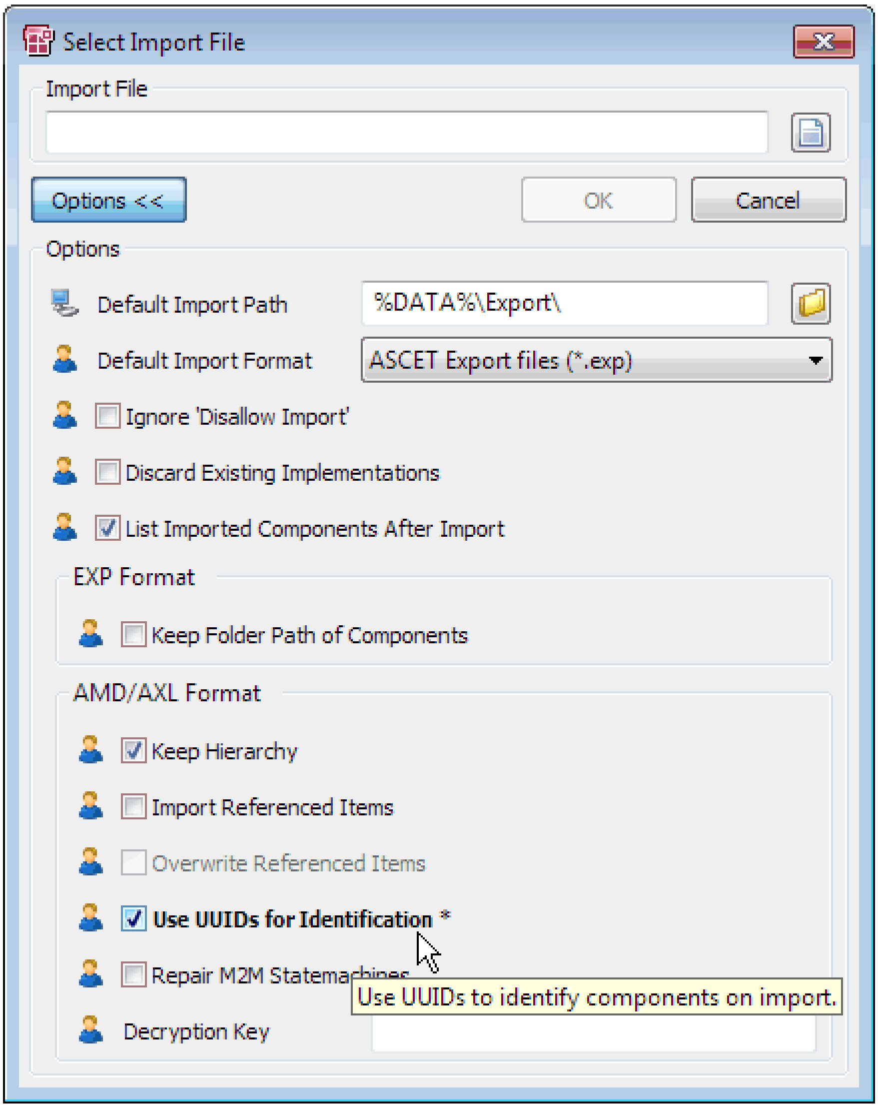


Figure 9: Using UUIDs to identify components on import

3.2.2 Bottom-Up Approach

For single application software components, ASCET can be used as authoring tool and behavior modeling tool. In the bottom-up approach, the AUTOSAR modeling elements supported in ASCET V6.4, i.e. Mode Group, Interface[2], Software Component, are created and maintained in the ASCET database/workspace.

See the ASCET online help for details on how to create and specify SWC, interfaces[2], and mode groups.

3.3 Working with the RTE Generator

The separation of the development and integration phases in AUTOSAR is reflected in a two-phase software component development process:

A. *Software Component Development*: the specification, design, and implementation of software components

[2] Sender-receiver, Client-server, Parameter (named Calibration Interface in ASCET), NVData

B. *Software Component Deployment*: the allocation of components to ECUs and the integration of components with the basic software on the ECU

The two phases of operation allow for initial software component configurations to be made and integrated onto the VFB (through some auxiliary design and development process) and then the RTE interface to be generated so that the software components can be implemented before the prototypes are defined and their particular allocations onto an ECU are known.

The phased development process means that some time can pass between the development of a component type and the allocation of its component prototypes to an ECU. Indeed, a component may be developed once and re-used multiple times over many generations of vehicles. Furthermore, the component may be supplied to an integrator in binary form only, but must be integrated to an ECU with other components that have not yet been written.

The RTE generator supports the phased process by allowing the interface to the RTE to be generated in advance of full knowledge of component prototype/ECU allocation. Given a software component description, the RTE generator has sufficient information to generate the interface definition files necessary for engineers to start developing software components. The interface defines the contract between the RTE and the component – what that component must provide if future integration work is to happen easily. This is known as *contract phase*.

When the system is integrated, and the mapping of software components to ECUs is known, the RTE itself can be generated. However, we now know how many instances of a software component exist, where runnable entities are executing, which communication is local to an ECU and which must be routed across the network, etc. The RTE generator can use this information to re-generate the interface definition files to include optimizations based on this additional context. This is known as *RTE phase*.

The following sections discuss the Contract and RTE phases in more detail.

3.3.1 Contract Phase

In the contract phase, the RTE generator produces header files to be used in the components under implementation. The header files define the contract between the component and the system as a whole and are suitable for both binary-code and source-code components. When running in the contract phase, the RTE generator only needs access to the software component description file(s). It is not necessary to have any information about system deployment.

The definitions in the ARXML file are used to define the APIs, and therefore only valid runnable entities can be declared without an error occurring when the component is compiled.

3.3.2 RTE Phase

Prior to using an RTE generator in RTE phase, a significant amount of system engineering is needed. The AUTOSAR development process assumes that there are a number of inputs to the system engineering process:

– *Software component descriptions* that define the software components, their ports, internal behavior and implementation characteristic, and the interfaces provided and required by the ports assuming their connection to the Virtual Function Bus. These are the same descriptions as used in contract phase.

- *ECU resource descriptions* that define the ECU hardware characteristics (e.g., communication ports)
- A *System constraint description* that defines aspects of the system (e.g., communication protocols)

To build an AUTOSAR system (i.e. a set of software components mapped to ECUs that communicate over a network) it is necessary to define the following:

- *ECU configuration descriptions* that define which software components are mapped to which ECUs, the resources available on the ECU, etc.
- A *System configuration description* that defines things like the network topology, how inter-ECU communication is mapped to the physical network etc.
- An *ECU Configuration* that defines the mapping between elements; for example, the mapping of runnable entities to AUTOSAR Operating System tasks and the mapping of AUTOSAR signals to AUTOSAR COM signals.

Once you have configured your AUTOSAR system with an allocation of component prototypes to ECU instances, the RTE generator is used in *RTE Generation* phase to create the following items:

A. the implementation of the RTE itself

B. optimized component header files that exploit mapping knowledge provided by your configuration

C. operating system tasks that package your runnable entities

D. (optional) an operating system configuration file for the RTE generated objects and required behavior

E. (optional) a communication stack configuration file for inter-ECU communication configuration

In the RTE phase, the RTE generates optimized application header files suitable for compiling source code components and, optionally, XML configuration files for the communication stack and operating system. When running the RTE phase, the RTE generator needs access to all system deployment information.

The RTE is generated as one or more C modules. Each module must be compiled according to the dependency information output by the RTE. The module `Rte.c` contains the core generated RTE.

4 Data Types

The types metamodel for AUTOSAR R4.* is a complete overhaul that replaces the former system. AUTOSAR R4.* defines three layers of data type abstraction as illustrated in Figure 10.

Application Data Types
Implementation Data Types
Base Types

Figure 10: AUTOSAR R4.* abstraction levels for describing data types

4.1 Application Data Types

Application data types are defined in physical terms. This allows application authors to create software components without deciding the C data type too early in the lifecycle.

Application data types contain the necessary information to support measurement and calibration tools.

Application data types also support automatic conversion of values from one unit to another.

The `<SHORT-NAME>` of an application data type is used within the scope of a software component type (SWCT), so it is possible to have multiple application data types with the same name when integrating several SWCTs on a single ECU (but not within a single SWCT).

The `<SHORT-NAME>` of an application data type is not used in generated code, in particular the RTE APIs are defined in terms of the mapped implementation data types.

To support more complex data types, an application data type can be composed of other application data types. This form of recursive definition permits records, arrays, and matrices to be defined.

When the RTE is generated, used application data types must be mapped to implementation types; see section 4.4, *Type Mappings*, on page 32 for details.

4.2 Implementation Data Types

Implementation data types represent C types in the generated code. The `<SHORT-NAME>` of an implementation data type defines the symbol used in C to access the type, e.g., in APIs and in user code.

In general, an implementation data type results in a `typedef` in the generated C code, written to the file `Rte_Type.h`. See the RTA-RTE user guide for information on the exceptions.

RTA-RTE always uses implementation data types in generated APIs. If the corresponding `<Variable-Data-Prototype>` is defined by reference to an application data type, then the mapped implementation data type is used in the API signature.

4.3 Base Types

Finally, *base types* describe the hardware-specific aspects of the data type, e.g., size and encoding. They form the basis on which the implementation data types are built. A base type can be referenced by several implementation data types (see section 4.2, *Implementation Data Types*, on page 31).

A base type's `<SHORT-NAME>` never appears in the generated code; it is only used as a reference target within the model. Only implementation data types are present in the generated code.

4.4 Type Mappings

An SWC-specific data type mapping is used to map application data types (cf. section 4.1) onto the implementing implementation data types (cf. section 4.2).

Mode type mappings are used to map mode declaration groups onto implementation types.

NOTE

RTA-RTE requires a data type mapping for each application type and a mode type mapping for each used mode declaration group in order to be able to generate the RTE.

In ASCET, these mappings are provided in the `Swc_mappings.arxml` file.

The data type mapping for a SWC is held within a `<DATA-TYPE-MAPPING-SET>` element:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>DataMappings</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
                    <SHORT-NAME>Impl</SHORT-NAME>
                    <ELEMENTS>
                        <DATA-TYPE-MAPPING-SET>
                            <SHORT-NAME>SWC</SHORT-NAME>
                            <DATA-TYPE-MAPS>
                                <Data-TYPE-MAP>
                                    ...
                                </Data-TYPE-MAP>
                                ...
                            </DATA-TYPE-MAPS>
                            <MODE-REQUEST-TYPE-MAPS>
                                <MODE-REQUEST-TYPE-MAP>
                                    ...
                                </MODE-REQUEST-TYPE-MAP>
                            </MODE-REQUEST-TYPE-MAPS>
                        </DATA-TYPE-MAPPING-SET>
                    </ELEMENTS>
                </AR-PACKAGE>
            </AR-PACKAGES>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 1: ARXML code – mapping application data types and mode type to implementation data types (AUTOSAR R4.2.2)

A data type mapping contains one or more data type maps. Each map references a single application data type and a single implementation data type; see Listing 3, Listing 7, Listing 10, or Listing 16 for ARXML examples.

For more information on data type and mode mapping, refer to the RTA-RTE user guide.

4.5 Platform Data Types

AUTOSAR specifies a set of *platform data types* for use in C code. These are implementation data types whose purpose is to provide a set of types with the same semantics across different target hardware. RTA-RTE uses platform data types when it needs to create types for internal variables.

Unlike most implementation data types, the platform data types are also defined in C language in the file `PlatformTypes.h`.

Beginning with R4.0.2, AUTOSAR also specifies the correct definitions and package name of the platform types. The platform types defined in the AUTOSAR "Specification of Platform Types" manual (`AUTOSAR_SWS_PlatformTypes.pdf`) and in the standard header file `Platform_Types.h` are:

- sint8 – 8-bit signed integer
- uint8 – 8-bit unsigned integer

- sint16 – 16-bit signed integer
- uint16 – 16-bit unsigned integer
- sint32 – 32-bit signed integer
- uint32 – 32-bit unsigned integer
- float32 – single precision floating point
- float64 – double precision floating point
- uint8_least – at least 8-bit unsigned integer
- uint16_least – at least 16-bit unsigned integer
- uint32_least – at least 32-bit unsigned integer
- sint8_least – at least 7-bit signed integer (plus sign bit)
- sint16_least – at least 15-bit signed integer (plus sign bit)
- sint32_least – at least 31-bit signed integer (plus sign bit)
- boolean – for use with TRUE/FALSE.

4.6 Primitive Data Types

The ASCET type system consists of model data types and implementation data types. Model data types are abstract generic data types that can be realized in one or more implementation data types.

The basic model data types for scalar elements are:

- Logic
- Limited Integer
- Wrap-Around Integer
- Signed Discrete
- Unsigned Discrete
- Continuous

All scalar elements in ASCET are implemented using one of the following data types:

- sint8
- sint16
- sint32
- uint8
- uint16
- uint32

Additionally, the model data type "cont" can be implemented as follows:

- real64
- real32

The model type "log" can also be implemented as follows:

- bool

To configure the default implementation of model types

1. In the component manager, select **Tools › Options**.

   The "Options" dialog window opens.
2. Open the "Modeling\Implementation\Default Implementation Types" node.
3. Configure the default implementation types, for instance, as shown below.

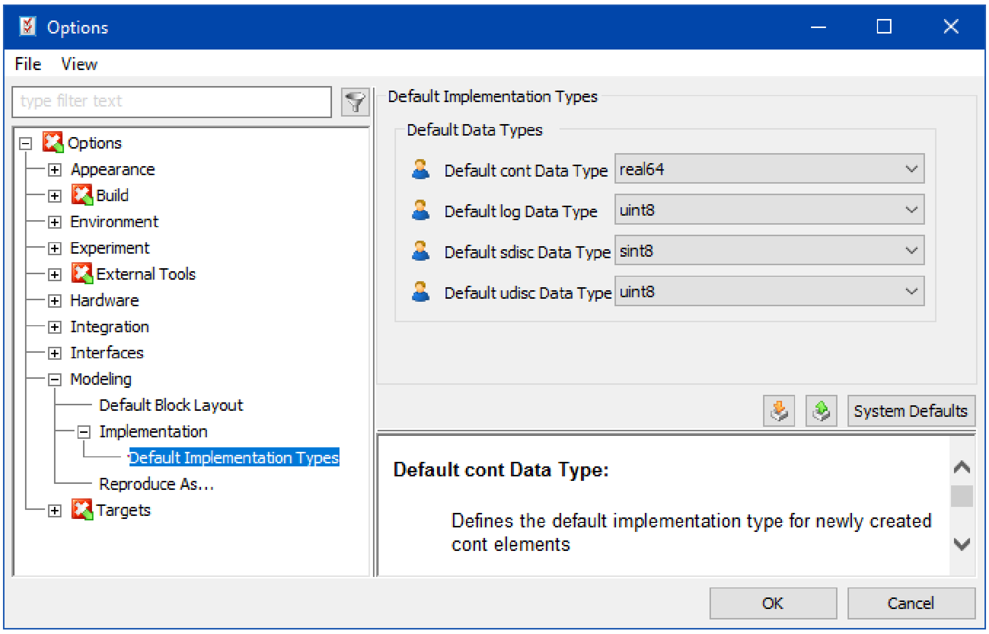


Figure 11: Default implementation of model types

4. Click **OK**.

The implementation of a model element can always be individually configured. The following instruction shows how to implement a variable `sdisc` as an 8 bit signed integer.

To implement a model type sdisc as a sint8

1. In the component manager, do one of the following:
   - Double-click the project `ARProject` created in section 3.1.2 *Code Generation Settings for AUTOSAR*.
   - Select the `ARProject` project and select **Edit › Open Component**.

   The project editor opens.
2. In the project editor, double-click the software component `Swc`.

   The software component editor opens.
3. Use the **Interrunnable Variable** button to create an interrunnable variable.

   The "Properties for Scalar Element" dialog window opens.
4. Do the following:

   i. Name the interrunnable variable `sdisc`.

   ii. Select the Basic Type Signed Discrete.

   iii. Close the properties editor with **OK**.

5. In the "Outline" tab, right-click the `sdisc` element and select **Implementation** from the context menu.

   The "Implementation for: sdisc" window opens.

6. In the "Master" field, activate **Implementation**.
7. In the "Type" combo box of the "Implementation" field, select `sint8`.
8. Close the "Implementation for: sdisc" window with **OK**.

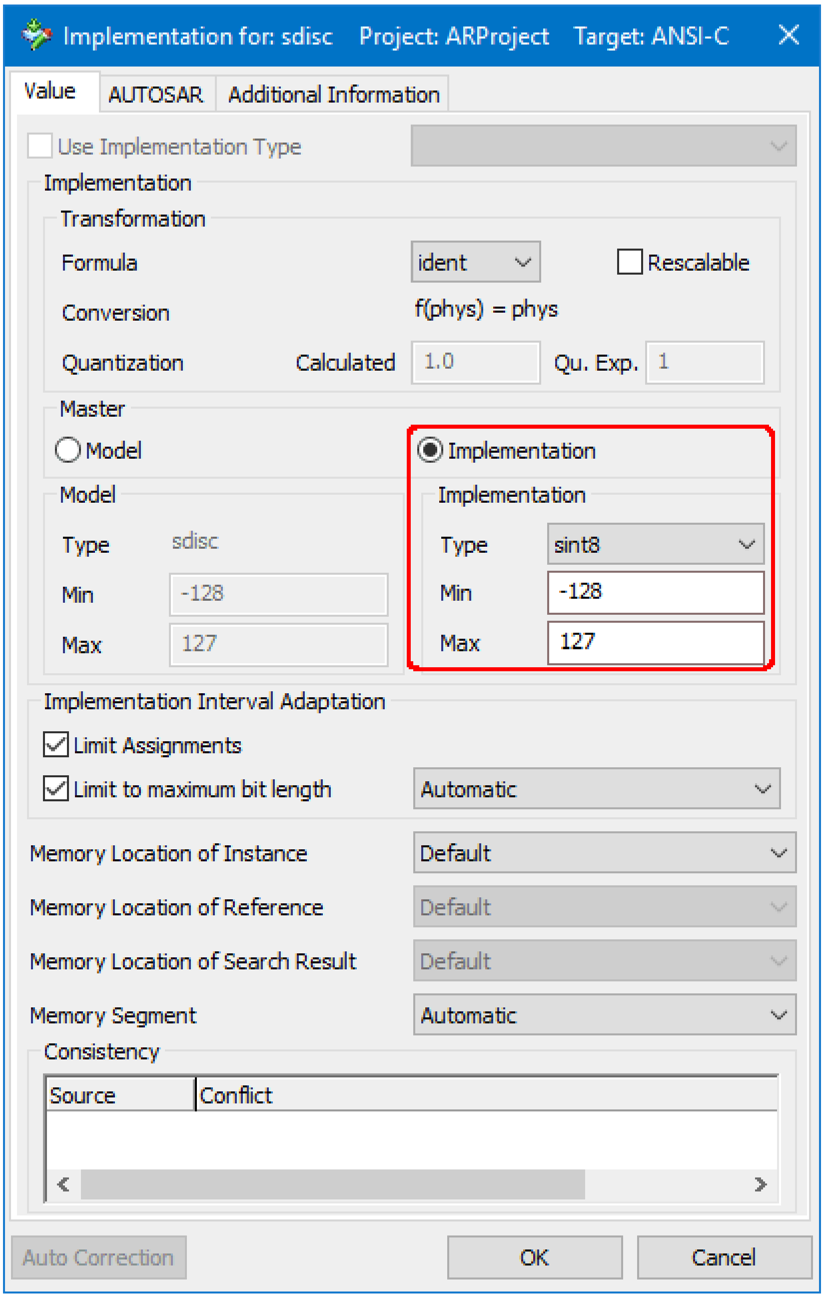


Figure 12: Implementation of the signed discrete element `sdisc` as `sint8`

When generating code for an AUTOSAR R4.* project, ASCET creates the files `Swc_appltypes.arxml` and `Swc_impltypes.arxml`, and copies the files

AUTOSAR_MOD_PlatformTypes.arxml and AUTOSAR_MOD_PlatformBaseTypes_RTARTE.arxml[3] to the code generation directory.

The following primitive application data type is defined in Swc_appltypes.arxml for the variable sdisc with sint8 implementation:

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <APPLICATION-PRIMITIVE-DATA-TYPE>
          <SHORT-NAME>SInt8</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>
                <COMPU-METHOD-REF DEST="COMPU-METHOD">
                /ASCET_CompuMethods/ident</COMPU-METHOD-REF>
                <DATA-CONSTR-REF DEST="DATA-CONSTR">
                /ASCET_DataConstrs/Physical/dc_SInt8_Phys
                </DATA-CONSTR-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </APPLICATION-PRIMITIVE-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 2: ARXML code – primitive application data type SInt8 (AUTOSAR R4.2.2)

In the file Swc_mappings.arxml, the application data type is mapped to an implementation data type:

[3] In older RTA-RTE versions: AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>DataMappings</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
                    <SHORT-NAME>Impl</SHORT-NAME>
                    <ELEMENTS>
                        <DATA-TYPE-MAPPING-SET>
                            <SHORT-NAME>Swc</SHORT-NAME>
                            <DATA-TYPE-MAPS>
                                ...
                                <DATA-TYPE-MAP>
                                    <APPLICATION-DATA-TYPE-REF DEST=
                                    "APPLICATION-PRIMITIVE-DATA-TYPE">
                                    /ASCET_Types/ApplicationDataTypes/SInt8
                                    </APPLICATION-DATA-TYPE-REF>
                                    <IMPLEMENTATION-DATA-TYPE-REF DEST=
                                    "IMPLEMENTATION-DATA-TYPE">
                                    /AUTOSAR_Platform/ImplementationDataTypes/sint8
                                    </IMPLEMENTATION-DATA-TYPE-REF>
                                </DATA-TYPE-MAP>
                                ...
                            <MODE-REQUEST-TYPE-MAPS>...</MODE-REQUEST-TYPE-MAPS>
                        </DATA-TYPE-MAPPING-SET>
                    </ELEMENTS>
                </AR-PACKAGE>
            </AR-PACKAGES>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 3: ARXML code – mapping of `SInt8` application data type and implementation data type (AUTOSAR R4.2.2)

The referenced implementation data type is a platform type, it does not appear in the `Swc_impltypes.arxml` file. In the `AUTOSAR_MOD_PlatformTypes.arxml` file, the referenced implementation data type looks as follows:

```
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>AUTOSAR_Platform</SHORT-NAME>
    ...
    <AR-PACKAGES>
      ...
      <AR-PACKAGE>
        <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
        <LONG-NAME>
          <L-4 L="EN">AUTOSAR Platform types</L-4>
        </LONG-NAME>
        <ELEMENTS>
          ...
          <IMPLEMENTATION-DATA-TYPE>
            <SHORT-NAME>sint8</SHORT-NAME>
            <LONG-NAME>
              <L-4 L="EN">signed integer 8bit</L-4>
            </LONG-NAME>
            <CATEGORY>VALUE</CATEGORY>
            <INTRODUCTION>
              <TRACE>
                <SHORT-NAME>PLATFORM016</SHORT-NAME>
                <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
                <P>
                  <L-1 L="EN">This standard AUTOSAR type shall be 8 bit
                  signed</L-1>
                </P>
              </TRACE>
            </INTRODUCTION>
            <SW-DATA-DEF-PROPS>
              <SW-DATA-DEF-PROPS-VARIANTS>
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                  <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                  /AUTOSAR_Platform/BaseTypes/sint8</BASE-TYPE-REF>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
              </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
          </IMPLEMENTATION-DATA-TYPE>
          ...
        </ELEMENTS>
      </AR-PACKAGE>
      ...
    </AR-PACKAGES>
  </AR-PACKAGE>
</AR-PACKAGES>
```

Listing 4: ARXML code – platform data type `sint8` (AUTOSAR R4.2.2)

The referenced base type is provided in the `AUTOSAR_MOD_PlatformBaseTypes_RTARTE.arxml`[4] file:

[4] In older RTA-RTE versions: `AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml`

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR_Platform</SHORT-NAME>
  <AR-PACKAGES>
    ...
    <AR-PACKAGE>
      <SHORT-NAME>BaseTypes</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AUTOSAR Base Types for AUTOSAR Platform types for TC1796</L-4>
      </LONG-NAME>
      <ELEMENTS>
        ...
        <SW-BASE-TYPE>
          <SHORT-NAME>sint8</SHORT-NAME>
          <LONG-NAME>
            <L-4 L="EN">signed integer 8bit</L-4>
          </LONG-NAME>
          <CATEGORY>FIXED_LENGTH</CATEGORY>
          <INTRODUCTION>
            <TRACE>
              <SHORT-NAME>PLATFORM016</SHORT-NAME>
              <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
              <P>
                <L-1 L="EN">This standard AUTOSAR type shall be 8 bit signed</L-1>
              </P>
            </TRACE>
          </INTRODUCTION>
          <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
          <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
        </SW-BASE-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 5: ARXML code – base type `sint8` (AUTOSAR R4.2.2)

The short-name of a data type must be a valid C identifier.

The types files are inputs for the RTE generator. The type definition for the user-defined primitive type is then included in the generated file `Rte_Type.h`. The implementation of the primitive types created by RTE references the BSW data types is defined for a particular micro-controller target by the AUTOSAR header file `Platform_Types.h`.

4.7 Primitive Data Types With Semantics

An additional data type in ASCET is Enumerations.

An enumeration in ASCET corresponds to an integer type with semantics. The semantic is given by a compu-method with category *Text Table*. A compu-method is a conversion formula from bit-pattern to physical value and vice versa.

To create an enumeration

1. In the component manager, select **Insert > Enumeration** or click the  **Enumeration** button.
2. Name the enumeration `Enumeration`.
3. Select the enumeration and switch to the "Contents" pane.
4. For the value 0, select **Enumeration > Rename** and set the label `red`.
5. Select **Enumeration > Add Enumeration > Append** or press the <INSERT> key to create two new enumerators with value 1 / label `yellow` and value 2 / label `green`.

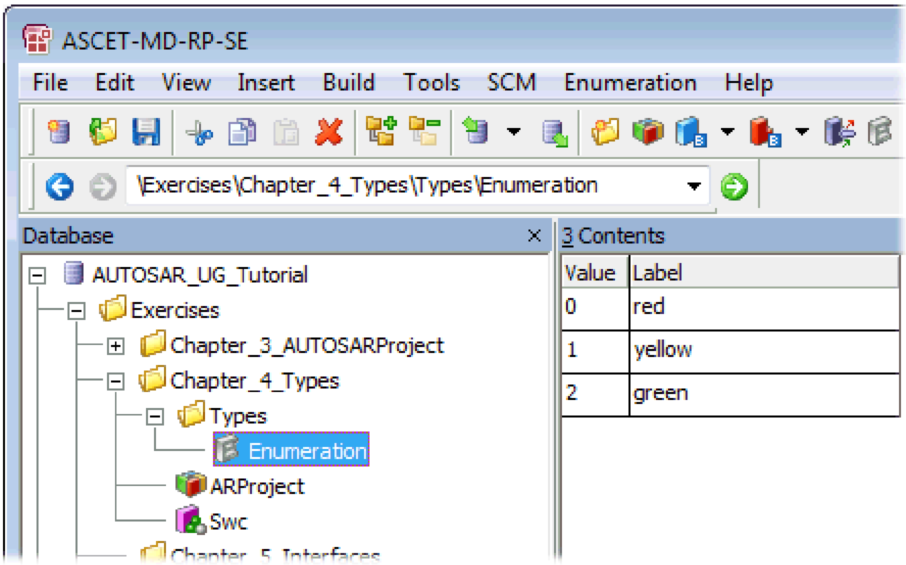


Figure 13: Example of an enumeration in ASCET

6. In the software component editor, create an interrunnable variable of basic type `Enumeration` and assign the enumeration you just created.

The definition of the data type and the compu-method in configuration language can be found in the AUTOSAR package `ASCET_Types`. The following application data type is defined in `Swc_appltypes.arxml` for the enumeration `Enumeration`:

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Types</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
         <ELEMENTS>
            ...
            <APPLICATION-PRIMITIVE-DATA-TYPE>
               <SHORT-NAME>Enumeration</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>
                        <DATA-CONSTR-REF DEST="DATA-CONSTR">
                        /ASCET_DataConstrs/Physical/dc_0to2</DATA-CONSTR-REF>
                     </SW-DATA-DEF-PROPS-CONDITIONAL>
                  </SW-DATA-DEF-PROPS-VARIANTS>
               </SW-DATA-DEF-PROPS>
            </APPLICATION-PRIMITIVE-DATA-TYPE>
            ...
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 6: ARXML code – application data type `Enumeration`

In the file `Swc_mappings.arxml`, the application data type is mapped to an implementation data type:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>DataMappings</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
                    <SHORT-NAME>Impl</SHORT-NAME>
                    <ELEMENTS>
                        <DATA-TYPE-MAPPING-SET>
                            <SHORT-NAME>SWC</SHORT-NAME>
                            <DATA-TYPE-MAPS>
                                ...
                                <DATA-TYPE-MAP>
                                    <APPLICATION-DATA-TYPE-REF DEST=
                                    "APPLICATION-PRIMITIVE-DATA-TYPE">
                                    /ASCET_Types/ApplicationDataTypes/Enumeration
                                    </APPLICATION-DATA-TYPE-REF>
                                    <IMPLEMENTATION-DATA-TYPE-REF DEST=
                                    "IMPLEMENTATION-DATA-TYPE">
                                    /ASCET_Types/ImplementationDataTypes/Enumeration
                                    </IMPLEMENTATION-DATA-TYPE-REF>
                                </DATA-TYPE-MAP>
                                ...
                            </DATA-TYPE-MAPS>
                            <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
                        </DATA-TYPE-MAPPING-SET>
                    </ELEMENTS>
                </AR-PACKAGE>
            </AR-PACKAGES>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 7: ARXML code – mapping of `Enumeration` application data type and implementation data type

The referenced implementation data type is not a platform data type, i.e. it appears in the `Swc_impltypes.arxml` file.

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Types</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
            <ELEMENTS>
                ...
                <IMPLEMENTATION-DATA-TYPE>
                    <SHORT-NAME>Enumeration</SHORT-NAME>
                    <CATEGORY>TYPE_REFERENCE</CATEGORY>
                    <SW-DATA-DEF-PROPS>
                        <SW-DATA-DEF-PROPS-VARIANTS>
                            <SW-DATA-DEF-PROPS-CONDITIONAL>
                                <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                                /AUTOSAR_Platform/ImplementationDataTypes/uint8
                                </IMPLEMENTATION-DATA-TYPE-REF>
                            </SW-DATA-DEF-PROPS-CONDITIONAL>
                        </SW-DATA-DEF-PROPS-VARIANTS>
                    </SW-DATA-DEF-PROPS>
                </IMPLEMENTATION-DATA-TYPE>
                ...
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 8: ARXML code – implementation data type `Enumeration`

The implementation data type references the `sint8` platform data type; see Listing 4 on page 39.

The `sint8` platform data type references the sint8 base type; see Listing 5 on page 40.

4.7.1 `Std_ReturnType`

The AUTOSAR standard defines "status" and "error" values returned by RTE API functions. The following values are defined in the `Std_ReturnType` type:

Code	Available in AUTOSAR Release	Meaning
`COM_BUSY`	≥ R4.2.2	The transmission/reception could not be performed due to another transmission/reception currently ongoing for the same signal.
`COM_STOPPED`	all supported	The RTE could not perform the operation because the communication service is currently not available (inter-ECU communication only).
`IN_EXCLUSIVE_AREA`	all supported	The error is returned by a blocking API and indicates that the runnable could not enter a wait state. This can happen, for example, because one executable entity of the current task's call stack has entered an exclusive area.
`INVALID`	all supported	data element is invalid
`LIMIT`	all supported	An internal RTE limit has been exceeded. Request could not be handled. OUT buffers are not modified.
`MAX_AGE_EXCEEDED`	all supported	data element is outdated Can be combined with other error codes.
`NEVER_RECEIVED`	all supported	No data received for the corresponding unqueued data element since system start or partition restart.
`NO_DATA`	all supported	An explicit read API call returned no data. (This is no error.)
`OK`	all supported	no error occurred
`OUT_OF_RANGE`	≥ R4.2.2	received data element out of range
`SEG_FAULT`	all supported	The error can be returned by an RTE API if the parameters contain a direct or indirect reference to memory that is not accessible from the caller's partition.
`HARD_TRANSFORMER_ERROR`	≥ R4.2.2	An error occurred during transformation.

Code	Available in AUTOSAR Release	Meaning
`TRANSFORMER_LIMIT`	≥ R4.2.2	Buffer for transformation operation could not be created.
`SOFT_TRANS-FORMER_ERROR`	≥ R4.2.2	An error occurred during transformation. The error is reported even though valid data is produced as output (comparable to a warning).
`TIMEOUT`	all supported	A blocking API call returned due to expiry of a local timeout rather than the intended result. OUT buffers are not modified. The interpretation of this being an error depends on the application.
`TRANSMIT_ACK`	all supported	transmission acknowledgment received
`UNCONNECTED`	all supported	The port used for communication is not connected.

NOTE

ASCET uses the error codes exactly as given in this table. The prefix **`RTE_E_`**, required to provide compliance to the AUTOSAR standard, is inserted during AUTOSAR code generation.

If you enter an error code with the prefix, e.g., `RTE_E_NO_DATA`, a warning (`Wmdl1192`) is issued during code generation:

```
Label <%1> for Rte_StdReturnType is deprecated, please use
<%2> instead
```

Table 3: AUTOSAR error codes

ASCET provides the `Std_ReturnType` type as a built-in enumeration. The error codes are reserved words in ASCET and cannot be used in other enumerations.

Furthermore, `OK` is also reserved in ASCET, which denotes that a server runnable returns no application error. The user shall specify – or import – the possible values of the application error in a standard enumeration.

4.8 Complex Data Types

4.8.1 Record Data Types

Record data types allow new complex data types to be created. A record data type creates a data structure consisting of one or more named members.

To create a record in ASCET

1. In the component manager, select **Insert > Record** or click the **Record** button.
2. Name the record `Record`.

3. Select **Edit › Open Component** or double-click the record.

   The record editor opens.

4. Use the **Unsigned Discrete Variable** button to create a `udisc` variable.

   The dialog "Properties for Scalar Element: udisc" opens.

5. Name the unsigned discrete variable `A`.

6. Use the **Logic Variable** button to create a `log` variable named `B`.

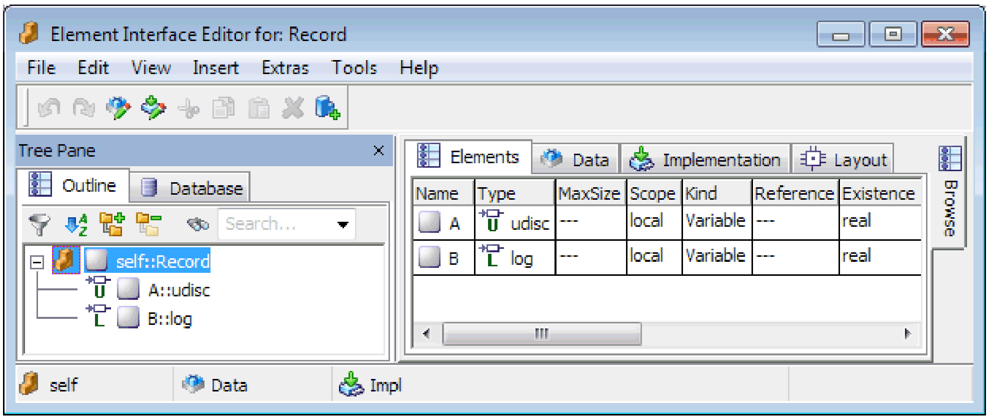


Figure 14: Record with elements `A` and `B`

<u>**To specify an implementation of a record**</u>

1. In the record editor, switch from the "Elements" tab to the "Implementation" tab.

2. In the "Implementation" tab, double-click the element `A`.

   The "Implementation for: A" window opens.

3. In the "Master" field, activate **Implementation**.

4. In the "Implementation" field, select `uint16`.

5. Right-click in the "Max" field and select **Default Value** from the context menu.

6. Close the "Implementation for: A" window with **OK**.

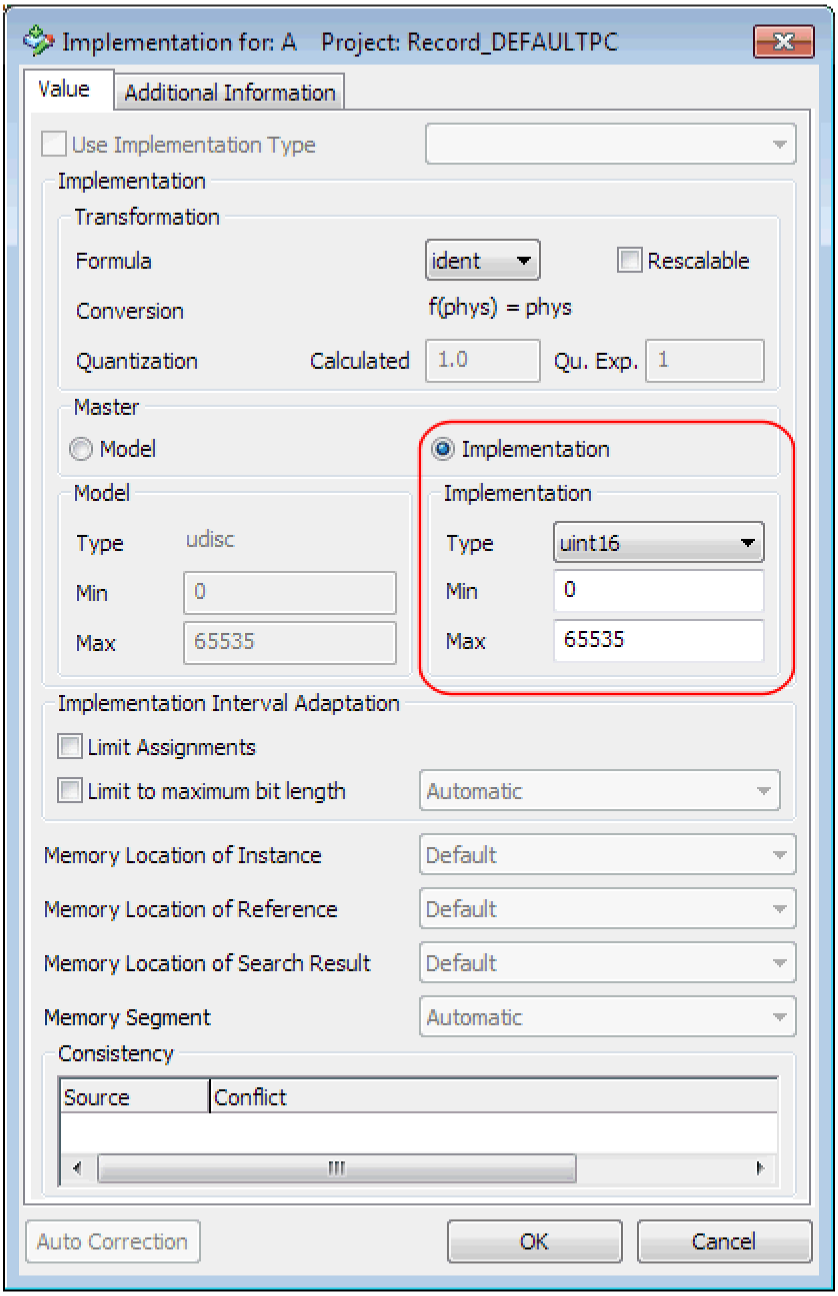


Figure 15: Implementation of the unsigned discrete element A as `uint16`

7. For the logic variable `B`, select the implementation type `bool`.

   The "Implementation" tab of the record editor looks like the figure below.

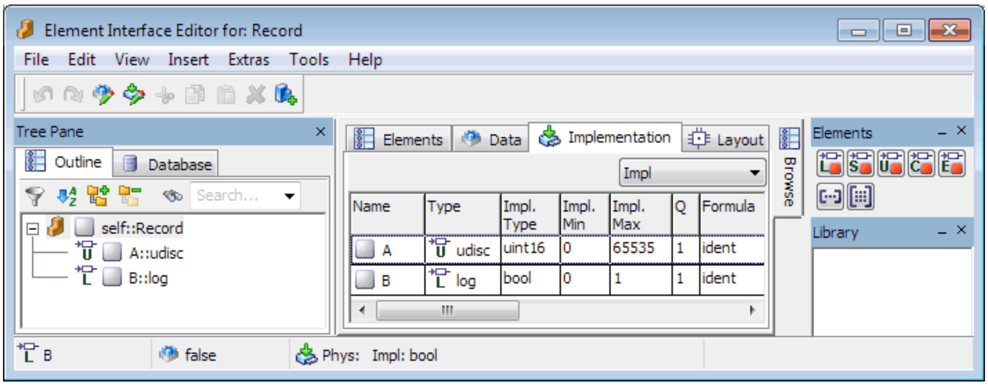


Figure 16: Implementation `Impl` of `Record` with elements `A` and `B`

An implementation of a record in ASCET corresponds to a record data type in AUTOSAR. The record data type in configuration language can be found in the AUTOSAR package `ASCET_types`. The RTE generator will generate a C structure type for each defined `<RECORD-TYPE>`. The structure definition is included in the generated file `Rte_Type.h`.

To create a new implementation of a record

1. In the record editor, select **Edit** › **Component** › **Implementation**.

   The "Implementation Editor for: Record" window opens.
2. Select **Implementation** › **Add** and name the new implementation, for instance, `Impl32`.
3. Set an implementation `uint32` with default min/max for `A`.
4. Set an implementation `bool` for `B`.
5. Click **OK**.

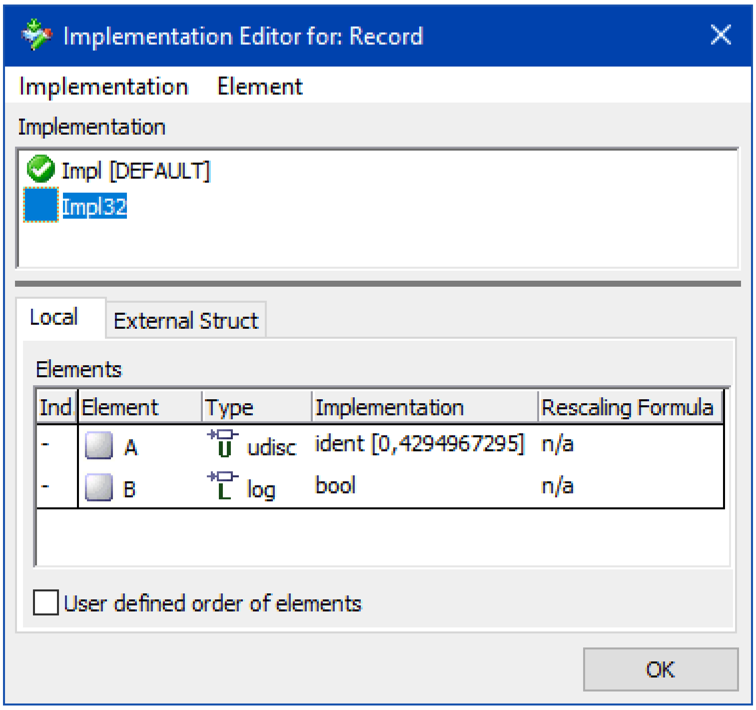


Figure 17: Record type `Record_Impl32`

Now insert the record as an interrunnable variable into the SWC and generate code[5] for the project.

The following application data type is defined in `Swc_appltypes.arxml` for the record type `Record_Impl`:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Types</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
            <ELEMENTS>
                ...
                <APPLICATION-RECORD-DATA-TYPE>
                    <SHORT-NAME>Record_Impl</SHORT-NAME>
                    <CATEGORY>STRUCTURE</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>
                    <ELEMENTS>
                        <APPLICATION-RECORD-ELEMENT>
                            <SHORT-NAME>A</SHORT-NAME>
                            <CATEGORY>VALUE</CATEGORY>
                            <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                            /ASCET_Types/ApplicationDataTypes/UInt16</TYPE-TREF>
                        </APPLICATION-RECORD-ELEMENT>
                        <APPLICATION-RECORD-ELEMENT>
                            <SHORT-NAME>B</SHORT-NAME>
                            <CATEGORY>VALUE</CATEGORY>
                            <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                            /ASCET_Types/ApplicationDataTypes/Boolean</TYPE-TREF>
                        </APPLICATION-RECORD-ELEMENT>
                    </ELEMENTS>
                </APPLICATION-RECORD-DATA-TYPE>
                ...
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 9: ARXML code – application data type `Record_Impl`

[5] see *To generate code in a project* on page 23

In the file `Swc_mappings.arxml`, the application data type `Record_Impl` is mapped to an implementation data type:

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>DataMappings</SHORT-NAME>
      <AR-PACKAGES>
        <AR-PACKAGE>
          <SHORT-NAME>Impl</SHORT-NAME>
          <ELEMENTS>
            <DATA-TYPE-MAPPING-SET>
              <SHORT-NAME>SWC</SHORT-NAME>
              <DATA-TYPE-MAPS>
                ...
                <DATA-TYPE-MAP>
                  <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-RECORD-DATA-TYPE">
                  /ASCET_Types/ApplicationDataTypes/Record_Impl
                  </APPLICATION-DATA-TYPE-REF>
                  <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                  /ASCET_Types/ImplementationDataTypes/Record_Impl
                  </IMPLEMENTATION-DATA-TYPE-REF>
                </DATA-TYPE-MAP>
                ...
              </DATA-TYPE-MAPS>
              <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
            </DATA-TYPE-MAPPING-SET>
          </ELEMENTS>
        </AR-PACKAGE>
      </AR-PACKAGES>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 10: ARXML code – mapping of `Record_Impl` application data type and implementation data type

The referenced implementation data type is not a platform data type, i.e. it appears in the `Swc_impltypes.arxml` file.

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Types</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
            <ELEMENTS>
                ...
                <IMPLEMENTATION-DATA-TYPE>
                    <SHORT-NAME>Record_Impl</SHORT-NAME>
                    <CATEGORY>STRUCTURE</CATEGORY>
                    <SUB-ELEMENTS>
                        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
                            <SHORT-NAME>A</SHORT-NAME>
                            <CATEGORY>TYPE_REFERENCE</CATEGORY>
                            <SW-DATA-DEF-PROPS>
                                <SW-DATA-DEF-PROPS-VARIANTS>
                                    <SW-DATA-DEF-PROPS-CONDITIONAL>
                                        <IMPLEMENTATION-DATA-TYPE-REF DEST=
                                        "IMPLEMENTATION-DATA-TYPE">
                                        /AUTOSAR_Platform/ImplementationDataTypes/uint16
                                        </IMPLEMENTATION-DATA-TYPE-REF>
                                    </SW-DATA-DEF-PROPS-CONDITIONAL>
                                </SW-DATA-DEF-PROPS-VARIANTS>
                            </SW-DATA-DEF-PROPS>
                        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
                        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
                            <SHORT-NAME>B</SHORT-NAME>
                            <CATEGORY>TYPE_REFERENCE</CATEGORY>
                            <SW-DATA-DEF-PROPS>
                                <SW-DATA-DEF-PROPS-VARIANTS>
                                    <SW-DATA-DEF-PROPS-CONDITIONAL>
                                        <IMPLEMENTATION-DATA-TYPE-REF DEST=
                                        "IMPLEMENTATION-DATA-TYPE">
                                        /AUTOSAR_Platform/ImplementationDataTypes/boolean
                                        </IMPLEMENTATION-DATA-TYPE-REF>
                                    </SW-DATA-DEF-PROPS-CONDITIONAL>
                                </SW-DATA-DEF-PROPS-VARIANTS>
                            </SW-DATA-DEF-PROPS>
                        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
                    </SUB-ELEMENTS>
                </IMPLEMENTATION-DATA-TYPE>
                ...
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 11: ARXML code – implementation data type `Record_Impl`

The implementation data type `Record_impl` references two platform data types, one for each record element.

In the `AUTOSAR_MOD_PlatformTypes.arxml` file, the referenced implementation data types look as follows (the `<INTRODUCTION>` elements may be non-empty in your generated code):

```
<IMPLEMENTATION-DATA-TYPE>
    <SHORT-NAME>boolean</SHORT-NAME>
    <LONG-NAME>
        <L-4 L="EN">Boolean</L-4>
    </LONG-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <INTRODUCTION>
        ...
    </INTRODUCTION>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_PlatformTypes/SwBaseTypes/boolean</BASE-TYPE-REF>
                <!-- CompuMethod for TRUE and FALSE -->
                <COMPU-METHOD-REF DEST="COMPU-METHOD">
                /AUTOSAR_PlatformTypes/CompuMethods/boolean</COMPU-METHOD-REF>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 12: ARXML code – platform data type `Boolean`

```
<IMPLEMENTATION-DATA-TYPE>
    <SHORT-NAME>uint16</SHORT-NAME>
    <LONG-NAME>
        <L-4 L="EN">unsigned integer 16bit</L-4>
    </LONG-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <INTRODUCTION>
        ...
    </INTRODUCTION>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_PlatformTypes/SwBaseTypes/uint16</BASE-TYPE-REF>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 13: ARXML code – platform data type `uint16`

The base types `boolean` and `uint16`, referenced in Listing 12 and Listing 13, are provided in the `AUTOSAR_MOD_PlatformBaseTypes_RTARTE.arxml`[6] file:

```
<SW-BASE-TYPE>
  <SHORT-NAME>boolean</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Boolean</L-4>
  </LONG-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM060</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">The boolean type shall always be mapped to a platform
        specific type where pointers can be applied to to enable a passing of
        parameters via API. There are specific BIT types of some HW platforms
        which are very efficient but where no pointers can point to.</L-1>
      </P>
    </TRACE>
    <TRACE>
      <SHORT-NAME>PLATFORM026</SHORT-NAME>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall only be used together with
        the definitions TRUE and FALSE. See PLATFORM027 for implementation and
        usage.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>BOOLEAN</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
...
<SW-BASE-TYPE>
  <SHORT-NAME>uint16</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">unsigned integer 16bit</L-4>
  </LONG-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM014</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall be of 16 bit unsigned.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <BASE-TYPE-SIZE>16</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
```

Listing 14: ARXML code – base types `boolean` and `uint16`

4.8.2 Array Data Types

Array data types, like record data types, allow new complex data types to be created. An array data type creates a sequence of values mapped to an index position.

To create an array

1. In the component manager, double-click the project `ARProject`.

   The project editor opens.

[6] In older RTA-RTE versions: `AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml`

2. In the project editor, double-click the software component `Swc`.

   The software component editor opens.

3. Use the **Array** button to create an array.

   The properties editor for the array opens.

4. Name the variable `array` and set the following properties:

Dimension X	16
Kind	Interrunnable Variable
Basic type	unsigned discrete

5. Close the properties editor with **OK**.
6. Open the implementation editor for `array`.
7. In the "Master" field, activate **Implementation**.
8. In the "Implementation" field, select `uint8`.
9. Close the "Implementation for: array" window with **OK**.

An implementation of an array in ASCET corresponds to an array data type in AUTOSAR. The array data type in configuration language can be found in the AUTOSAR package `ASCET_types`.

The following application data type is defined in `Swc_appltypes.arxml` for the array `array`:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Types</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
            <ELEMENTS>
                ...
                <APPLICATION-ARRAY-DATA-TYPE>
                    <SHORT-NAME>UInt8_16</SHORT-NAME>
                    <!--
                        array of 16 &quot;UInt8&quot; values
                    -->
                    <CATEGORY>ARRAY</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>
                    <ELEMENT>
                        <SHORT-NAME>ElementName</SHORT-NAME>
                        <CATEGORY>VALUE</CATEGORY>
                        <SW-DATA-DEF-PROPS>
                            <SW-DATA-DEF-PROPS-VARIANTS>
                                <SW-DATA-DEF-PROPS-CONDITIONAL>
                                </SW-DATA-DEF-PROPS-CONDITIONAL>
                            </SW-DATA-DEF-PROPS-VARIANTS>
                        </SW-DATA-DEF-PROPS>
                        <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                        /ASCET_Types/ApplicationDataTypes/UInt8</TYPE-TREF>
                        <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
                        <MAX-NUMBER-OF-ELEMENTS>16</MAX-NUMBER-OF-ELEMENTS>
                    </ELEMENT>
                </APPLICATION-ARRAY-DATA-TYPE>
                ...
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 15: ARXML code – application data type `UInt8_16` of category `ARRAY`

In the `Swc_mappings.arxml` file, the application data type is mapped to an implementation data type:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>DataMappings</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
                    <SHORT-NAME>Impl</SHORT-NAME>
                    <ELEMENTS>
                        <DATA-TYPE-MAPPING-SET>
                            <SHORT-NAME>SWC</SHORT-NAME>
                            <DATA-TYPE-MAPS>
                                ...
                                <DATA-TYPE-MAP>
                                    <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-ARRAY-DATA-TYPE">
                                    /ASCET_Types/ApplicationDataTypes/UInt8_16
                                    </APPLICATION-DATA-TYPE-REF>
                                    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                                    /ASCET_Types/ImplementationDataTypes/uint8_16
                                    </IMPLEMENTATION-DATA-TYPE-REF>
                                </DATA-TYPE-MAP>
                                ...
                            </DATA-TYPE-MAPS>
                            <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
                        </DATA-TYPE-MAPPING-SET>
                    </ELEMENTS>
                </AR-PACKAGE>
            </AR-PACKAGES>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 16: ARXML code – mapping of `UInt8_16` application data type and implementation data type

The referenced implementation data type is not a platform type, i.e. it appears in the `Swc_impltypes.arxml` file.

```
<AR-PACKAGE>
    <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
    <ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE>
            <SHORT-NAME>uint8_16</SHORT-NAME>
            <!--
                array of 16 &quot;uint8&quot; values
            -->
            <CATEGORY>ARRAY</CATEGORY>
            <SUB-ELEMENTS>
                <IMPLEMENTATION-DATA-TYPE-ELEMENT>
                    <SHORT-NAME>ElementName</SHORT-NAME>
                    <CATEGORY>TYPE_REFERENCE</CATEGORY>
                    <ARRAY-SIZE>16</ARRAY-SIZE>
                    <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
                    <SW-DATA-DEF-PROPS>
                        <SW-DATA-DEF-PROPS-VARIANTS>
                            <SW-DATA-DEF-PROPS-CONDITIONAL>
                                <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                                /AUTOSAR_PlatformTypes/ImplementationDataTypes/uint8
                                </IMPLEMENTATION-DATA-TYPE-REF>
                            </SW-DATA-DEF-PROPS-CONDITIONAL>
                        </SW-DATA-DEF-PROPS-VARIANTS>
                    </SW-DATA-DEF-PROPS>
                </IMPLEMENTATION-DATA-TYPE-ELEMENT>
            </SUB-ELEMENTS>
        </IMPLEMENTATION-DATA-TYPE>
        ...
    </ELEMENTS>
</AR-PACKAGE>
```

Listing 17: ARXML code – implementation data type `uint8_16`

The implementation data type references the `uint8` platform type. In the `AUTOSAR_MOD_PlatformTypes.arxml` file, the referenced implementation data type looks as follows (the `<INTRODUCTION>` element may be non-empty in your generated code):

```
<IMPLEMENTATION-DATA-TYPE>
    <SHORT-NAME>uint8</SHORT-NAME>
    <LONG-NAME>
        <L-4 L="EN">unsigned integer 8bit</L-4>
    </LONG-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <INTRODUCTION>
    </INTRODUCTION>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_PlatformTypes/SwBaseTypes/uint8
                </BASE-TYPE-REF>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 18: ARXML code – platform data type `uint8`

The `uint8` platform type references the `uint8` base type; the latter is provided in the `AUTOSAR_MOD_PlatformBaseTypes_RTARTE.arxml`[7] file:

```
<IMPLEMENTATION-DATA-TYPE>
    <SHORT-NAME>uint8</SHORT-NAME>
    <LONG-NAME>
        <L-4 L="EN">unsigned integer 8bit</L-4>
    </LONG-NAME>
    <CATEGORY>VALUE</CATEGORY>
    <INTRODUCTION>
    </INTRODUCTION>
    <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_PlatformTypes/SwBaseTypes/uint8
                </BASE-TYPE-REF>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

Listing 19: ARXML code – base type `uint8`

The RTE generator will generate a C array type for each defined `<ARRAY-TYPE>`. Therefore, array types must be declared according to the same semantics as the

[7] In older RTA-RTE versions: `AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml`

C array. The array type definition (e.g., `typedef uint8 uint8_16[16];`) is included in the generated file `Rte_Type.h`.

> **NOTE**
>
> The implementation of arrays in application software components shall be consistent with their declaration in the generated RTE. For more information, refer to the `AUTOSAR_SWS_RTE.pdf` manual of your AUTOSAR release.

4.8.3 Matrix Data Types

Matrix data types, like array and record data types, allow new complex data types to be created. In the ARXML code, a matrix is generated as an *array of arrays*.

> **NOTE**
>
> If you use matrices, make sure that the "Number of dimensions for fixed matrixes" target option is set to `Two-dimensional`.
>
> Otherwise, a warning `WMdl653` is issued during AUTOSAR code generation. By default, this warning is promoted to an error.

To create a matrix

1. In the component manager, double-click the project `ARProject`.

   The project editor opens.
2. In the project editor, double-click the software component `Swc`.

   The software component editor opens.
3. Use the **Matrix** button to create a matrix.

   The properties editor for the matrix opens.
4. Name the variable `IRV_matrix` and set the following properties:

Dimension X	4
Dimension Y	3
Kind	Interrunnable Variable
Basic type	Wrap-Around integer
Min / Max	0 / 255
Type	uint8

5. Close the properties editor with **OK**.

   You do not need to edit the implementation of `IRV_matrix`. The values you entered for min, max, and type

An implementation of a matrix in ASCET is represented as an array of arrays in AUTOSAR. Two data types can be found in the `ASCET_types` AUTOSAR package in `Swc_appltypes.arxml`, one for the array of arrays[8], and one for the arrays[9] that form the array of arrays.

The following application data types are defined in `Swc_appltypes.arxml` for the matrix (or array of arrays) `IRV_matrix`:

```
<AR-PACKAGE>
  <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
  <ELEMENTS>
    ...
    <APPLICATION-ARRAY-DATA-TYPE>
      <SHORT-NAME>UInt8_4_3</SHORT-NAME>
      <!--
        array of 4 &quot;UInt8_3&quot; values
      -->
      <CATEGORY>ARRAY</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>
      <ELEMENT>
        <SHORT-NAME>ElementName</SHORT-NAME>
        <CATEGORY>VALUE</CATEGORY>
        <SW-DATA-DEF-PROPS>
          <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
        <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
        /ASCET_Types/ApplicationDataTypes/UInt8_3</TYPE-TREF>
        <ARRAY-SIZE-SEMANTICS>FIXED-SIZE
        </ARRAY-SIZE-SEMANTICS>
        <MAX-NUMBER-OF-ELEMENTS>4</MAX-NUMBER-OF-ELEMENTS>
      </ELEMENT>
    </APPLICATION-ARRAY-DATA-TYPE>
    ...
    <APPLICATION-ARRAY-DATA-TYPE>
      <SHORT-NAME>UInt8_3</SHORT-NAME>
      <!--
        array of 3 &quot;UInt8&quot; values
      -->
```

[8] `<SHORT-NAME>UInt8_4_3</SHORT-NAME>` in Listing 20

[9] `<SHORT-NAME>UInt8_3</SHORT-NAME>` in Listing 20

```
                <CATEGORY>ARRAY</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>
                <ELEMENT>
                    <SHORT-NAME>ElementName</SHORT-NAME>
                    <CATEGORY>VALUE</CATEGORY>
                    <SW-DATA-DEF-PROPS>
                        <SW-DATA-DEF-PROPS-VARIANTS>
                            <SW-DATA-DEF-PROPS-CONDITIONAL>
                            </SW-DATA-DEF-PROPS-CONDITIONAL>
                        </SW-DATA-DEF-PROPS-VARIANTS>
                    </SW-DATA-DEF-PROPS>
                    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/UInt8</TYPE-TREF>
                    <ARRAY-SIZE-SEMANTICS>FIXED-SIZE
                    </ARRAY-SIZE-SEMANTICS>
                    <MAX-NUMBER-OF-ELEMENTS>3</MAX-NUMBER-OF-ELEMENTS>
                </ELEMENT>
            </APPLICATION-ARRAY-DATA-TYPE>
            ...
        </ELEMENTS>
    </AR-PACKAGE>
```

Listing 20: ARXML code – application data types `UInt8_4_3` and `UInt8_3` of category `ARRAY` (generated for `IRV_matrix`)

In the `Swc_mappings.arxml` file, the application data types are mapped to implementation data types:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>DataMappings</SHORT-NAME>
            <AR-PACKAGES>
                <AR-PACKAGE>
                    <SHORT-NAME>Impl</SHORT-NAME>
                    <ELEMENTS>
                        <DATA-TYPE-MAPPING-SET>
                            <SHORT-NAME>Swc</SHORT-NAME>
                            <DATA-TYPE-MAPS>
                                ...
```

```
                <DATA-TYPE-MAP>
                    <APPLICATION-DATA-TYPE-REF
                    DEST="APPLICATION-ARRAY-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/UInt8_4_3
                    </APPLICATION-DATA-TYPE-REF>
                    <IMPLEMENTATION-DATA-TYPE-REF
                    DEST="IMPLEMENTATION-DATA-TYPE">/ASCET_Types
                    /ImplementationDataTypes/uint8_4_3
                    </IMPLEMENTATION-DATA-TYPE-REF>
                </DATA-TYPE-MAP>
                ...
                <DATA-TYPE-MAP>
                    <APPLICATION-DATA-TYPE-REF
                    DEST="APPLICATION-ARRAY-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/UInt8_3
                    </APPLICATION-DATA-TYPE-REF>
                    <IMPLEMENTATION-DATA-TYPE-REF
                    DEST="IMPLEMENTATION-DATA-TYPE">/ASCET_Types
                    /ImplementationDataTypes/uint8_3
                    </IMPLEMENTATION-DATA-TYPE-REF>
                </DATA-TYPE-MAP>
                ...
            </DATA-TYPE-MAPS>
            <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
          </DATA-TYPE-MAPPING-SET>
        </ELEMENTS>
      </AR-PACKAGE>
    </AR-PACKAGES>
  </AR-PACKAGE>
 </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 21: ARXML code – mapping of `UInt8_4_3` and `UInt8_3` application types and implementation data types

The referenced implementation data types are no platform types, i.e. they appear in the `Swc_impltypes.arxml` file.

```
<AR-PACKAGE>
  <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
    <ELEMENTS>
    ...
    <IMPLEMENTATION-DATA-TYPE>
      <SHORT-NAME>uint8_4_3</SHORT-NAME>
      <!--
        array of 4 &quot;uint8_3&quot; values
      -->
      <CATEGORY>ARRAY</CATEGORY>
      <SUB-ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
          <SHORT-NAME>ElementName</SHORT-NAME>
          <CATEGORY>TYPE_REFERENCE</CATEGORY>
```

```
                <ARRAY-SIZE>4</ARRAY-SIZE>
                <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
                <SW-DATA-DEF-PROPS>
                    <SW-DATA-DEF-PROPS-VARIANTS>
                        <SW-DATA-DEF-PROPS-CONDITIONAL>
                            <IMPLEMENTATION-DATA-TYPE-REF
                            DEST="IMPLEMENTATION-DATA-TYPE">
                            /ASCET_Types/ImplementationDataTypes/uint8_3
                            </IMPLEMENTATION-DATA-TYPE-REF>
                        </SW-DATA-DEF-PROPS-CONDITIONAL>
                    </SW-DATA-DEF-PROPS-VARIANTS>
                </SW-DATA-DEF-PROPS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
        </SUB-ELEMENTS>
    </IMPLEMENTATION-DATA-TYPE>
    ...
    <IMPLEMENTATION-DATA-TYPE>
        <SHORT-NAME>uint8_3</SHORT-NAME>
        <!--
            array of 3 &quot;uint8&quot; values
        -->
        <CATEGORY>ARRAY</CATEGORY>
        <SUB-ELEMENTS>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
                <SHORT-NAME>ElementName</SHORT-NAME>
                <CATEGORY>TYPE_REFERENCE</CATEGORY>
                <ARRAY-SIZE>3</ARRAY-SIZE>
                <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
                <SW-DATA-DEF-PROPS>
                    <SW-DATA-DEF-PROPS-VARIANTS>
                        <SW-DATA-DEF-PROPS-CONDITIONAL>
                            <IMPLEMENTATION-DATA-TYPE-REF
                            DEST="IMPLEMENTATION-DATA-TYPE">
                            /AUTOSAR_Platform/ImplementationDataTypes/uint8
                            </IMPLEMENTATION-DATA-TYPE-REF>
                        </SW-DATA-DEF-PROPS-CONDITIONAL>
                    </SW-DATA-DEF-PROPS-VARIANTS>
                </SW-DATA-DEF-PROPS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
        </SUB-ELEMENTS>
    </IMPLEMENTATION-DATA-TYPE>
    ...
  </ELEMENTS>
</AR-PACKAGE>
```

Listing 22: ARXML code – implementation data types `uint8_4_3` and `uint8_3` generated for `IRV_matrix`

The implementation data type references the `uint8` platform type. In the `AUTOSAR_MOD_PlatformTypes.arxml` file, the referenced implementation data type looks as shown in Listing 18 on page 56.

The `uint8` platform type references the `uint8` base type; the latter is provided in the `AUTOSAR_MOD_PlatformBaseTypes_RTARTE.arxml`[10] file; see Listing 19 on page 56.

The RTE generator will generate two C array types for each defined matrix. These array type definitions are included in the generated file `Rte_Type.h`.

For `IRV_matrix`, the generated array type definitions look as follows:

```
typedef uint8 uint8_3[3];
typedef uint8_3 uint8_4_3[4];
```

NOTE

The implementation of arrays of arrays, and arrays in application software components shall be consistent with their declaration in the generated RTE. For more information, refer to the `AUTOSAR_SWS_RTE.pdf` manual of your AUTOSAR release.

[10] In older RTA-RTE versions: `AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml`

5 Interfaces

When an application consists of multiple software components, it may be necessary for the software components to communicate, either to exchange data or to trigger some function. Communication between AUTOSAR software components is designed in terms of ports and interfaces. The following interface types are available:

A. Sender-receiver (signal passing) – see section 5.1
B. Mode-switch (communication of mode switches) – see section 5.2
C. Client-server (function invocation) – see section 5.3
D. Calibration – see section 5.4
E. NV-data (non-volatile signal passing) – see section 5.5

These communication models are known as interfaces in AUTOSAR.

All ports of a software component (whether a provided or a required port) are typed by a specific interface. Interface types are defined using either the `<SENDER-RECEIVER-INTERFACE>` or `<MODE-SWITCH-INTERFACE>` or `<CLIENT-SERVER-INTERFACE>` or `<PARAMETER-INTERFACE>` or `<NV-DATA-INTERFACE>` elements.

The definition of sender-receiver, mode-switch, client-server, calibration, and NV-data interfaces is considered in detail in this section.

Note that the way the software component interacts with the interface is defined by the `<INTERNAL-BEHAVIOR>` element that references a software component. This is discussed in chapter 7*, Internal Behavior*, on page 105.

5.1 Sender-Receiver

Sender-receiver communication involves the transmission and reception of signals consisting of atomic data elements sent by one component and received by one or more components.

Each sender-receiver interface may contain multiple data elements, each of which can be sent and received independently.

To create a sender-receiver interface

1. In the component manager, select **Insert › AUTOSAR › SenderReceiver-Interface**.
2. Name the sender-receiver interface `SRInterface`.
3. Insert `SRInterface` into `SWC`.

When generating code for an AUTOSAR project, ASCET defines a `<SENDER-RECEIVER-INTERFACE>` element in the file `Swc_interfaces.arxml`. The `<SENDER-RECEIVER-INTERFACE>` element has the following structure in the configuration language:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>Impl</SHORT-NAME>
            <ELEMENTS>
                ...
                <SENDER-RECEIVER-INTERFACE>
                    <SHORT-NAME>SRInterface</SHORT-NAME>
                    <IS-SERVICE>false</IS-SERVICE>
                    <DATA-ELEMENTS>
                        ...
                    </DATA-ELEMENTS>
                </SENDER-RECEIVER-INTERFACE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 23: ARXML code – sender-receiver interface definition

The name of the sender-receiver interface definition is given by the `<SHORT-NAME>`. The name is used within other elements that need to reference the interface type, for example a software component may specify that it uses sender-receiver interface `SRInterface`.

The short-name of a sender-receiver interface should be a valid C identifier.

A sender-receiver interface can be used to communicate data (using data element prototypes within the `<DATA-ELEMENTS>` element) or modes (see section 5.2, *Mode Switch*, on page 67 for more details).

> **NOTE**
>
> In AUTOSAR R4.*, a sender-receiver interface must contain *either* data elements *or* a single mode group. If a sender-receiver interface contains both kinds of elements, an error is issued during code generation.

Data Element Prototypes

Each sender-receiver interface can specify zero or more data elements that constitute the AUTOSAR signals communicated over the interface. Each data item defines a prototype of a specific type and can be a primitive data type, a `RECORD` or an `ARRAY` type. See chapter 4, *Data Types*, on page 31 for details of defining data types.

To create data elements in ASCET

1. In the component manager, double-click `SRInterface`.

   The "Sender Receiver Interface Editor for: SRInterface" editor opens.

2. Use the **Limited Integer Variable** button to create a `limitInt` variable.

The properties editor for the new variable opens.

3. Name the variable `Speed` and set "Min" and "Max" to -32768 and 32767.

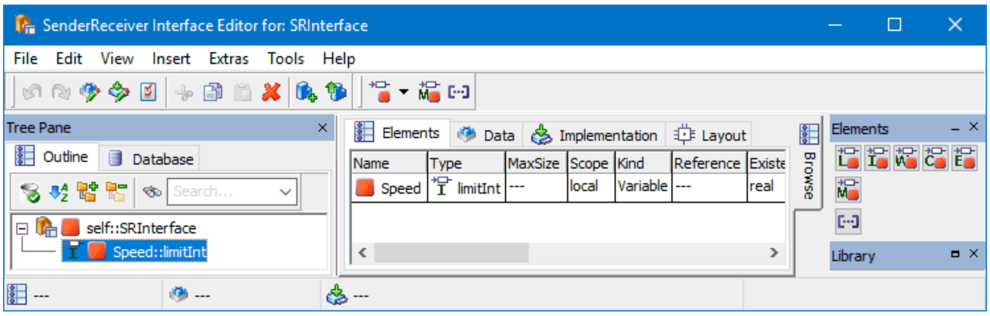


Figure 18: Data element `Speed` for the sender-receiver interface `SRInterface`

4. Create a logical variable named `log`.

To create an implementation of a data element

1. In the "Sender Receiver Interface Editor for: SRInterface" editor, go to the **Implementation** tab.
2. In the **Implementation** tab, double-click the `Speed` element.

   The "Implementation for: Speed" window opens.
3. In the "Implementation" field, select the type `sint16`.
4. Close the "Implementation for: sdisc" window with **OK**.

   The **Implementation** tab of the "Sender Receiver Interface Editor for: SRInterface" editor shall look like the figure below.
5. For `log`, select the implementation data type `bool`.

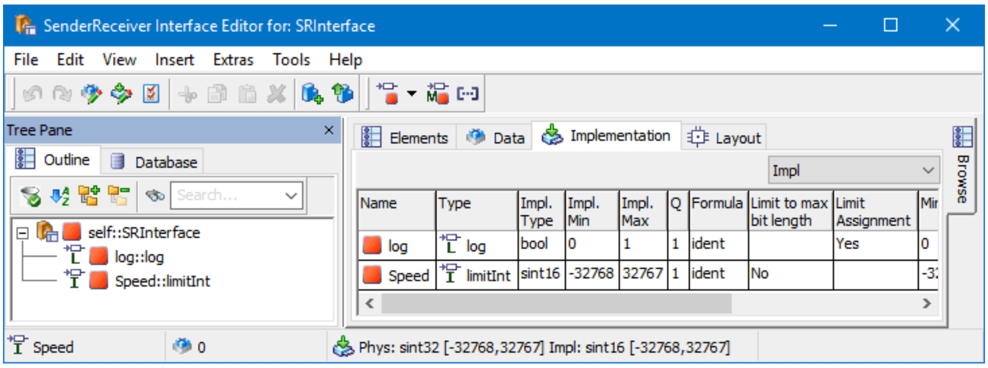


Figure 19: Implementation `Impl` of the sender-receiver interface `SRInterface` with data elements `Speed` and `log`

An implementation of a sender-receiver interface in ASCET corresponds to a sender-receiver interface in AUTOSAR. The sender-receiver interface in configuration language is generated by ASCET in the file `Swc_interfaces.arxml`.

The implementation editor of a sender-receiver interface element also contains an "AUTOSAR" tab with a policy and an invalidation policy. These are written to the ARXML, the latter can also specify the existance of the `Rte_IStatus` macro. See the online help, section "RTE Access Macros", for details.

In AUTOSAR R4. *, the declaration of data elements within a sender-receiver interface definition has the following structure:

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <ELEMENTS>
        ...
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>SRInterface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>log</SHORT-NAME>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
                    /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
                    <SW-CALIBRATION-ACCESS>READ-ONLY
                    </SW-CALIBRATION-ACCESS>
                    <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
              <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/Boolean</TYPE-TREF>
            </VARIABLE-DATA-PROTOTYPE>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>Speed</SHORT-NAME>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
                    /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
                    <SW-CALIBRATION-ACCESS>READ-ONLY
                    </SW-CALIBRATION-ACCESS>
                    <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
              <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 24: ARXML code – declaration of data elements within sender-receiver interface

A data element is defined using the <VARIABLE-DATA-PROTOTYPE> element, and all elements must be defined within an encapsulating <DATA-ELEMENTS> element.

Each <VARIABLE-DATA-PROTOTYPE> element must specify:

- the <SHORT-NAME> that you will use to refer to the item
- the <SW-DATA-DEF-PROPS> data properties, among them
  - the <SW-CALIBRATION-ACCESS>

– a `<TYPE-TREF>` reference to the type of the data item

5.2 Mode Switch

An AUTOSAR system can be configured to operate in one or more application modes. A mode-switch interface can specify zero or more mode groups that define application modes.

In ASCET, mode-switch interfaces are realized as sender-receiver interface components that contain mode groups.

Since AUTOSAR R4.0, a sender-receiver interface that contains a mode group must not contain data elements, and vice versa. Mixing both kinds of elements leads to a code generation error.

To create a mode group

1. In the component manager, select **Insert › AUTOSAR › Mode Group**.
2. Name the mode group `OnOffMode`.
3. In the "Database" or "Workspace" pane, select `OnOffMode` and go to the "Contents" pane.
4. Select **Mode › Rename** to rename the label `mode` as `off`.
5. Select **Mode › Add Mode › As Last** to create a new mode `on`.

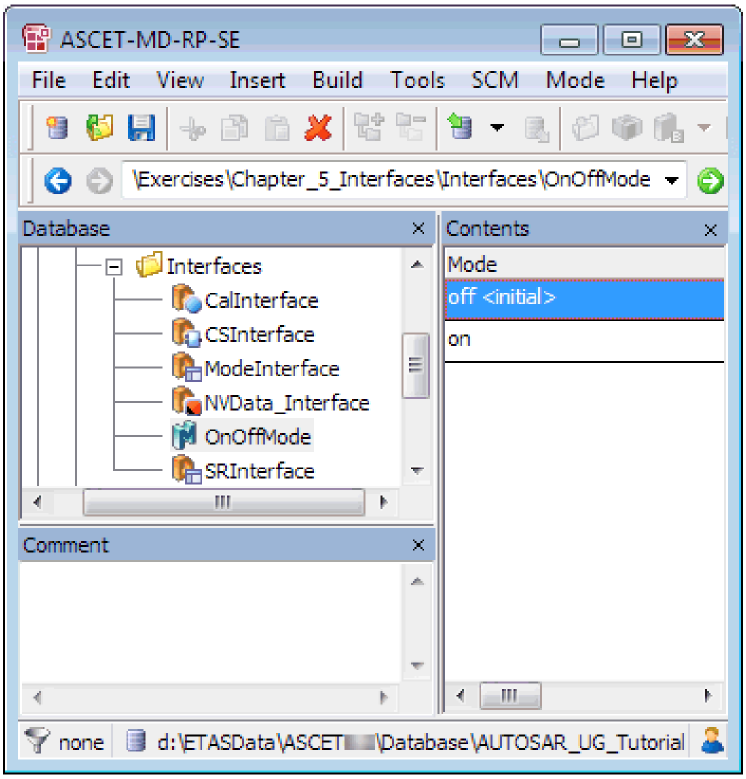


Figure 20: Mode declaration group `OnOffMode`

In AUTOSAR R4.*, ASCET declares the `<MODE-DECLARATION-GROUP>` in the *`<swc name>`*`_appltypes.arxml` file, AUTOSAR package `ASCET_types`, sub-package `ApplicationDataTypes`.

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Types</SHORT-NAME>
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
            <ELEMENTS>
                <MODE-DECLARATION-GROUP>
                    <SHORT-NAME>OnOffMode</SHORT-NAME>
                    <CATEGORY>EXPLICIT_ORDER</CATEGORY>
                    <INITIAL-MODE-REF DEST="MODE-DECLARATION">
                    /ASCET_Types/ApplicationDataTypes/OnOffMode/off
                    </INITIAL-MODE-REF>
                    <MODE-DECLARATIONS>
                        <MODE-DECLARATION>
                            <SHORT-NAME>off</SHORT-NAME>
                            <VALUE>1</VALUE>
                        </MODE-DECLARATION>
                        <MODE-DECLARATION>
                            <SHORT-NAME>on</SHORT-NAME>
                            <VALUE>2</VALUE>
                        </MODE-DECLARATION>
                    </MODE-DECLARATIONS>
                    <ON-TRANSITION-VALUE>3</ON-TRANSITION-VALUE>
                </MODE-DECLARATION-GROUP>
                ...
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 25: ARXML code – mode declaration group

To create a mode-switch interface

> **NOTE**
>
> AUTOSAR R4.* allows a single mode group in a sender-receiver interface with no data elements.

1. In the component manager, select **Insert › AUTOSAR › SenderReceiver Interface**.
2. Name the sender-receiver interface `ModeInterface`.
3. Double-click `ModeInterface`.

   The "Sender Receiver Interface Editor for: ModeInterface" editor opens.
4. Select **Insert › Component**.

   The "Select Item" window opens.

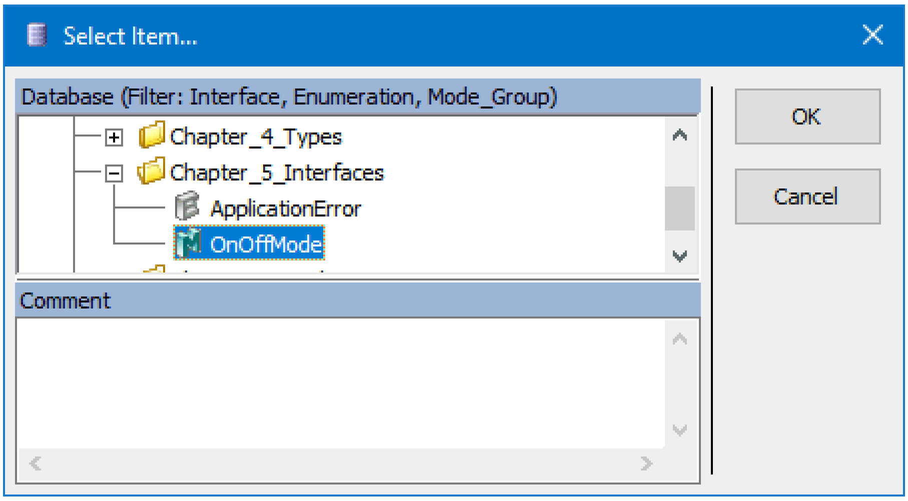


Figure 21: Selection of the mode group `OnOffMode`

5. In the "Database" or "Workspace" field of the "Select Item" window, select the mode group `OnOffMode`.
6. Click **OK** to close the "Select Item" window and insert `OnOffMode` into `ModeInterface`.

   The "Properties for Element: OnOffMode" window opens. You can enter a name and a comment for the `OnOffMode` instance.
7. Click **OK** to use the default name and comment.

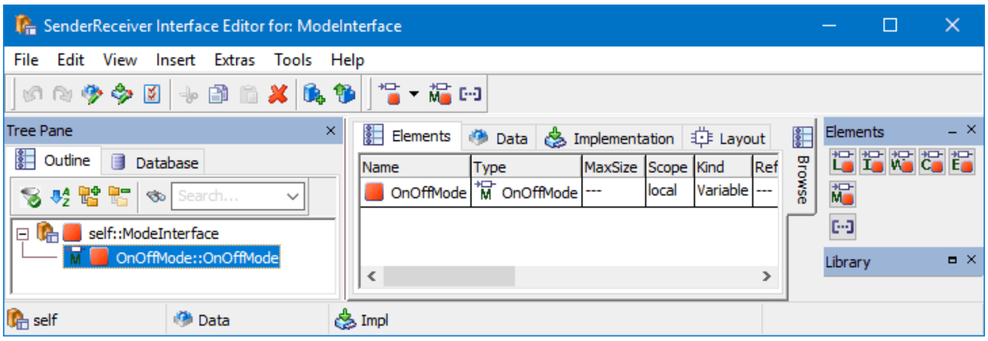


Figure 22: Mode-switch interface `ModeInterface`

8. Insert `ModeInterface` into `SWC`.
9. Generate code for the AUTOSAR project.

In AUTOSAR R4.*, the declaration of the mode group within a mode-switch interface definition has the following structure:

```
<MODE-SWITCH-INTERFACE>
   <SHORT-NAME>ModeInterface</SHORT-NAME>
   <MODE-GROUP>
      <SHORT-NAME>OnOffMode</SHORT-NAME>
      <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
      /ASCET_Types/ApplicationDataTypes/OnOffMode</TYPE-TREF>
   </MODE-GROUP>
</MODE-SWITCH-INTERFACE>
```

Listing 26: ARXML code – declaration of mode group within mode-switch interface

In AUTOSAR R4.*, a mode group is defined using the `<MODE-GROUP>` element.

Each `<MODE-GROUP>` element must specify the following:

– the `<SHORT-NAME>` that you will use to refer to the item

– the `<TYPE-TREF>` reference to mode declaration group

The use of mode declaration prototypes within sender-receiver interfaces is considered in detail in chapter 8, *Modes*, on page 165.

5.3 Client-Server

Client-server communication involves a component invoking a defined "server" function in another component, which may or may not return a reply. Each client-server interface can contain multiple operations, each of which can be invoked separately.

To create a client-server interface:

1. In the component manager, select **Insert > AUTOSAR > ClientServer Interface**.
2. Name the client-server interface `CSInterface`.
3. Insert `CSInterface` into `SWC`.

When generating code in an AUTOSAR project, ASCET defines the `<CLIENT-SERVER-INTERFACE>` element in the file `Swc_interfaces.arxml`. The `<CLIENT-SERVER-INTERFACE>` element has the following structure in the configuration language:

```
<CLIENT-SERVER-INTERFACE>
   <SHORT-NAME>CSInterface</SHORT-NAME>
   <IS-SERVICE>false</IS-SERVICE>
   <OPERATIONS>
      ...
   </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

Listing 27: ARXML code – client-server interface structure (all AUTOSAR versions)

A client-server interface is named using the `<SHORT-NAME>` element. The name is used within other elements that need to reference the interface type.

The short-name of a client-server interface should be a valid C identifier.

A client-server interface consists of one or more operations defined using the `<OPERATIONS>` container element.

Operations

An operation in a client-server interface can take zero or more parameters. The return value of an operation is either of type `Std_ReturnType` or of an enumeration type, depending on whether or not the operation returns an application error.

To create an operation

1. In the component manager, double-click `CSInterface`.

   The "Interface Editor for: CSInterface" editor opens.
2. In the "Outline" tab, select the `Main` diagram.
3. Select **Insert > Method Signature**.

   An operation is added.
4. Name the operation `MaximumValue`.

To create arguments in an operation

1. Double-click the operation `MaximumValue`.

   The "Method Signature Editor for: MaximumValue" window opens.
2. Select **Argument > Add** and name the first argument `InputA`. Set the following parameters:
   - Argument Type: `sdisc`
   - Direction: `in`
3. Create a second argument `InputB` with the same type and direction.
4. Create a third argument `OutputMaximum` with type `sdisc` and direction `Out`.

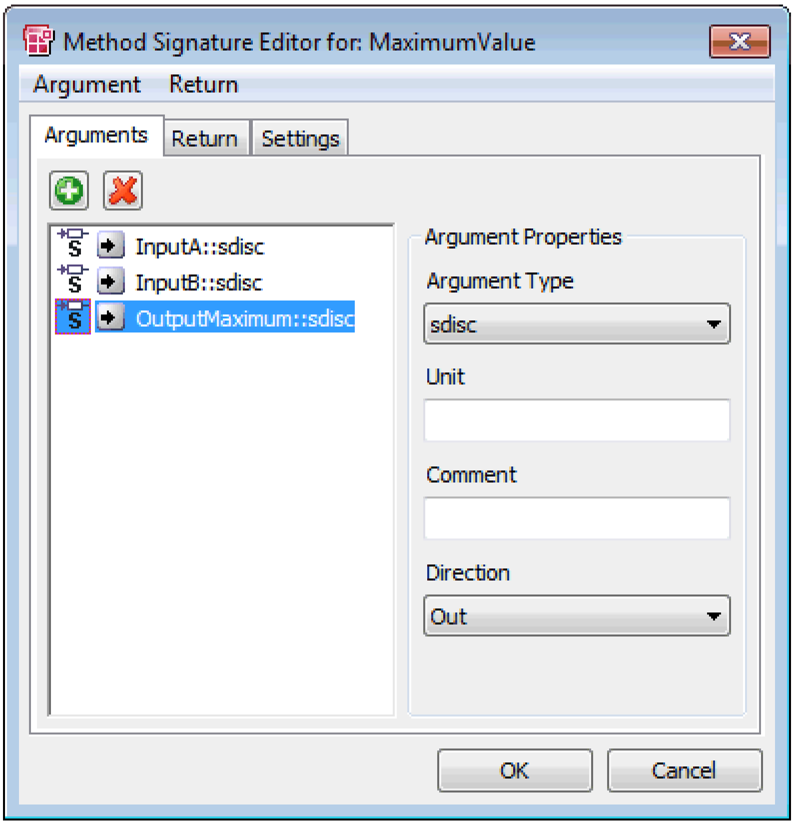


Figure 23: Arguments of the operation `MaximumValue`

5. Click **OK**.

   ASCET represents the client-server interface `CSInterface` with operation `MaximumValue` and arguments `InputA`, `InputB` and `OutputMaximum` as follows.

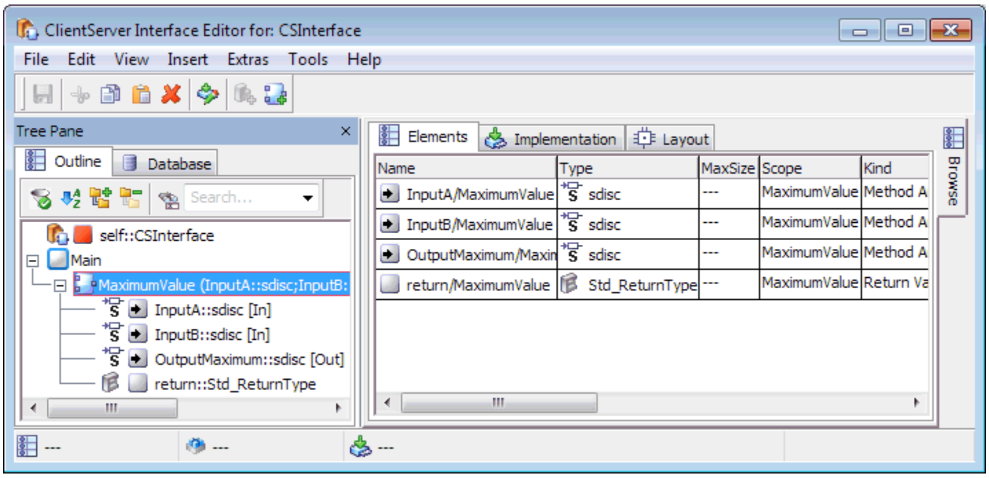


Figure 24: Operation `MaximumValue` for the client-server interface `CSInter-face`

To create an implementation of an operation

1. In the "Interface Editor for: CSInterface" editor, go to the "Implementation" tab.

2. In the "Implementation" tab, double-click the `InputA` element.

   The "Implementation for: InputA" window opens.
3. In the "Master" field, activate **Implementation**.
4. In the "Implementation" field, select `sint16`.
5. Right-click in the "Min" and "Max" fields and select **Default Value** from the context menu.
6. Close the "Implementation for: InputA" window with **OK**.
7. Repeat the implementation procedure for the arguments `InputB` and `OutputMaximum`.

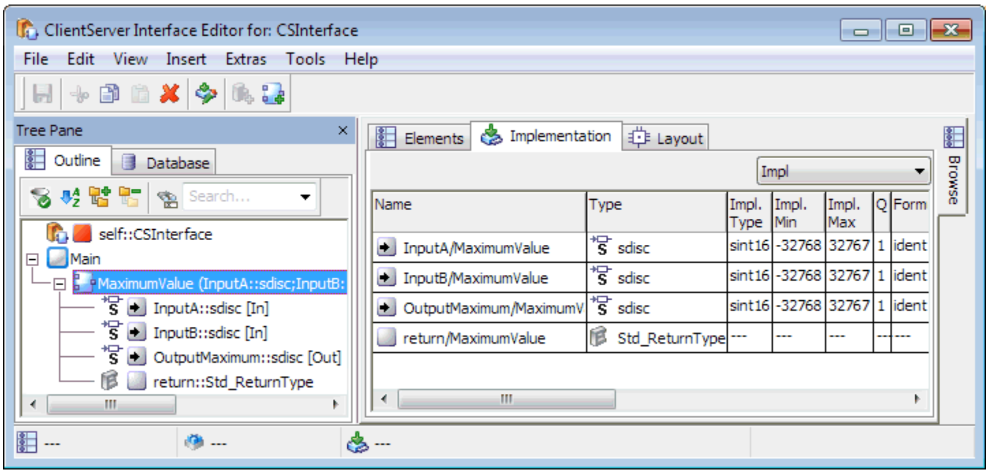


Figure 25: Implementation of the operation `MaximumValue`

8. Generate code for the AUTOSAR project.

An implementation of a client-server interface in ASCET corresponds to a client-server interface in AUTOSAR. The client-server interface in configuration language is generated by ASCET in the `Swc_interfaces.arxml` file. The `<OPERATIONS>` element encapsulates one or more `<CLIENT-SERVER-OPERATION>` elements, each of which defines a single operation in the client-server interface.

```xml
<CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>CSInterface</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <OPERATIONS>
        <CLIENT-SERVER-OPERATION>
            <SHORT-NAME>MaximumValue</SHORT-NAME>
            <ARGUMENTS>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>InputA</SHORT-NAME>
                    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>InputB</SHORT-NAME>
                    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>OutputMaximum</SHORT-NAME>
                    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
                    /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
                    <DIRECTION>OUT</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
            </ARGUMENTS>
        </CLIENT-SERVER-OPERATION>
    </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

Listing 28: ARXML code – operation in a client-server interface

Each operation is named using the `<SHORT-NAME>` element. The name specified here will form part of the name used by the RTE to refer to the operation in your code.

The `<ARGUMENTS>` element encapsulates one or more `<ARGUMENT-DATA-PROTOTYPE>` elements that define each argument (parameter) of the operation.

Each `<ARGUMENT-DATA-PROTOTYPE>` definition must define the following:

- the `<SHORT-NAME>` of the parameter
- a `<TYPE-TREF>` reference to the type of the parameter

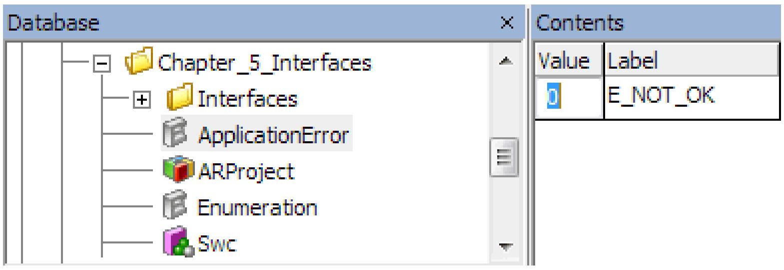
  The referenced type must correspond to a defined type – see chapter 4, *Data Types*, on page 31
- the `<DIRECTION>` of the parameter as `"IN"` (read only), `"OUT"` (write only) or `"INOUT"` (readable and writable by the component)

If nothing else is specified, operations in client-server interfaces return the RTE standard return type `Std_ReturnType`. It is also possible to return an application error. This is done by selecting a previously defined ASCET enumeration that contains all possible errors.

To create an enumeration with the possible errors in an application error

1. In the component manager, select **Insert › Enumeration** or click the **Enumeration** button.

2. Name the enumeration `ApplicationError`.
3. In the "Contents" pane, select the enumerator.
4. Select **Enumeration > Rename** and set the label to `E_NOT_OK`.
5. Double-click the value 0.



6. Set the value to a number in the range `2..63`.

> **NOTE**
>
> The value range for application errors is `[2..63]`. If the ASCET enumeration for the application errors contains a value less than 2 or larger than 63, an error is issued during code generation.

To assign an application error to the return value of an operation

1. Open `CSInterface` in the client-server interface editor.
2. Create another operation (see page 71) and name it `Notification`.
3. Double-click the operation `Notification`.

   The "Method Signature Editor for: Notification" window opens.
4. Go to the "Return" tab and open the "Return Type" combo box.

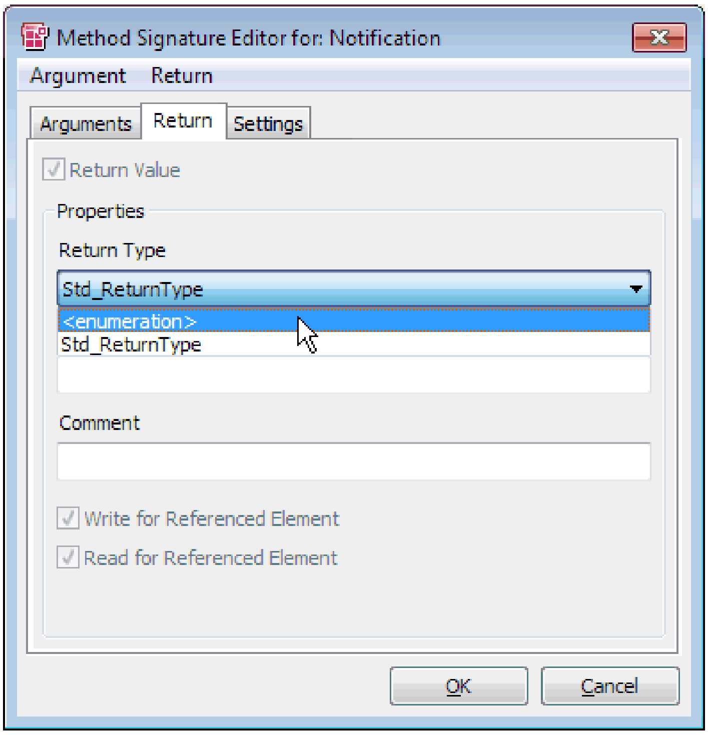
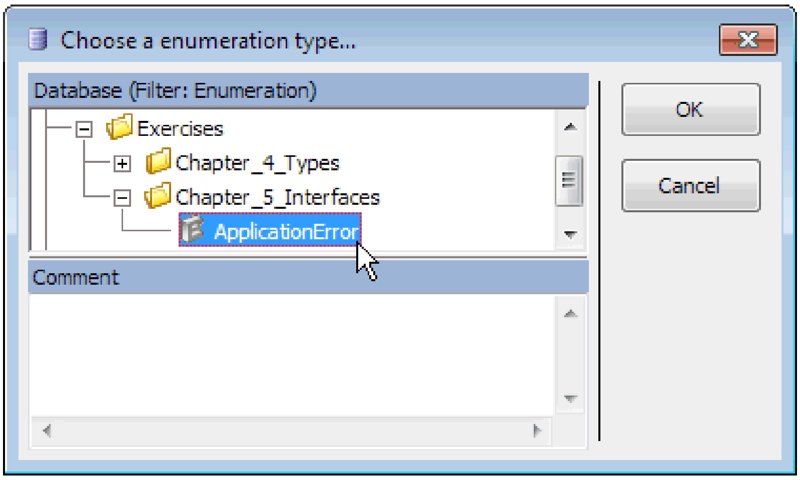


Figure 26: Return type for the operation `Notification`

5. Select <enumeration>.

   The "Choose a enumeration type..." window opens.



6. Select the enumeration `ApplicationError`.
7. Click **OK** to close the "Choose a enumeration type..." window.
8. Click **OK** to close the method signature editor.

The operation `Notification` and the possible application errors in configuration language are generated by ASCET in the `Swc_interfaces.arxml` file:

```
<CLIENT-SERVER-INTERFACE>
   <SHORT-NAME>CSInterface</SHORT-NAME>
   <IS-SERVICE>false</IS-SERVICE>
   <OPERATIONS>
      <CLIENT-SERVER-OPERATION>
         <SHORT-NAME>MaximumValue</SHORT-NAME>
         ...
      </CLIENT-SERVER-OPERATION>
      <CLIENT-SERVER-OPERATION>
         <SHORT-NAME>Notification</SHORT-NAME>
         <ARGUMENTS></ARGUMENTS>
         <POSSIBLE-ERROR-REFS>
            <POSSIBLE-ERROR-REF DEST="APPLICATION-ERROR">
            /ASCET_Interfaces/Impl/CSInterface/E_NOT_OK
            </POSSIBLE-ERROR-REF>
         </POSSIBLE-ERROR-REFS>
      </CLIENT-SERVER-OPERATION>
   </OPERATIONS>
   <POSSIBLE-ERRORS>
      <APPLICATION-ERROR>
         <SHORT-NAME>E_NOT_OK</SHORT-NAME>
         <ERROR-CODE>2</ERROR-CODE>
      </APPLICATION-ERROR>
   </POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
```

Listing 29: ARXML code – operation with possible application errors

> **NOTE**
>
> Application errors are coded in the least significant 6 bits of `Std_ReturnType`. The value range for application errors is `[2..63]`. If the ASCET enumeration for the application errors contains a value less than 2 or larger than 63, an error is issued during code generation.

5.4 Calibration

Calibration interfaces are used for communication with Calibration components. Calibration components are a kind of software component, which uniquely consist of calibration information (parameters and characteristics).

Each calibration interface can contain multiple calibration parameters. A port of a software component that requires an AUTOSAR calibration interface can independently access any of the parameters defined in the interface by making an RTE API to the required port. Calibration components provide the calibration interface and thus provide implementations of the calibration parameters.

To create a calibration interface

1. In the component manager, select **Insert › AUTOSAR › Calibration Interface**.
2. Name the calibration interface `CalInterface`.

3. Insert `CalInterface` into `SWC`.

When generating code for an AUTOSAR project, ASCET defines a `<PARAMETER-INTERFACE>` element in the file `Swc_interfaces.arxml`. The `<PARAMETER-INTERFACE>` element has the following structure in the configuration language:

```
<PARAMETER-INTERFACE>
    <SHORT-NAME>CalInterface</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <PARAMETERS>
        <PARAMETER-DATA-PROTOTYPE>
            ...
        </PARAMETER-DATA-PROTOTYPE>
        ...
    </PARAMETERS>
</PARAMETER-INTERFACE>
```

Listing 30: ARXML code – calibration interface structure

A calibration interface is named using the `<SHORT-NAME>` element. The name is used within other elements that need to reference the interface type.

The short-name of a calibration interface should be a valid C identifier.

A calibration interface consists of one or more calibration elements defined using the `<PARAMETER-DATA-PROTOTYPE>` container element.

Calibration Parameters

To create a calibration parameter

1. In the component manager, double-click `CalInterface`.

   The "Calibration Interface Editor for: CalInterface" editor opens.

2. Use the **Logic Parameter** button to create a logic parameter.

   The dialog "Properties for Scalar Element: log" window opens.

3. Name the parameter `CalParam1`.
4. Create another logic parameter `CalParam2`.
5. Create ( ) an unsigned discrete parameter `CalParam3`.

To create an implementation of a calibration parameter

1. In the "Calibration Interface Editor for: CalInterface" editor, go to the "Implementation" tab.
2. In the "Implementation" tab, double-click the `CalParam3` element.
3. The "Implementation for: CalParam3" dialog opens.
4. In the "Master" area, activate **Model**.
5. In the "Model" area, enter the value 24 in the "Max" field.
6. Click **OK**.

The **Implementation** tab of the "Calibration Interface Editor for: CalInterface" editor shall look like the figure below.

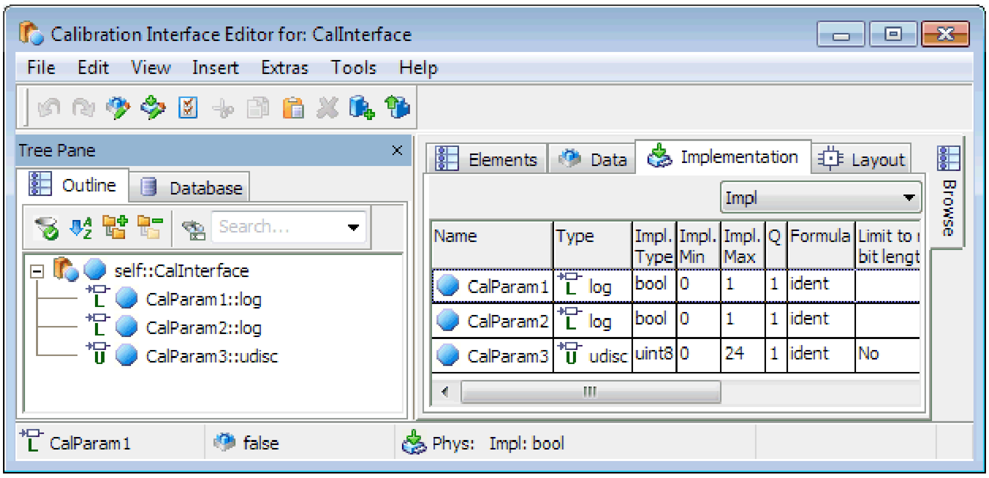


Figure 27: Implementation `Impl` of the calibration interface `CalInterface`

The implementation editor also contains an "AUTOSAR" tab with policy settings. These are written to the ARXML files.

An implementation of a calibration interface in ASCET corresponds to a calibration interface in AUTOSAR. The calibration interface in configuration language is generated by ASCET in the `Swc_interfaces.arxml` file. The declaration of calibration elements within a calibration interface definition has the following structure:

```
<PARAMETER-INTERFACE>
  <SHORT-NAME>CalInterface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <PARAMETERS>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam1</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/CAL_MEM_CLEARED</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/Boolean</TYPE-TREF>
    </PARAMETER-DATA-PROTOTYPE>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam2</SHORT-NAME>
      ...
    </PARAMETER-DATA-PROTOTYPE>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam3</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/CAL_MEM_CLEARED</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/SInt8_ident_p0_p24</TYPE-TREF>
    </PARAMETER-DATA-PROTOTYPE>
  </PARAMETERS>
</PARAMETER-INTERFACE>
```

Listing 31: ARXML code – declaration of calibration elements within a calibration interface definition

A calibration element is defined using the <PARAMETER-DATA-PROTOTYPE> element, and all elements must be defined within an encapsulating <PARAMETERS> element.

Each <PARAMETER-DATA-PROTOTYPE> element must specify the following:

- the <SHORT-NAME> that you will use to refer to the item
- the <SW-DATA-DEF-PROPS> data properties, among them
  - the <SW-CALIBRATION-ACCESS>
- a <TYPE-TREF> reference to the type of the data item

5.5 NVData

AUTOSAR R4.0 introduced the <NV-DATA-INTERFACE> element, which defines an interface used by an Nv-block software component type. Each NVData interface may contain multiple NVData elements, which can be sent and received independently.

To create an NVData interface

1. In the component manager, select **Insert > AUTOSAR > NVData Interface**.
2. Name the NVData interface `NVData_Interface`.
3. Insert `NVData_Interface` into `SWC`.

When generating code for an AUTOSAR project, ASCET defines an `<NVDATA-INTERFACE>` element in the file `Swc_interfaces.arxml`. The `<NVDATA-INTERFACE>` element has the following structure in the configuration language:

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <ELEMENTS>
        <NV-DATA-INTERFACE>
          <SHORT-NAME>NVData_Interface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <NV-DATAS>
            <VARIABLE-DATA-PROTOTYPE>
              ...
            </VARIABLE-DATA-PROTOTYPE>
            ...
          </NV-DATAS>
        </NV-DATA-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 32: ARXML code – NVData interface structure

The name of the NVData interface definition is given by the `<SHORT-NAME>` element. The name is used within other elements that need to reference the interface type, for example, a software component may specify that it uses NVData interface `NVData_Interface`.

The short-name of an NVData interface should be a valid C identifier.

An NVData interface can be used to communicate non-volatile data using NVData elements, i.e. variable data prototypes, within the `<NV-DATAS>` element.

Variable Data Prototypes

Each NVData interface can specify zero or more NVData elements, or variable data prototypes, which constitute the AUTOSAR signals communicated over the interface. Each data element defines a prototype of a specific type and can be a primitive data type, a `RECORD` or an `ARRAY` type. See chapter 4, *Data Types*, on page 31 for details of defining data types.

To set up an NVData element in ASCET

1. Double-click `NVData_Interface`.

   `NVData_Interface` opens in the NVData interface editor.
2. Create an sdisc element named `Speed_NV`, as described on page 64.
3. Create the same implementation for `Speed_NV` as described on page 65.

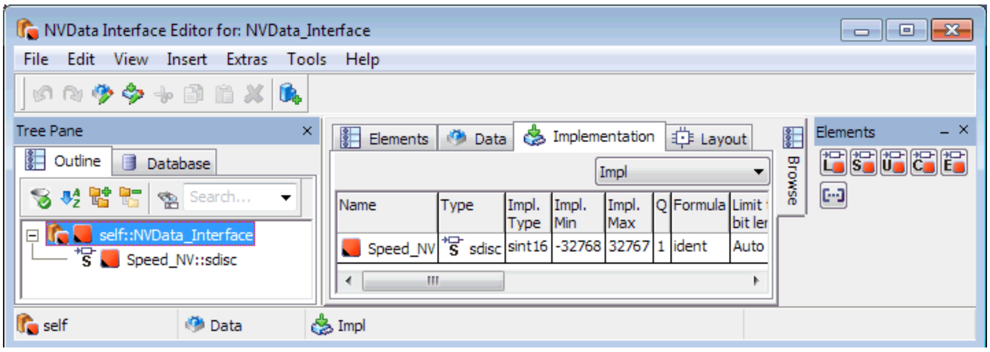


Figure 28: NVData element `Speed_NV` of the NVData interface `NVData_Inter-face` with implementation `Impl`

An implementation of an NVData interface in ASCET corresponds to an NVData interface in AUTOSAR. The NVData interface in configuration language is generated by ASCET in the file `Swc_interfaces.arxml`. The declaration of NVData elements within an NVData interface definition has the following structure:

```
<NV-DATA-INTERFACE>
  <SHORT-NAME>NVData_Interface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <NV-DATAS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>Speed_NV</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-ONLY
            </SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/SInt32</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </NV-DATAS>
</NV-DATA-INTERFACE>
```

Listing 33: ARXML code – declaration of NVData elements within NVData interface

An NVData element is defined using the `<VARIABLE-DATA-PROTOTYPE>` element, and all elements must be defined within an encapsulating `<NV-DATAS>` element.

Each `<VARIABLE-DATA-PROTOTYPE>` element must specify:

- the `<SHORT-NAME>` that you will use to refer to the item
- the `<SW-DATA-DEF-PROPS>` data properties, among them
  - the `<SW-ADDR-METHOD-REF>`
  - the `<SW-CALIBRATION-ACCESS>`
  - the `<SW-IMPL-POLICY>`
- a `<TYPE-TREF>` reference to the type of the data item

5.6 Implementations of Interfaces

The implementation editor for an AUTOSAR interface offers the following settings:

- **Interface represents an AUTOSAR service** option

  If this option is activated, ASCET sets the `IS-SERIVCE` element to true in the generated ARXML code (i.e. `<IS-SERVICE>true</IS-SERVICE>`). This tells an AUTOSAR RTE generator that the interface is to be used for communication between an application software component and a service component (e.g., an AUTOSAR service, ECU abstraction, or complex driver) located on the same ECU.

  For communication between application software components, this option must be deactivated.

- "AUTOSAR package name" field

  Allows to determine an AUTOSAR package name for the interface. The AUTOSAR package name must be of the following form:

  `/`*`<package>`*`/`*`<subpackage>`*`[/`*`<interface>`*`]`

  At least `/`*`<package>`*`/`*`<subpackage>`* must be given; otherwise, an error is issued during code generation.

  The semantic of the give package name is as follows:

  - If the name is empty, the general template for the associated interface kind, specified in the "ARXML Configuration Settings" window, will apply.
  - If the name is not empty, and of the proper form, it will be used as the package name (this includes the short-name) of the associated AUTOSAR interface.

  The given name must not contain any template parameter (i.e. %...%), but is taken as is.

- **Interface is defined externally** option

  If this option is activated, the interface is defined externally and will not be generated by ASCET, only referred to.

  > **NOTE**
  >
  > If **Interface is defined externally** is activated, the "AUTOSAR package name" field must not be empty. If it is, an error (`MMd16490`) is issued during code generation.

- "AUTOSAR Data Type Mapping Set" field

  Allows to determine a data type mapping set that will be added to the software component for data types used in an external interface.

  If the name is empty, the general template specified in the "ARXML Configuration Settings" window will apply.

6 Software Component Types

A software component is the atomic software unit of application in AUTOSAR. Software components interact through ports, which are typed interfaces. The interfaces control what can be communicated and the semantics of the communication.

To create an AUTOSAR software component

1. In the component manager, select **Insert > AUTOSAR > Software Component Block Diagram** or **Software Component ESDL**.
2. Name the software component `Swc`.
3. Follow the steps described in section 3.1.2, *Code Generation Settings for AUTOSAR*, on page 17 to create an AUTOSAR project `ARProject` and set the AUTOSAR code generation settings.
4. Insert the software component `Swc` in the project, as described on page 23.

To open a software component in an AUTOSAR project

1. In the component manager, double-click the `ARProject` project.

   The project editor window opens.
2. In the "Outline" tab of the project editor, double-click the `Swc` software component.

   The software component editor window opens.

Each software component must have its component type declared in the RTE generator's configuration. The component type makes the component available for composition into a larger software system. An application software component type is defined using the `<APPLICATION-SOFTWARE-COMPONENT-TYPE>` element in the `<`*`swc name`*`>.arxml` file.

```
<APPLICATION-SW-COMPONENT-TYPE>
   <SHORT-NAME>SWC</SHOET-NAME>
   <PORTS>
      ...
   </PORTS>
</APPLICATION-SW-COMPONENT-TYPE>
```

Listing 34: ARXML code – definition of application software component type

The software component type must be named using the `<SHORT-NAME>` element. The name must be system-wide unique; it is used within other elements to reference the software component type.

The short-name of a software-component must be a valid C identifier.

6.1 Ports

Ports provide the software component access to the interface. There are two classes of ports: provided ports (Pports) and required ports (Rports).

The ports of a software component are defined within the `<PORTS>` element.

```
<PORTS>
   <R-PORT-PROTOTYPE>
      ...
   </R-PORT-PROTOTYPE>
   <P-PORT-PROTOTYPE>
      ...
   </P-PORT-PROTOTYPE>
</PORTS>
```

Listing 35: ARXML code – port definition structure (all AUTOSAR versions)

Within the `<PORTS>` element, the `<P-PORT-PROTOTYPE>` and the `<R-PORT-PROTOTYPE>` elements are used to define provided ports and required ports respectively. When two components communicate, then typically both provided and required ports reference the same interface definition. This guarantees that they are compatible.

6.1.1 Provided Ports

Pports are used by a software component to provide data or services to other software components. Provided ports implement senders and servers.

6.1.1.1 Sender Port

To create a sender port

1. In the "Software Component Editor for: Swc", select **Insert › Component**.

   The "Select item..." window opens.

2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `SRInterface` and click **OK**.

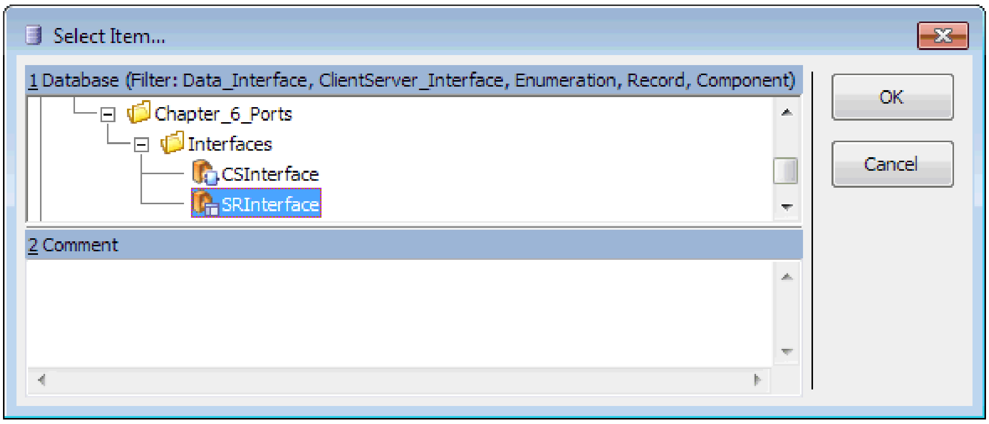


Figure 29: Selection of the item `SRInterface`

   The "Properties for complex element: SRInterface" window opens.

3. Name the Port `Sender`, activate **Provided** in the "Internal Access" area and click **OK**.

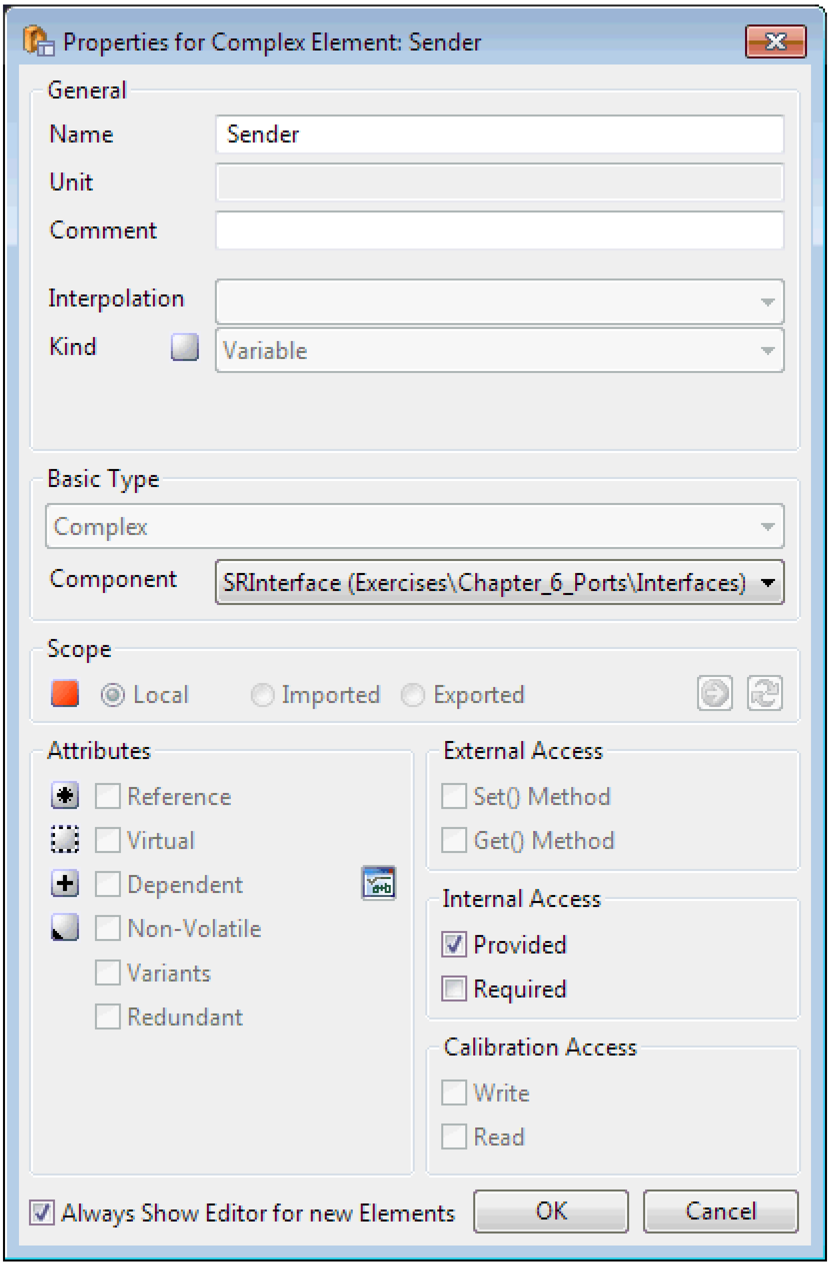


Figure 30: Provided port `Sender` of type `SRInterface`

4. If you are working in the block diagram editor for SWC, drag the element `Sender` from the "Outline" tab and drop it on the drawing area of the software component editor.

   The Pport `Sender` with elements `Speed` and `log` appears in the drawing area as follows.

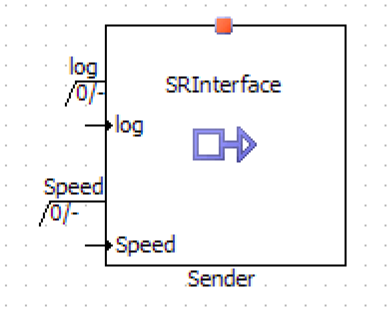


Figure 31: Pport `Sender` in the drawing area of the block diagram editor for software components

A provided port within a software component type definition is named using the `<SHORT-NAME>` element. The name is used within other elements to reference the port. The short-name of a provided port must be a valid C identifier.

Each provided port definition must specify the interface type over which it will communicate with other ports. This is done in the *`<swc name>`*`.arxml` file, using the `<PROVIDED-INTERFACE-TREF>` element. This `<PROVIDED-INTERFACE-TREF>` element must identify the required interface.

In addition, AUTOSAR R4.* requires the `<PROVIDED-COM-SPECS>` element that contains details about individual data elements, among them the following:

– `<DATA-ELEMENT-REF>` - which identifies the data element

– `<INIT-VALUE>` - which specifies the initial value of the data element

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_ComponentTypes</SHORT-NAME>
    <ELEMENTS>
        <APPLICATION-SW-COMPONENT-TYPE>
            <SHORT-NAME>SWC</SHORT-NAME>
            <PORTS>
                <P-PORT-PROTOTYPE>
                    <SHORT-NAME>Sender</SHORT-NAME>
                    <PROVIDED-COM-SPECS>
                        <NONQUEUED-SENDER-COM-SPEC>
                            <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
                            /ASCET_Interfaces/Impl/SRInterface/log</DATA-ELEMENT-REF>
                            <INIT-VALUE>
                                <NUMERICAL-VALUE-SPECIFICATION>
                                    <VALUE>FALSE</VALUE>
                                </NUMERICAL-VALUE-SPECIFICATION>
                            </INIT-VALUE>
                        </NONQUEUED-SENDER-COM-SPEC>
                        <NONQUEUED-SENDER-COM-SPEC>
                            <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
                            /ASCET_Interfaces/Impl/SRInterface/Speed</DATA-ELEMENT-REF>
                            <INIT-VALUE>
                                <NUMERICAL-VALUE-SPECIFICATION>
                                    <VALUE>0</VALUE>
                                </NUMERICAL-VALUE-SPECIFICATION>
                            </INIT-VALUE>
                        </NONQUEUED-SENDER-COM-SPEC>
                    </PROVIDED-COM-SPECS>
                    <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
                    /ASCET_Interfaces/Impl/SRInterface</PROVIDED-INTERFACE-TREF>
                </P-PORT-PROTOTYPE>
            </PORTS>
            ...
        </APPLICATION-SW-COMPONENT-TYPE>
    </ELEMENTS>
    ...
</AR-PACKAGE>
```

Listing 36: ARXML code – provided port `Sender` definition

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see Listing 36. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

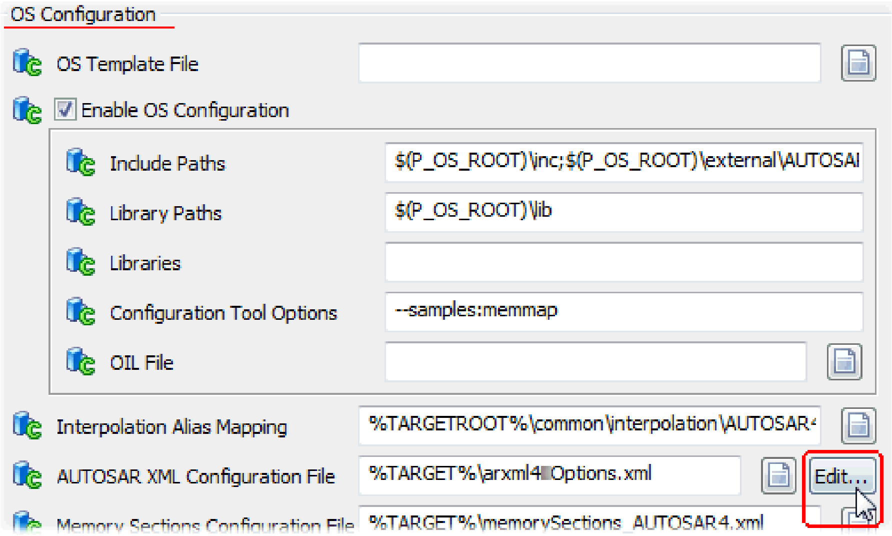
Boolean initial values `TRUE/FALSE` are generated as `0/1` when using the `<APPLICATION-VALUE-SPECIFICATION>`.

To select the specification means for initial values

The selection made here applies to data elements in `<PROVIDED-COM-SPECS>` and `<REQUIRED-COM-SPECS>` of sender-receiver and NVData interfaces, data elements in `<PARAMETER-REQUIRE-COM-SPEC>` of calibration interfaces, and interrunnable variables.

1. Open the parent project or default project of your SWC.
2. In the project editor, select **File › Properties**.

   The "Project Properties" dialog window opens.
3. In the "OS Configuration" node (cf. Figure 6 on page 21), click the **Edit** button to open the "ARXML Configuration Settings" dialog window.



4. In the "Miscellaneous" node of the "ARXML Configuration Settings" dialog window, do one of the following:
   - Activate **Generate Application Init Values** to select application value specification (cf. Listing 37 on page 92).
   - Deactivate **Generate Application Init Values** to select numerical value specification (cf. Listing 36 on page 89).

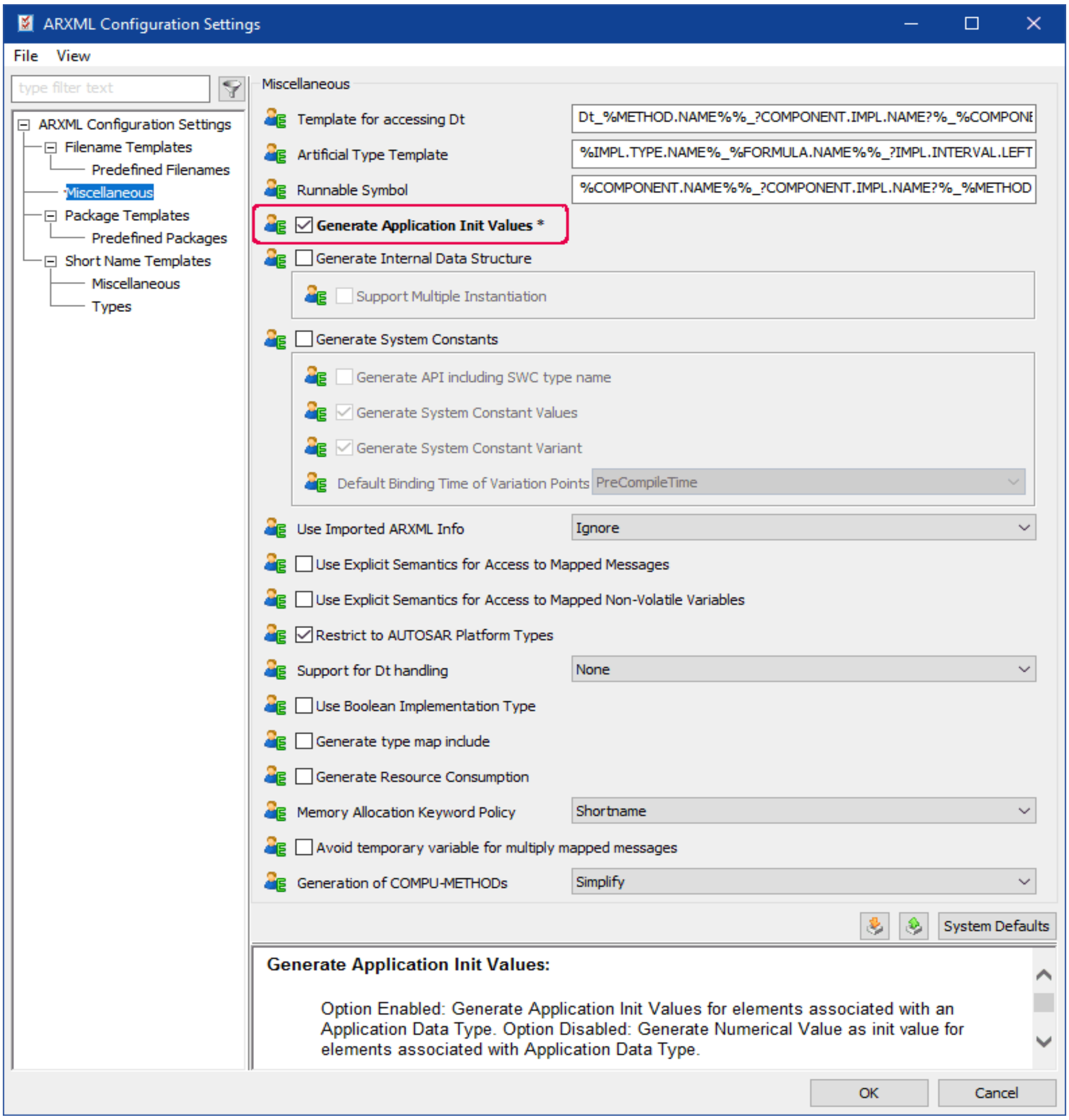


Figure 32: "ARXML Configuration Settings" window, "Miscellaneous" node

5. Close the "ARXML Configuration Settings " window with **OK**.
6. Close the "Project Properties" window.

> **NOTE**
>
> The changes in the "ARXML Configuration Settings" window are kept even if you leave the "Project Properties" window with **Cancel**.

7. Generate code for the project.

```
<P-PORT-PROTOTYPE>
    <SHORT-NAME>Sender</SHORT-NAME>
    <PROVIDED-COM-SPECS>
        <NONQUEUED-SENDER-COM-SPEC>
            <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_Interfaces/Impl/SRInterface/log</DATA-ELEMENT-REF>
            <INIT-VALUE>
                <APPLICATION-VALUE-SPECIFICATION>
                    <CATEGORY>VALUE</CATEGORY>
                    <SW-VALUE-CONT>
                    (A) <UNIT-REF DEST="UNIT">/ASCET_Units/NoUnit</UNIT-REF>
                        <SW-VALUES-PHYS>
                            <V>1</V>
                        </SW-VALUES-PHYS>
                    </SW-VALUE-CONT>
                </APPLICATION-VALUE-SPECIFICATION>
            </INIT-VALUE>
        </NONQUEUED-SENDER-COM-SPEC>
        <NONQUEUED-SENDER-COM-SPEC>
            <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_Interfaces/Impl/SRInterface/Speed</DATA-ELEMENT-REF>
            <INIT-VALUE>
                <APPLICATION-VALUE-SPECIFICATION>
                    <CATEGORY>VALUE</CATEGORY>
                    <SW-VALUE-CONT>
                    (B) <UNIT-REF DEST="UNIT">/ASCET_Units/kmqrh</UNIT-REF>
                        <SW-VALUES-PHYS>
                            <V>0</V>
                        </SW-VALUES-PHYS>
                    </SW-VALUE-CONT>
                </APPLICATION-VALUE-SPECIFICATION>
            </INIT-VALUE>
        </NONQUEUED-SENDER-COM-SPEC>
    </PROVIDED-COM-SPECS>
    <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
        /ASCET_Interfaces/Impl/SRInterface</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```

Listing 37: ARXML code – provided port `Sender` definition with `<APPLICATION-VALUE-SPECIFICATION>`

AUTOSAR requires the `<APPLICATION-VALUE-SPECIFICATION>` initial values to refer to a `UNIT`; see Listing 37 on page 92. A dummy unit named `ASCET_empty_unit` (see A in Listing 37) is created and referred to by all initial values with an empty unit specified at the transformation formula (see Figure 33) selected in the implementation editor of the element.

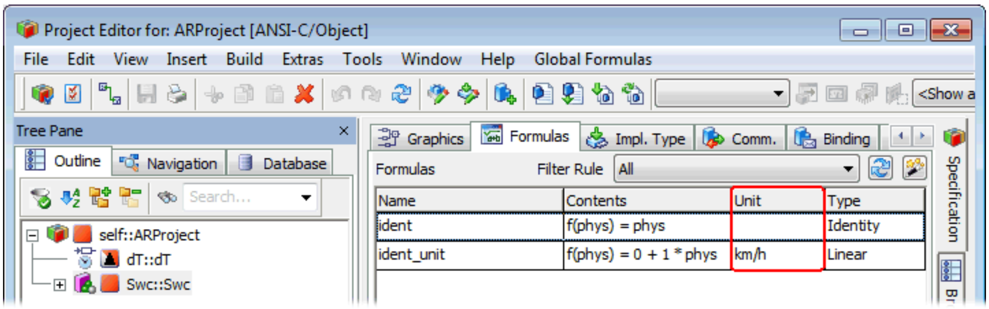


Figure 33: Project editor, "Formulas" tab

Units are defined in the `Swc_compumethods.arxml` file, in the `ASCET_Units` package.

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_Units</SHORT-NAME>
    <ELEMENTS>
        <UNIT>
            <SHORT-NAME>ASCET_empty_unit</SHORT-NAME>
            <DISPLAY-NAME>ASCET_empty_unit</DISPLAY-NAME>
        </UNIT>
        <UNIT>
            <SHORT-NAME>kmqrh</SHORT-NAME>
            <DISPLAY-NAME>km/h</DISPLAY-NAME>
        </UNIT>
    </ELEMENTS>
</AR-PACKAGE>
```

Listing 38: ARXML code – `Swc_compumethods.arxml` file, `ASCET_Units` definition

6.1.1.2 Server Port

To create a server port

1. In the "Software Component Editor for: Swc", select **Insert › Component**.

   The "Select item..." window opens.

2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `CSInterface` and click **OK**.

   The "Properties for complex element: CSInterface" opens.

3. Name the Port "`Server`", activate **Provided** in the "Internal Access" area, and click **OK**.

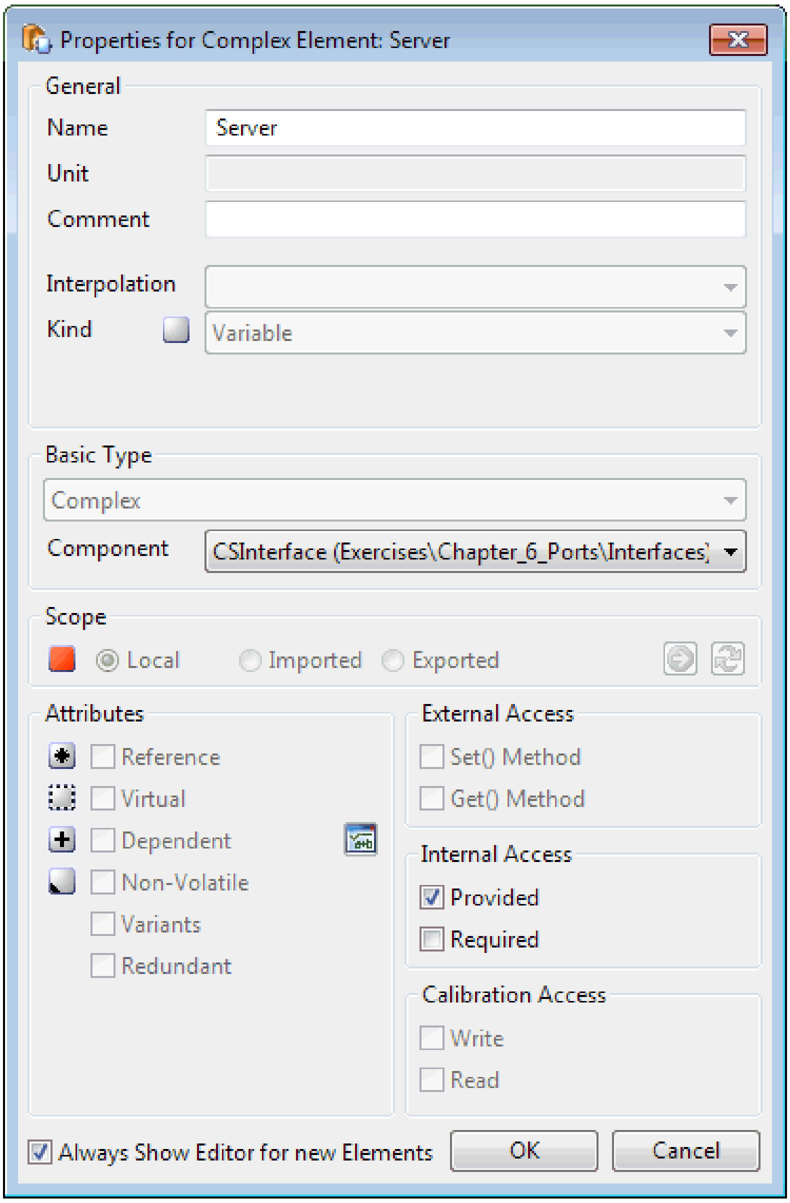


Figure 34: Provided port `Server` of type `CSInterface`

A message window opens and informs you that all graphical occurrences of the port will be removed when you set the port to **Provided**.

4. Confirm the message with **OK**.

   ASCET creates the following items:

   - a server node named `Server::CSInterface` under the folder "`Realized Interfaces`"
   - a diagram `Server_CSInterface`
   - a server runnable for each operation in the client-server interface `CSInterface`

     In Figure 35 below, ASCET has created the runnables `Server_MaximumValue` and `Server_Notification`.

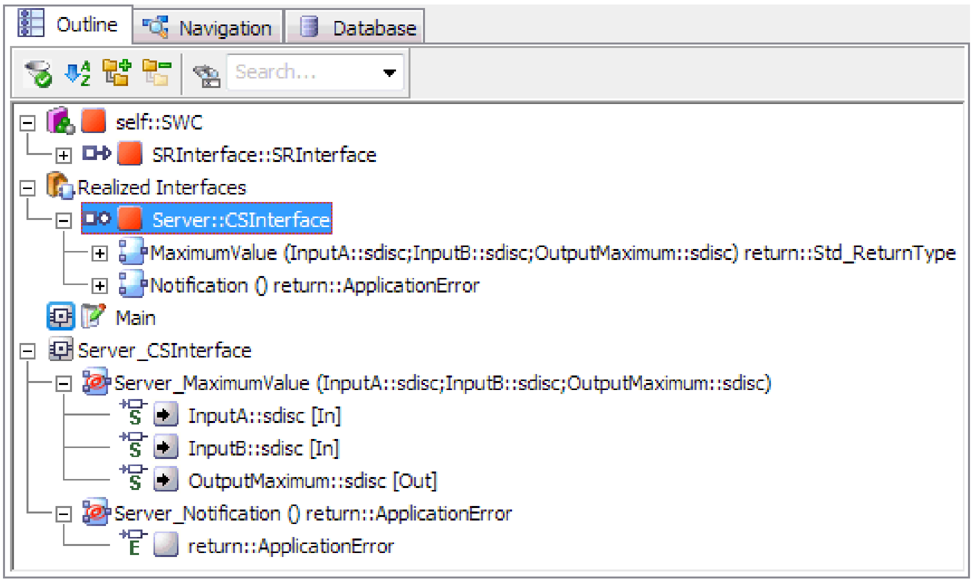


Figure 35: Pport `Server` in the "Outline" tab of the software component `Swc`

The entry function of the server runnable has a return type of `void` or `Std_ReturnType`, depending on whether or not the server returns an application error.

The provided port must specify the interface type over which it will communicate with other ports using the `<PROVIDED-INTERFACE-TREF>`. This `<PROVIDED-INTERFACE-TREF>` element must identify the required interface.

```
<P-PORT-PROTOTYPE>
   <SHORT-NAME>Server</SHORT-NAME>
   <PROVIDED-COM-SPECS>
     ...
   </PROVIDED-COM-SPECS>
   <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
       /ASCET_Interfaces/Impl/CSInterface</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```

Listing 39: ARXML code – provided port `Server` definition

Furthermore, ASCET provides additional information to the internal behavior of the software component `Swc`. On the one hand, one operation-invoked event is created for each operation in the server port. On the other hand, a runnable entity is created for each operation in the server port. Refer to chapter 7, *Internal Behavior*, on page 105 for more detailed information.

> ⓘ **NOTE**
>
> A client-server interface might be edited once a server is inserted in a software component. In this case, you must update the server interface in the software component using the menu option **Build › Update Interfaces**.

6.1.2 Required Ports

Rports are used by a software component to require data or services from other software components. Required ports implement receivers, clients, calibration ports, and NVData ports.

The definition of a required port is identical to that of a provided port, with the exception that the `<R-PORT-PROTOTYPE>` element is used.

6.1.2.1 Receiver Port

To create a receiver port

1. In the "Software Component Editor for: Swc", select **Insert > Component**.

   The "Select item..." window opens.
2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `SRInterface` and click **OK**.

   The "Properties for complex element: SRInterface" window opens.
3. Name the Port `Receiver`, activate **Required** in the "Internal Access" area and click **OK**.

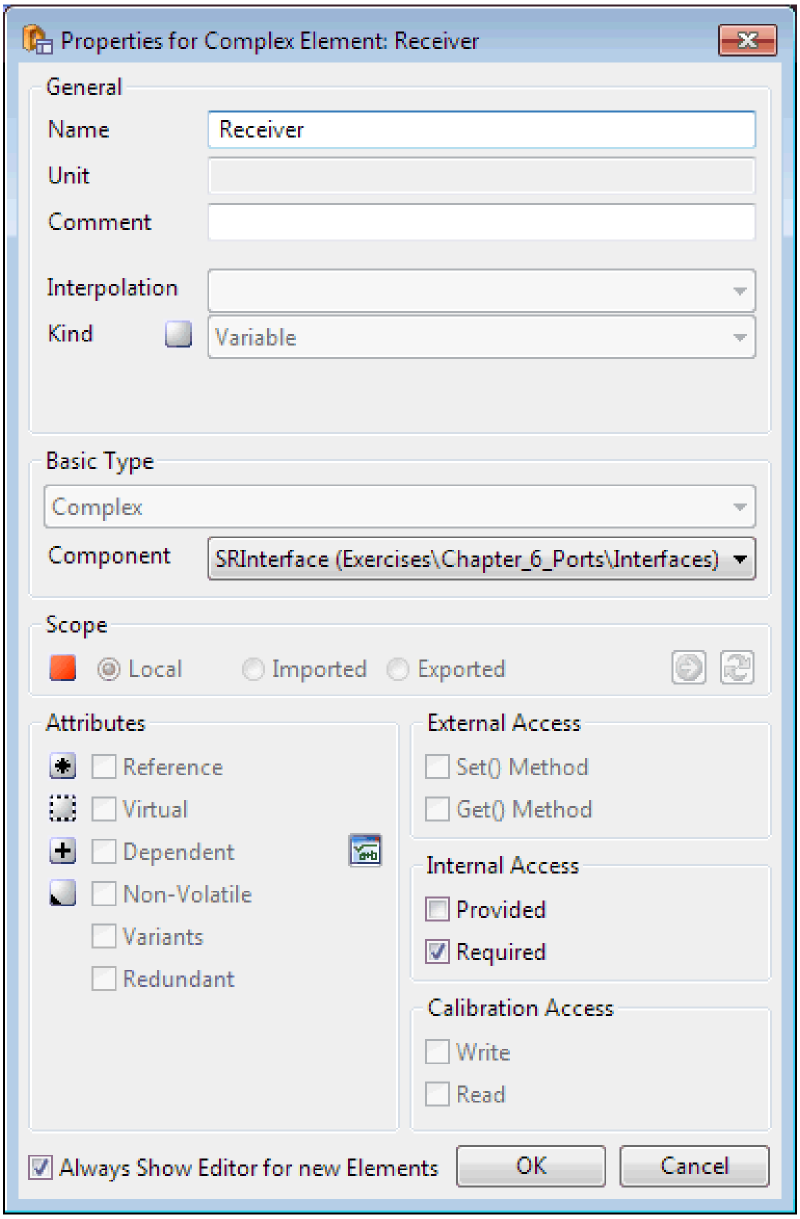


Figure 36: Required port `Receiver` of type `SRInterface`

4. If you are working in the block diagram editor for SWC, drag the element `Receiver` from the "Outline" tab and drop it on the drawing area of the software component editor.

   The Rport `Receiver` with element `Speed` appears in the drawing area as follows.

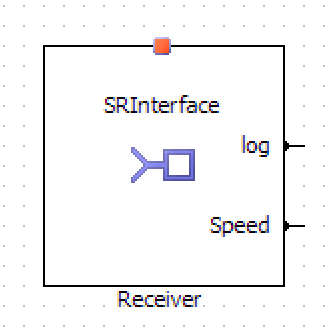


Figure 37: Rport `Receiver` in the drawing area of the block diagram editor for software components

A required port within a software component type definition is named using the `<SHORT-NAME>` element. The name is used within other elements to reference the software component type. The short-name of a required port must be a valid C identifier.

The required port definition must reference an interface definition defined using the `<REQUIRED-INTERFACE-TREF>` element.

In addition, AUTOSAR R4.* requires the `<REQUIRED-COM-SPECS>` element that contains details about individual data elements, e.g.,

- `<DATA-ELEMENT-REF>` - which identifies the data element,
- `<INIT-VALUE>` - which specifies the initial value of the data element,

and others.

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Receiver</SHORT-NAME>
   <REQUIRED-COM-SPECS>
      <NONQUEUED-RECEIVER-COM-SPEC>
         <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_Interfaces/Impl/SRInterface/log</DATA-ELEMENT-REF>
         <INIT-VALUE>
            <NUMERICAL-VALUE-SPECIFICATION>
               <VALUE>FALSE</VALUE>
            </NUMERICAL-VALUE-SPECIFICATION>
         </INIT-VALUE>
      </NONQUEUED-RECEIVER-COM-SPEC>
      <NONQUEUED-RECEIVER-COM-SPEC>
         <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_Interfaces/Impl/SRInterface/Speed</DATA-ELEMENT-REF>
         <INIT-VALUE>
            <NUMERICAL-VALUE-SPECIFICATION>
               <VALUE>0</VALUE>
            </NUMERICAL-VALUE-SPECIFICATION>
         </INIT-VALUE>
      </NONQUEUED-RECEIVER-COM-SPEC>
   </REQUIRED-COM-SPECS>
   <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
   /ASCET_Interfaces/Impl/SRInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Listing 40: ARXML code – required port `Receiver` definition

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see Listing 36. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values on page* 89 for an instruction how to select the specification means.

6.1.2.2 Client Port

To create a client port

1. In the "Software Component Editor for: Swc", select **Insert › Component**.

   The "Select item..." window opens.

2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `CSInterface` and click **OK**.

   The "Properties for complex element: CSInterface“ editor opens.

3. Name the Port "`Client`", activate **Required** in the "Internal Access" area and click **OK**.

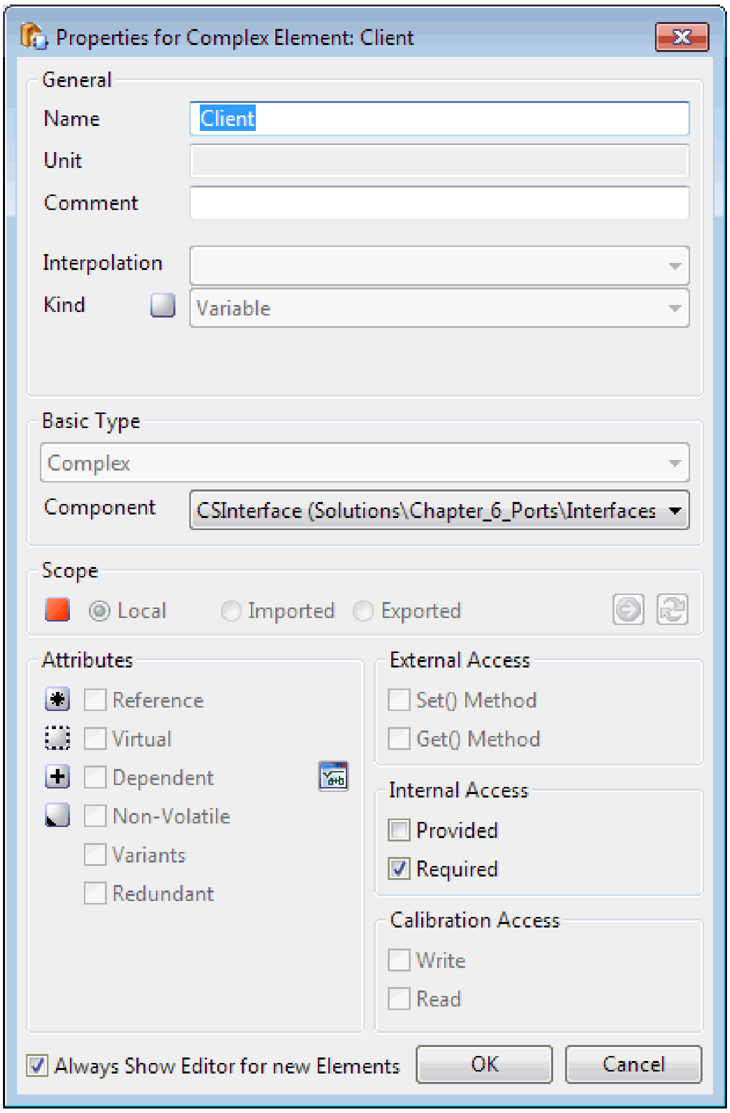


Figure 38: Required port `Client` of type `CSInterface`

4. In the block diagram editor for SWC, do also the following:
   i. Drag the element `Client` from the "Outline" tab and drop it on the drawing area of the software component editor.
   ii. If necessary, activate flexible layout for the interface `CSInterface`:
      a. Go to the component manager.
      b. In the "Database" or "Workspace" pane, right-click `CSInterface` and select **Flexible Class Layout › Activate** from the context menu.
      c. In the "Change Flexible Class Layout State" window, select `CSInterface` and click **OK**.
   iii. In the drawing area, right-click the `Client` port and select **Ports › Methods** from the context menu.

      The "Port Editor <CSInterface>" window opens.
   iv. Deactivate the method `Notification` and click **OK**.

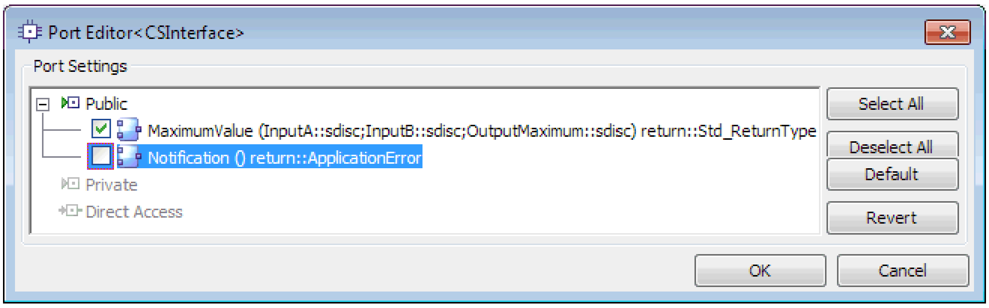


Figure 39: Port editor window to select/deselect methods

v. Resize the `Client` block and reposition the pins.

The Rport `Client` with operation `MaximumValue` appears as follows (or similar) in the drawing area.

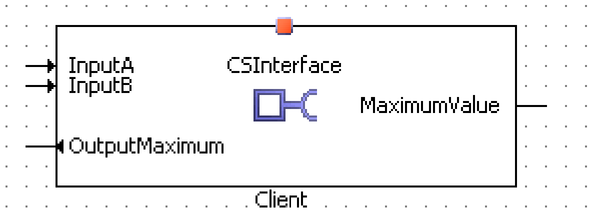


Figure 40: Rport `Client` in the drawing area of the block diagram editor for software components

The required port definition must reference an interface definition defined using the `<REQUIRED-INTERFACE-TREF>` element. In addition, the `<REQUIRED-COM-SPECS>` element is required, which contains details about individual operations.

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Client</SHORT-NAME>
   <REQUIRED-COM-SPECS>
      <CLIENT-COM-SPEC>
         <OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
             /ASCET_Interfaces/Impl/CSInterface/MaximumValue
             </OPERATION-REF>
      </CLIENT-COM-SPEC>
      ...
   </REQUIRED-COM-SPECS>
   <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
       /ASCET_Interfaces/Impl/CSInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Listing 41: ARXML code – required port `Client` definition

6.1.2.3 Calibration Port

To create a calibration port

1. In the "Software Component Editor for: Swc", select **Insert › Component**.

   The "Select item..." window opens.

2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `CalInterface` and click **OK**.

The "Properties for complex element: CalInterface" window opens.

3. Name the Port `Calibration` and click **OK**.

   The "Internal Access" is set to **Required**; it cannot be changed.

4. If you are working in the block diagram editor for SWC, drag the element `Calibration` from the "Outline" tab and drop it on the drawing area of the software component editor.

   The Rport `Calibration` with elements `CalParam1`, `CalParam2` and `CalParam3` appears in the drawing area as follows.

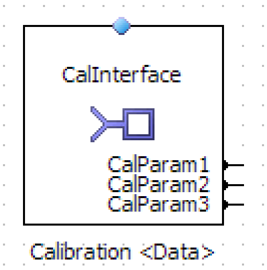


Figure 41: Rport `Calibration` in the drawing area of the block diagram editor for software components

The required port definition must reference an interface definition defined using the `<REQUIRED-INTERFACE-TREF>` element.

In addition, AUTOSAR R4.* requires the `<REQUIRED-COM-SPECS>` element that contains one `<PARAMETER-REQUIRE-COM-SPEC>` element for each parameter interface with details about the respective parameter, e.g.,

– `<PARAMETER-REF>` - which identifies the parameter,

– `<INIT-VALUE>` - which specifies the initial value of the parameter,

and others.

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Calibration</SHORT-NAME>
   <REQUIRED-COM-SPECS>
      <PARAMETER-REQUIRE-COM-SPEC>
         <INIT-VALUE>
            <NUMERICAL-VALUE-SPECIFICATION>
               <VALUE>FALSE</VALUE>
            </NUMERICAL-VALUE-SPECIFICATION>
         </INIT-VALUE>
         <PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">
         /ASCET_Interfaces/Impl/CalInterface/CalParam1</PARAMETER-REF>
      </PARAMETER-REQUIRE-COM-SPEC>
      ...
   </REQUIRED-COM-SPECS>
   <REQUIRED-INTERFACE-TREF DEST="PARAMETER-INTERFACE">
   /ASCET_Interfaces/Impl/CalInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Listing 42: ARXML code – required port `Calibration` definition

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see Listing 36. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values* on page 89 for an instruction how to select the specification means.

6.1.2.4 NVData Port

To create an NVData port

1. In the "Software Component Editor for: Swc", select **Insert > Component**.

   The "Select item..." window opens.

2. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `NVData_Interface` and click **OK**.

   The "Properties for complex element: NVData_Interface" window opens.

3. Name the Port `NVData` and click **OK**.

   The "Internal Access" is set to **Required**; it cannot be changed.

4. If you are working in the block diagram editor for SWC, drag the element `NVData` from the "Outline" tab and drop it on the drawing area of the software component editor.

   The Rport `NVData` with element `Speed_NV` appears in the drawing area as follows.

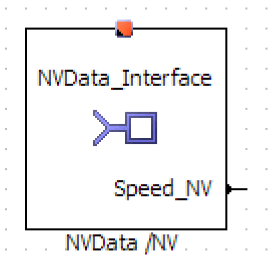


Figure 42: Rport `NVData` in the drawing area of the block diagram editor for software components

AUTOSAR R4.* requires the `<REQUIRED-COM-SPECS>` element that contains one `<NV-REQUIRE-COM-SPEC>` element for each NVData element with details about the respective NVData element, among them the following:

– `<VARIABLE-REF>` - which identifies the NVData element
– `<INIT-VALUE>` - which specifies the initial value of the NVData element

```
<R-PORT-PROTOTYPE>
  <SHORT-NAME>NVData</SHORT-NAME>
  <REQUIRED-COM-SPECS>
    <NV-REQUIRE-COM-SPEC>
      <INIT-VALUE>
        <NUMERICAL-VALUE-SPECIFICATION>
          <VALUE>0</VALUE>
        </NUMERICAL-VALUE-SPECIFICATION>
      </INIT-VALUE>
      <VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
      /ASCET_Interfaces/Impl/NVData_Interface/Speed_NV</VARIABLE-REF>
    </NV-REQUIRE-COM-SPEC>
  </REQUIRED-COM-SPECS>
  <REQUIRED-INTERFACE-TREF DEST="NV-DATA-INTERFACE">
  /ASCET_Interfaces/Impl/NVData_Interface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Listing 43: ARXML code – required port `NVData` definition

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see Listing 36. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values* on page 89 for an instruction how to select the specification means.

7 Internal Behavior

The internal behavior of a software component defines how the code that implements the component interacts with the ports. In this chapter, you will see how to configure the internal behavior.

Internal behavior elements are used to define how the software component will interact with the RTE at runtime. The internal behavior of a software component specifies:

- The runnable entities that belong to the software component and how they interact (if at all) with the ports of the software component.
- The events that cause runnable entities to be activated at runtime.
- The interrunnable variables used for communication between the runnables of a software component.
- The exclusive areas that exist so runnable entities can execute all or part of their code in mutual exclusion from other runnable entities.

Each internal behavior description is applicable to a single software component and therefore must reference the software component type to which it applies. In AUTOSAR R4.*, the reference is established in the <*SWC name*>.arxml file, using the <DATA-TYPE-MAPPING-REF> element.

```
<INTERNAL-BEHAVIORS>
   <SWC-INTERNAL-BEHAVIOR>
      <SHORT-NAME>bSwc</SHORT-NAME>
      <DATA-TYPE-MAPPING-REFS>
         <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
         /ASCET_Mappings/DataMappings/Impl</DATA-TYPE-MAPPING-REF>
      </DATA-TYPE-MAPPING-REFS>
      <EXCLUSIVE-AREAS>
         ...
      </EXCLUSIVE-AREAS>
      <EVENTS>
         ...
      </EVENTS>
      <EXPLICIT-INTER-RUNNABLE-VARIABLES>
         ...
      </EXPLICIT-INTER-RUNNABLE-VARIABLES>
      <IMPLICIT-INTER-RUNNABLE-VARIABLES>
         ...
      </IMPLICIT-INTER-RUNNABLE-VARIABLES>
      <RUNNABLES>
         ...
      </RUNNABLES>
   </SWC-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
```

Listing 44: ARXML code – internal behavior description for Swc

An internal behavior must be named using the <SHORT-NAME> element. The name is used within other elements to reference the behavior. ASCET automatically names the internal behavior of a software component with a prefix b followed by the name of the software component.

The short-name of an internal behavior does not need to be a valid C identifier (but it must pass the syntactic checks enforced by the XML Schema).

The following sections first outline the basic framework for events and runnable entities before showing how to configure the RTE to achieve different types of runnable entity/interface interaction.

7.1 Events

Events control how runnable entities are triggered by the generated RTE at runtime. ASCET V6.4 supports the following events:

- TIMING-EVENT – activates a runnable entity periodically. The <TIMING-EVENT> allows you to execute a runnable entity to poll an Rport to check if data has been received, periodically call a server (i.e. be a client), periodically send data on a Pport, or simply to execute some internal software component functionality. Runnable entities that are activated in response to a timing event are said to be *time-triggered*.
- OPERATION-INVOKED-EVENT – activates a runnable entity to handle a server call for an operation on a provided port characterized by a client-server interface.
- MODE-SWITCH-EVENT – activates a runnable entity on either entry to, or exit from, an application mode.

The structure for specifying events is similar to the structure shown in Listing 45. The actual sequence of events is determined by the event names.

```
<EVENTS>
   <TIMING-EVENT>
      ...
   </TIMING-EVENT>
   <OPERATION-INVOKED-EVENT>
      ...
   </OPERATION-INVOKED-EVENT>
   <SWC-MODE-SWITCH-EVENT>
      ...
   </SWC-MODE-SWITCH-EVENT>
</EVENTS>
```

Listing 45: ARXML code – structure for event specification

An event can be used to activate a runnable entity when the event occurs. An event references the runnable entity that is to be activated when the event occurs.

7.1.1 Timing Events

A <TIMING-EVENT> is used to indicate that a runnable entity will be activated periodically by the operating system. The RTE generator will use this information to generate an appropriate schedule table that must be ticked from application code.

To create a timing event

1. In the "Software Component Editor for: Swc", go to the "Event Specification" tab.
2. Select **Event > Add Event** and name the event `Cyclic_10ms`.
3. In the "Event Kind" combo box, select `Timing`.
4. In the "Period" field, enter a period of 0.01 seconds.

   The timing event `Cyclic_10ms` appears in the "Event Specification" tab as follows.

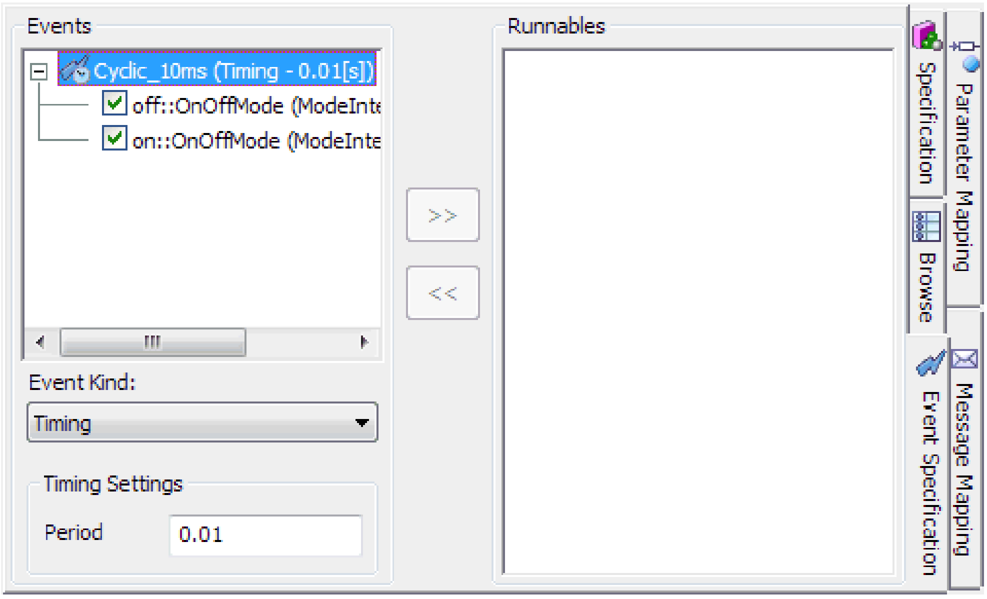


Figure 43: Definition of the timing event `Cyclic_10ms`

ASCET enables the user to specify the modes in which this timing event shall be activated. For the use of application modes refer to chapter 8, *Modes*, on page 165.

When the timing event is mapped to a runnable entity (see section 7.2, *Runnable Entities*, on page 110), then ASCET generates the `<TIMING-EVENT>` element in the configuration language:

```
<EVENTS>
   <TIMING-EVENT>
      <SHORT-NAME>Cyclic_10ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
         /ASCET_ComponentTypes/Swc/bSwc/RunnableEntity
      </START-ON-EVENT-REF>
      <PERIOD>0.01</PERIOD>
   </TIMING-EVENT>
   ...
</EVENTS>
```

Listing 46: ARXML code – definition of a timing event (all AUTOSAR versions)

A timing event must be named using the <SHORT-NAME> element. The name is used within other elements to reference the timing event. The short-name of a timing event does not need to be a valid C identifier.

The <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY"> element defines the runnable entity that is to be activated when the event occurs. The <PERIOD> element specifies the time raster in seconds to be used by the RTE generator.

7.1.2 Operation-Invoked Events

Operation-Invoked events are automatically inserted in an ASCET software component when you create a server port (see section 6.1.1, *Provided Ports*, on page 86 for how to create a server port).

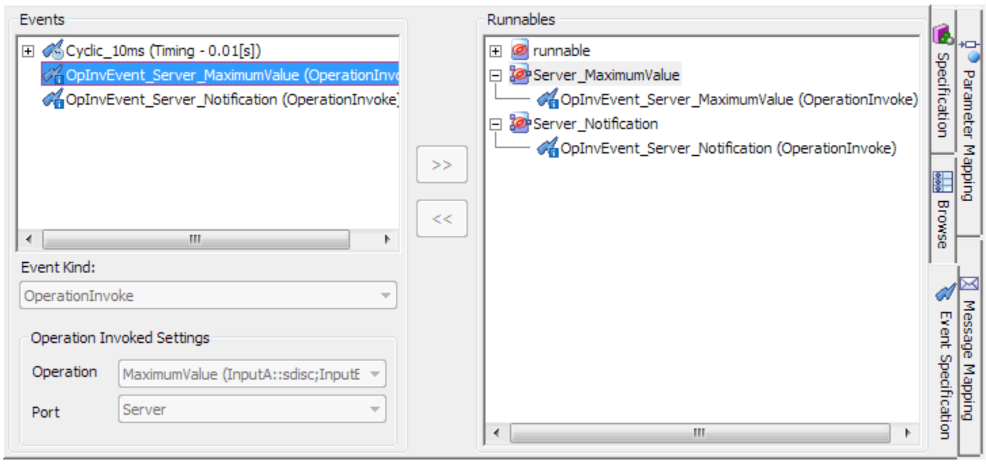


Figure 44: Operation-Invoked event for the server operations `MaximumVal` and `Notification`

An Operation-Invoked event is defined using the <OPERATION-INVOKED-EVENT> element. Each <OPERATION-INVOKED-EVENT> element must specify:

– the <SHORT-NAME> to refer to the event, which can be edited manually in ASCET by the user
– a <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY"> reference to the runnable entity
– an <OPERATION-IREF> reference to the operation prototype and server port

The Operation-Invoked event for the operation `MaximumValue` is defined in the configuration language as follows:

```
<OPERATION-INVOKED-EVENT>
   <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_ComponentTypes/SWC/bSWC/Server_MaximumValue
   </START-ON-EVENT-REF>
   <OPERATION-IREF>
      <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
      /ASCET_ComponentTypes/SWC/Server</CONTEXT-P-PORT-REF>
      <TARGET-PROVIDED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
      /ASCET_Interfaces/Impl/CSInterface/MaximumValue
      </TARGET-PROVIDED-OPERATION-REF>
   </OPERATION-IREF>
</OPERATION-INVOKED-EVENT>
```

Listing 47: ARXML code – definition of an Operation-Invoked event

7.1.3 Mode-Switch Events

Mode-switch events activate a runnable entity on entry to, or exit from, an application mode.

To create a mode-switch event

1. In the "Software Component Editor for: Swc", go to the **Event Specification** tab.
2. Select **Event > Add Event** and name the event `ModeEvent`.
3. In the "Event Kind" combo box, select `ModeSwitch`.
4. Set the following mode switch settings:
   - Activation: **entry**
   - Assigned Mode: `on::OnOffMode`

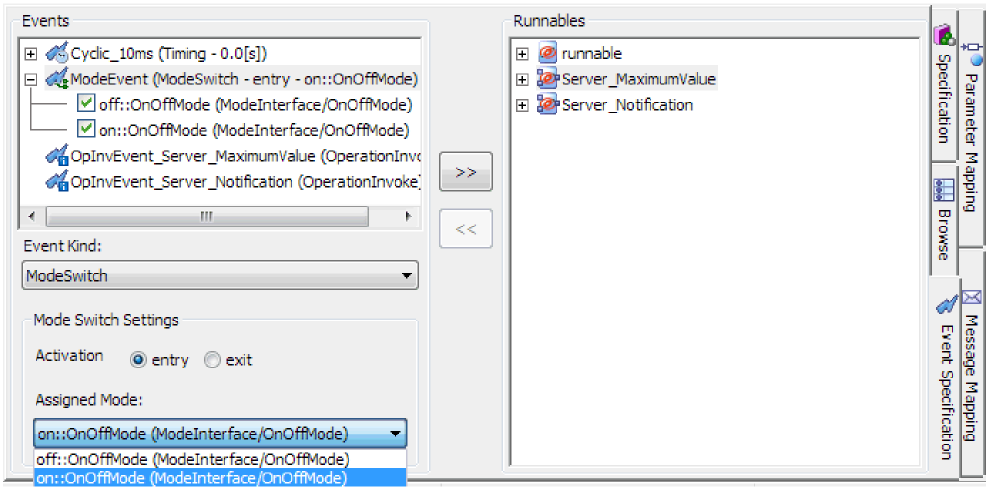


Figure 45: Modeling `ModeEvent` on **entry** with mode `on` of the application mode `OnOffMode`

When the mode-switch event is mapped to a runnable entity (see section 7.3, *Responding to Timing Events*, on page 113), then ASCET generates the `<SWC-MODE-SWITCH-EVENT>` element in the configuration language:

```
<SWC-MODE-SWITCH-EVENT>
   <SHORT-NAME>ModeEvent</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_ComponentTypes/SWC/bSWC/ModeRunnable
   </START-ON-EVENT-REF>
   <ACTIVATION>ON-ENTRY</ACTIVATION>
   <MODE-IREFS>
      <MODE-IREF>
         <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/ModeInterface
         </CONTEXT-PORT-REF>
         <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
         "MODE-DECLARATION-GROUP-PROTOTYPE">
         /ASCET_Interfaces/Impl/ModeInterface/OnOffMode
         </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
         <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
         /ASCET_Types/ApplicationDataTypes/OnOffMode/on
         </TARGET-MODE-DECLARATION-REF>
      </MODE-IREF>
   </MODE-IREFS>
</SWC-MODE-SWITCH-EVENT>
```

Listing 48: ARXML code – definition of a Mode-Switch event

In the "Events" field, all modes in the assigned mode group are shown below the Mode-Switch event. They can be enabled/disabled individually. If at least one mode is deactivated (see Figure 71 on page 170), the <DISABLED-MODE-IREFS> element is added to the configuration language, with one <DISABLED-MODE-IREF> element for each deactivated mode.

For ARXML code examples, see Listing 90 on page 171.

See section 8.3.3, *Disabling Modes*, on page 170 for more information on disabled modes.

A Mode-Switch event must be named using the <SHORT-NAME> element. The name is used within other elements to reference the timing event. The short-name of a timing event does not need to be a valid C identifier.

Each <MODE-SWITCH-EVENT> element must specify the following:

- a <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY"> reference to the runnable entity
- an <ACTIVATION> value, ENTRY or EXIT, for the activation type
- a <MODE-IREF> element, which defines the mode associated with the event
- optionally, a <MODE-DEPENDENCY> reference to a mode declaration

7.2 Runnable Entities

A runnable entity, or simply *runnable*, is a piece of code in a software component that is triggered by the RTE at runtime. A software component comprises one or more runnables, and each runnable must have a unique handle so that the RTE can access it at runtime.

To create a runnable entity

1. In the "Software Component Editor for: Swc", select a diagram (e.g., `Main`) in the "Outline" tab.
2. Select **Insert › Runnable** and name it `RunnableEntity`.

All runnable entities must be defined in the Software Component Template within the `<RUNNABLES>` definition in an `<SWC-INTERNAL-BEHAVIOR>` definition.

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-ENTER-EXCLUSIVE-AREA-REFS>
      ...
   </CAN-ENTER-EXCLUSIVE-AREA-REFS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
      /ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <SYMBOL>Swc_Impl_RunnableEntity</SYMBOL>
   ...
</RUNNABLE-ENTITY>
```

Listing 49: ARXML code – runnable entity definition (AUTOSAR R4.2.2)

A `<RUNNABLE-ENTITY>` must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the runnable entity.

The `<SHORT-NAME>` denotes the name of the runnable entity in the XML namespace, but it does not tell the RTE what the associated function body you will provide in your code is called. This information is provided by the `<SYMBOL>` declaration that links the runnable entity to the C function name you will use in your implementation. The `<SYMBOL>` name must be a valid C identifier.

In AUTOSAR R4.*, the `<SW-ADDR-METHOD-REF>` element is used to determine the memory class for the generated code.

The symbol of a runnable entity is optional information in ASCET. If not defined, ASCET takes the name of the function in the ASCET-generated code that implements the runnable entity. In the example, ASCET generates the C function `SWC_IMPL_RunnableEntity`. If the symbol is defined, then ASCET generates C code for the runnable entity according to the given symbol.

To set the C identifier for a runnable

1. In the "Outline" tab of the software component editor, select the runnable `RunnableEntity` and select **Edit › Implementation**.

   The window "Implementation for: RunnableEntity" opens.
2. Enter the symbol `RteRunnable_Swc_RunnableEntity`.
3. Click **OK**.

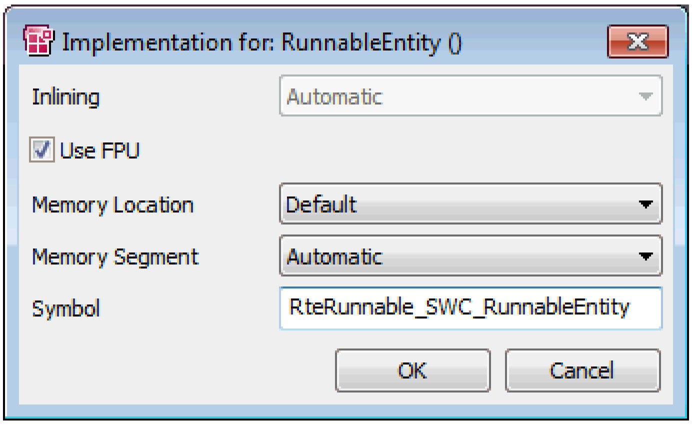


Figure 46: Setting the symbol `RteRunnable_Swc_RunnableEntity` for the runnable `RunnableEntity`

With that, ASCET will generate C code for the implemented runnable and name it `RteRunnable_Swc_RunnableEntity` (see file `Swc.c` in this example):

```
FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   ...
}
```

The `<RUNNABLE-ENTITY>` description is generated accordingly:

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-ENTER-EXCLUSIVE-AREA-REFS>
      ...
   </CAN-ENTER-EXCLUSIVE-AREA-REFS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
      /ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
   ...
</RUNNABLE-ENTITY>
```

Listing 50: ARXML code – runnable entity definition with user-defined `<SYMBOL>` (AUTOSAR R4.2.2)

This declaration is sufficient if your runnable entity does not need to interact with the software component's ports. However, if a runnable entity needs to communicate through ports, then you need to specify additional information that allows the RTE generator to generate APIs to allow interaction to take place, for example:

A. What data items the runnable entity can send.

B. What data items the runnable entity can receive.

C. Which servers the runnable entity calls and how it expects the result to be returned.

You can use the same runnable entity to receive data on one port and send data on another port, or to receive data on a port and then call a server port to process the received data. For example, you may create a runnable entity that reads an integer value from an Rport, multiplies it by two and sends it out on a Pport.

A runnable entity that is not invoked by an Operation-Invoked event can also specify a minimum start interval to control the rate at which activations occur. A minimum start interval will delay the activation of a runnable to prevent that the runnable is started more than once within the interval.

> **NOTE**
>
> When using minimum start intervals, check how the runnable activation is implemented by the RTE generator in use.

7.3 Responding to Timing Events

A runnable entity is executed periodically at runtime when the runnable entity is associated with a timing event. Timing events specify how often the runnable entities should execute.

The `<TIMING-EVENT>` element specifies the `<PERIOD>` of occurrence in seconds and must reference a runnable entity defined in the component's internal behavior using a `<START-ON-EVENT-REF>` element. A period of zero is illegal.

The following example shows how to configure the RTE to activate a runnable entity every 10 milliseconds.

To assign a timing event to a runnable

1. Go to the "Event Specification" tab of the "Software Component Editor for: Swc".
2. In the "Events" field, select the event `Cyclic_10ms`.
3. In the "Runnables" field, select the runnable `RunnableEntity`.
4. Select **Event > Assign Event** or click the **>>** button.

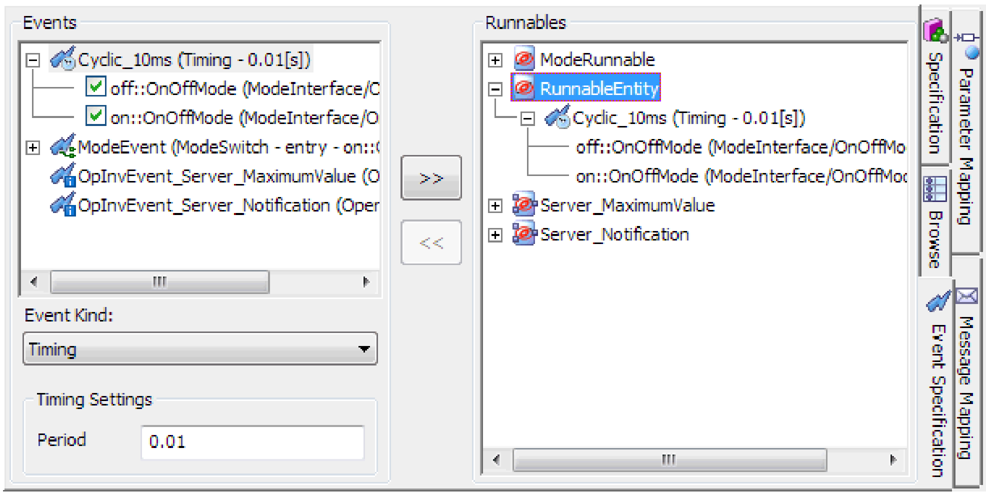


Figure 47: The event `Cyclic_10ms` is assigned to `RunnableEntity`

In the `<TIMING-EVENT>` element, the `<START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">` element defines the runnable entity that is to be activated when the event occurs. The `<PERIOD>` element specifies the time raster to be used by the RTE generator.

A timing event must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the timing event. The short-name of a timing event does not need to be a valid C identifier.

See also Listing 46: *ARXML code – definition of a timing event (all AUTOSAR versions)*, on page 107.

7.4 Sending to a Port

If your software component provides a sender-receiver interface, you must define at least one runnable entity that sends data over the interface.

The runnable can send data in two ways:

- *Explicitly*, in which case the RTE generates an explicit API call that may be optimized to a macro. The sent datum may be either queued or unqueued.
- *Implicitly*, in which case the RTE generates an implicit API call that will be optimized to a macro. The sent datum must not be queued.

For senders, it does not matter how the runnable entity is triggered, so any event can be used to activate the runnable entity.

7.4.1 Explicit Communication

To send to a port with explicit communication

1. Add a Pport `Sender` to `Swc`, as described in section *To create a sender port* on page 86.
2. Insert a literal with value 120.
3. Choose the runnable **RunnableEntity** in the tree pane.
4. If you are working in the *block diagram editor* for SWC, proceed as follows:

i. Drag the Pport `Sender` from the "Outline" tab and drop it in the drawing area of the software component editor.

ii. Use the **RTE Access** button to create an RTE Access operator and place it in the drawing area.

iii. Connect the output of the RTE Access operator with the data element `Speed` of the `Sender` port.

iv. Double-click the sequence call of `Speed`.

ASCET automatically assigns a sequence number for the sending of the data element `Speed` within the runnable `RunnableEntity`, i.e. the sequence 5.

v. Connect the literal to the input of the RTE Access operator.

5. If you are working in the *ESDL editor* for SWC, proceed as follows:

i. Enter the following code send to a port without status inquiry:

```
Sender.speed.explicitWrite(120);
```

You can now generate code (see *To generate code in a project* on page 23).

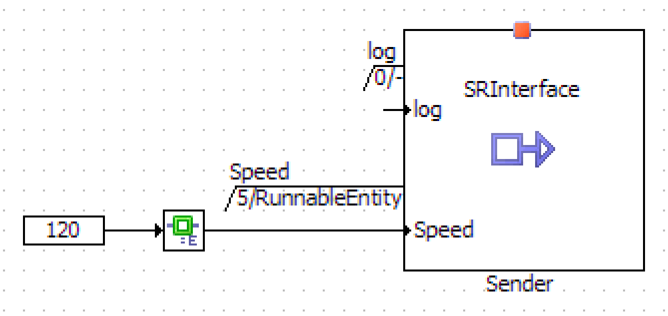


Figure 48: Sending a value 120 to a sender port with explicit communication (block diagram editor for SWC)

Runnable entities sending data with explicit communication must define a `<DATA-SEND-POINTS>` element that specifies the data items that will be sent for a given interface.

In AUTOSAR R4.*, a sent data item is described in a `<VARIABLE-ACCESS>` element. Each `<VARIABLE-ACCESS>` element must specify the following properties:

- the `<SHORT-NAME>` that you will use to refer to the item (the short-name does not need to be a valid C identifier)
- the `<ACCESSED-VARIABLE>` element that includes the `<AUTOSAR-VARIABLE-IREF>` element that contains
  - a `<P-PORT-PROTOTYPE-REF>` reference to the Pport
  - a `<TARGET-DATA-PROTOTYPE-REF>` reference to the sent element

```
<RUNNABLE-ENTITY>
    <SHORT-NAME>RunnableEntity</SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
    <DATA-SEND-POINTS>
        <VARIABLE-ACCESS>
            <SHORT-NAME>DataSendPoint1</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                    <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
                    /ASCET_ComponentTypes/SWC/Sender</PORT-PROTOTYPE-REF>
                    <TARGET-DATA-PROTOTYPE-REF DEST=
                    "VARIABLE-DATA-PROTOTYPE">
                    /ASCET_Interfaces/Impl/SRInterface/Speed
                    </TARGET-DATA-PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </DATA-SEND-POINTS>
    <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```


Listing 51: ARXML code – runnable entity with explicit send

For senders, it does not matter how the runnable entity is triggered, so any event can be used to activate the runnable entity.

For the ASCET-generated C code, refer to section 9.3.1, *Sending to a Port: Explicit Communication*, on page 177.

7.4.2 Implicit Communication

Runnable entities can also communicate using implicit data read/write access. Such configuration is guaranteed to be implemented as a simple macro that accesses global storage defined in the RTE rather than through a C function call.

There are two possibilities to model implicit communication in ASCET:

A. Changing the RTE access from explicit to implicit.

B. Modeling the implicit communication without using the RTE access operator.

To change the RTE access to implicit

This instruction is relevant only for the block diagram editor for SWC.

1. In the drawing area, right-click the RTE access operator from the example of section 7.4.1 and select **Access > Implicit** from the context menu as shown in Figure 49.

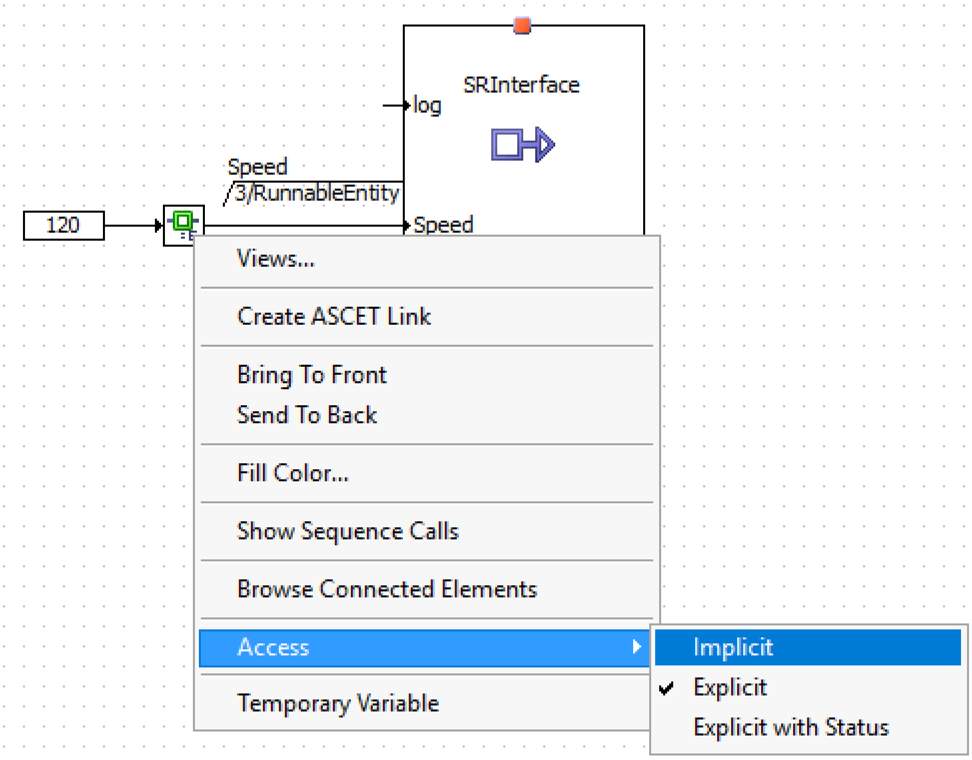


Figure 49: Changing the access type of the RTE Access operator to implicit (block diagram editor for SWC)

To send to a port with implicit communication

1. Insert a literal with value 120.
2. Choose the runnable **RunnableEntity** in the tree pane.
3. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Drag the Pport `Sender` from the "Outline" tab and drop it in the drawing area of the software component editor.
   ii. Connect the literal to the data element `Speed` of the port `Sender`.
   iii. Double-click the sequence call of the data element `Speed`.

   ASCET automatically assigns a sequence number for the sending of the data element `Speed` within the runnable `RunnableEntity`, i.e. the sequence 10.
4. If you are working in the *ESDL editor* for SWC, proceed as follows:
   i. Enter the following code send to a port without status inquiry:

```
Sender.speed = 120;
```

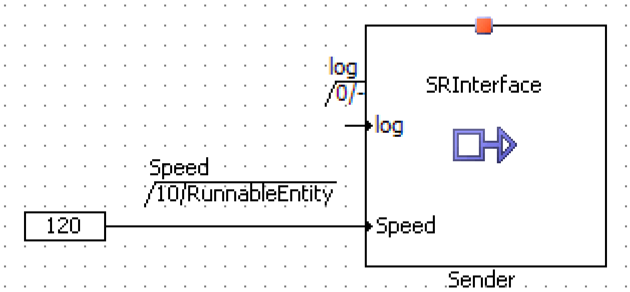


Figure 50: Writing a value 120 to a sender port with implicit communication (block diagram editor for SWC)

The configuration of the implicit communication is almost identical to the explicit communication. Instead of a <DATA-SEND-POINTS> element, the implicit communication is defined using a <DATA-WRITE-ACCESSS> element:

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-WRITE-ACCESSS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataWriteAccess1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
               /ASCET_ComponentTypes/SWC/Sender</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST=
               "VARIABLE-DATA-PROTOTYPE">
               /ASCET_Interfaces/Impl/SRInterface/Speed
               </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-WRITE-ACCESSS>
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

Listing 52: ARXML code – runnable entity with implicit send

For the ASCET-generated C code, refer to section 9.3.1, *Sending to a Port: Explicit Communication*, on page 177.

7.5 Receiving from a Port

Similarly, if your software component requires a sender-receiver interface, then you must define at least one runnable entity that receives data over the interface. Data can be received in the following ways:

- *Implicit* data read access – your runnable is activated by some event, e.g., a timing event, and makes an RTE API call to read data
- *Explicit* Data read access – your runnable entity is activated by an event and makes an RTE API call to read/receive the data. The receiver uses a non-blocking API to poll for the data.

7.5.1 Explicit Data Read Access

To receive from a port with explicit communication

1. Add an Rport `Receiver` to SWC, as described in section *To create a receiver port* on page 96.
2. Insert a limitedInteger variable named `SpeedSwc`, with range `[-32768, 32767]`.
3. Implement the variable `SpeedSwc` as a `sint16`.
4. In the "Outline" tab, select the runnable `RunnableEntity`.
5. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Drag the Rport `Receiver` from the "Outline" tab and drop it in the drawing area of the software component editor.
   ii. Use the **RTE Access** button to create an RTE Access operator and place it in the drawing area.
   iii. Connect the data element `Speed` of the `Receiver` port to the input of the **RTE Access** operator.
   iv. Double-click the empty sequence call of the variable `SpeedSwc`.

   ASCET automaticallys assign a sequence number to `SpeedSwc` within the runnable `RunnableEntity`, e.g., the sequence 10.
6. If you are working in the *ESDL editor* for SWC, proceed as follows:
   i. Enter the following code to receive from a port without status inquiry:

```
Receiver.Speed.explicitRead(SpeedSWC);
```

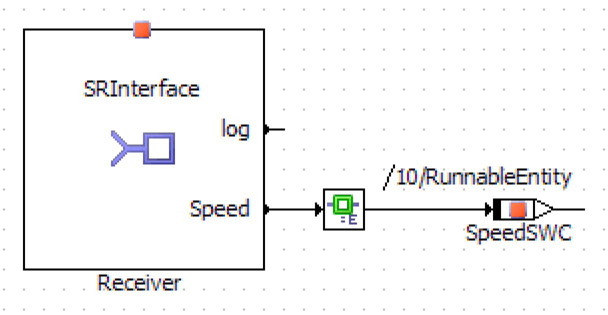


Figure 51: Receiving the value `Speed` from the Rport `Receiver` with explicit communication (block diagram editor for SWC)

Runnables that are required to receive data with explicit "data read access" must define a `<DATA-RECEIVE-POINT-BY-VALUES>` element that specifies the received data items.

In AUTOSAR R4.*, a received data item is described in a `<VARIABLE-ACCESS>` element. Each `<VARIABLE-ACCESS>` element must specify the following properties:

– the `<SHORT-NAME>` that you will use to refer to the item (the short-name does not need to be a valid C identifier)
– the `<ACCESSED-VARIABLE>` element that includes the `<AUTOSAR-VARIABLE-IREF>` element that contains

- a <P-PORT-PROTOTYPE-REF> reference to the Rport
- a <TARGET-DATA-PROTOTYPE-REF> reference to the received element

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-RECEIVE-POINT-BY-VALUES>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataReceivePoint1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
                   /ASCET_ComponentTypes/SWC/Receiver</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST=
                   "VARIABLE-DATA-PROTOTYPE">
                   /ASCET_Interfaces/Impl/SRInterface/Speed
                   </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-RECEIVE-POINT-BY-VALUES>
   ...
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

Listing 53: ARXML code – runnable entity with explicit receive

Using data read access implies that the runnable entity is polling the Rport for the specified data item. It is common, therefore, that a runnable entity that defines a <DATA-RECEIVE-POINT-BY-VALUES> element will be activated by a <TIMING-EVENT> that specifies the required polling period.

For the ASCET-generated C code, refer to section 9.3.4, *Receiving from a Port: Explicit Communication*, on page 180.

7.5.2 Implicit Data Read Access

The following possibilities to model implicit communication are available in ASCET:

A. Changing the RTE access from explicit to implicit.

B. Modeling the implicit communication without using the RTE access operator.

To change the RTE access to implicit

This instruction is relevant only or the block diagram editor for SWC.

1. In the drawing area, right-click the RTE access operator from the example of section 7.5.1 and select **Access > Implicit** from the context menu as shown in Figure 52.

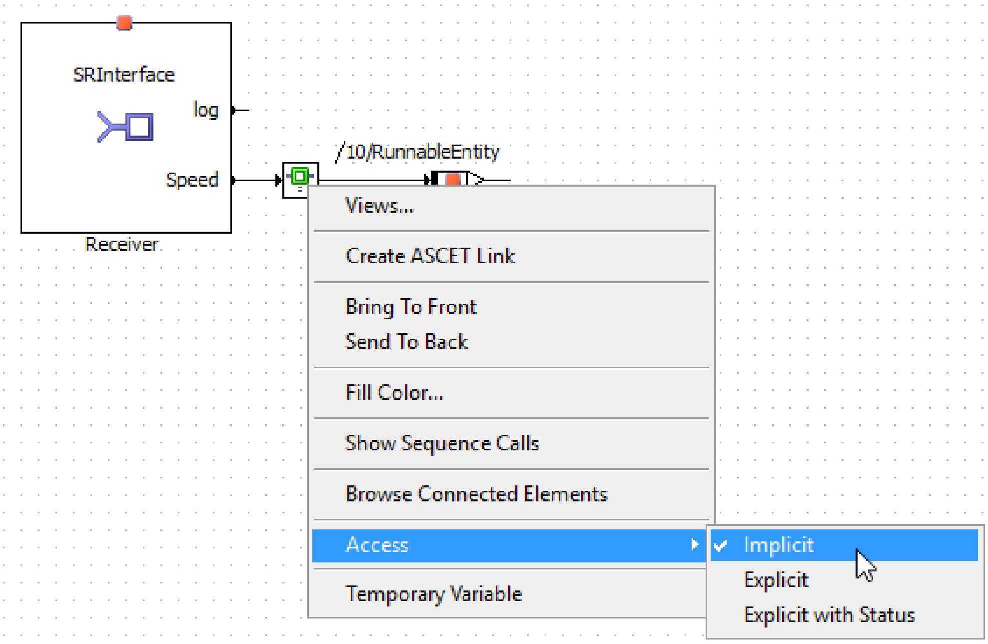


Figure 52: Changing the access type to implicit in the RTE Access operator (block diagram editor for SWC)

To receive from a port with implicit communication

1. Insert a signed discrete variable, name it `SpeedSwc`, and implement it as a `sint16` with implementation range `[-32768, 32767]`.
2. In the "Outline" tab, select the runnable `RunnableEntity`.
3. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Drag the Rport `Receiver` from the "Outline" tab and drop it in the drawing are of the software component editor.
   ii. Connect the data element `Speed` of the `Receiver` port to the variable `SpeedSwc`.
   iii. Double-click the empty sequence call of the variable `SpeedSwc`.

   ASCET automatically assigns a sequence number to `SpeedSwc` within the runnable `RunnableEntity`, e.g., the sequence 10.
4. If you are working in the *ESDL editor* for SWC, proceed as follows:
   i. Enter the following code to receive from a port:

```
Receiver.Speed.explicitRead(SpeedSWC);
```

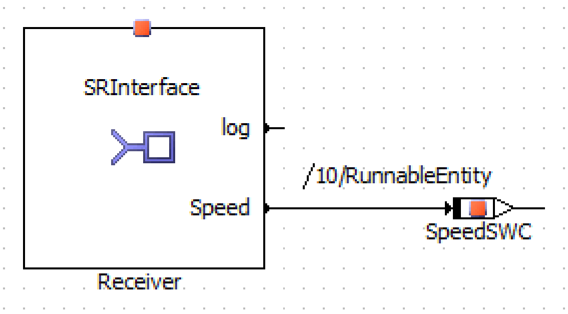


Figure 53: Receiving the value `Speed` from the Rport `Receiver` with implicit communication (block diagram editor for SWC)

Likewise, runnables that are required to receive data with implicit data read access must define a <DATA-READ-ACCESSS> element that specifies the data items they will receive.

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-READ-ACCESSS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataReadAccess1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
               /ASCET_ComponentTypes/SWC/Receiver</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST=
               "VARIABLE-DATA-PROTOTYPE">
               /ASCET_Interfaces/Impl/SRInterface/Speed
               </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-READ-ACCESSS>
   ...
   <SYMBOL>RteRunnable</SYMBOL>
</RUNNABLE-ENTITY>
```

Listing 54: ARXML code – runnable entity with implicit receive

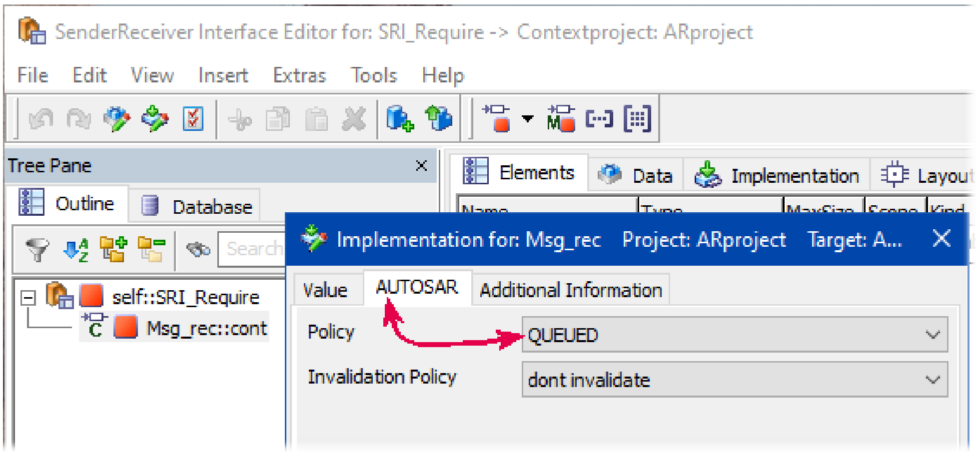
A single received data item is described by a <VARIABLE-ACCESS> element. A <VARIABLE-ACCESS> element must be named using the <SHORT-NAME> element. The name is used within other elements to reference the data read access. The short-name does not need to be a valid C identifier.

For the ASCET-generated code, refer to section 9.3.6, *Receiving from a Port: Implicit Communication*, on page 182.

7.6 Queued Communication

Elements in SenderReceiver interfaces can use queued communication. If set to queued, the semantics is that the corresponding element needs to be added to a queue (i.e., a FIFO data structure), from which it is later consumed by the actual receiver software-component.

In ASCET, queued communication is activated in the AUTOSAR tab of the implementation editor of a SenderReceiver interface element:



The following rules apply:

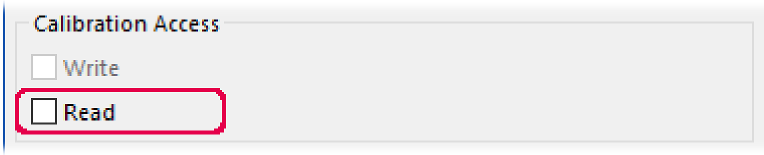
- Queued elements must use explicit access.

  If an implicit RTE Access operator is connected to a queued element, an error `MMd11132` is issued during code generation.

- ASCET allows queued communication only for the elements of SenderReceiver interfaces.

  If `SW-IMPL-POLICY` is set to `QUEUED` for an element of another interface, an error `MMd1282` is issued during code generation.

- Calibration access to a queued element must be deactivated, i.e. `SW-CALIBRATION-ACCESS` must be set to `NOT-ACCESSIBLE`. In ASCET, this is done by deactivating both options in the Calibration Access area in the element's properties editor.

  If `SW-CALIBRATION-ACCESS` is set to `ReadOnly` (realized in by activating **Read** in the **Calibration Access** area of the properties editor), a warning `WMd1283` is issued during code generation.

Access to queued elements uses special RTE macros, `Rte_Send` and `Rte_Receive`. For more information on these macros, see the ASCET online help.

One SenderReceiver interface can have some elements with queued communication and other elements with non-queued communication.

To activate queued communication for an element

1. Create a SenderReceiver interface (named, e.g., `SRI_queued`) with two elements.

Name	`cont_queued`	`log_queued`
Calibration access	none	none
Model data type	`cont`	`log`
Implementation data type	`sint16`	`bool`

2. Open the implementation editor for each element and set the **Policy** to `Queued`.

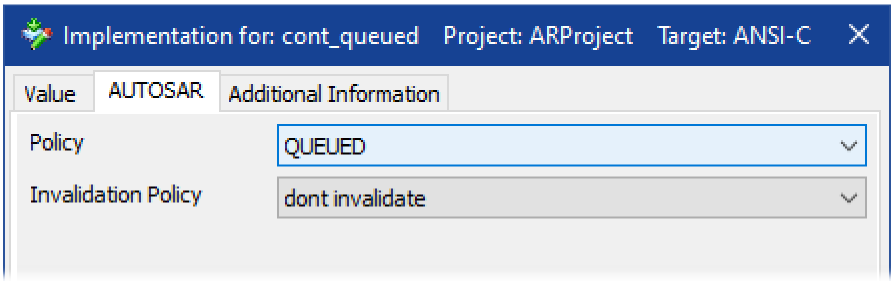
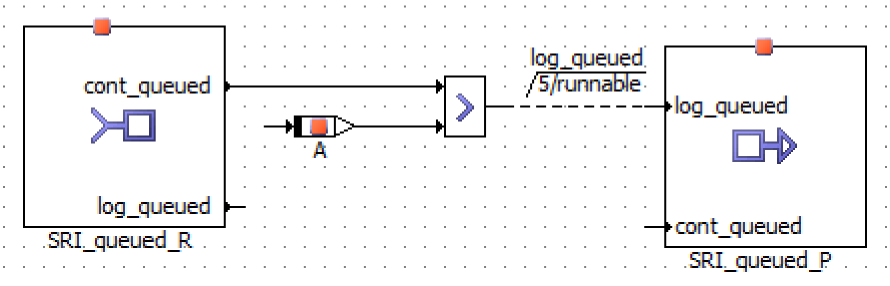


Figure 54: Implementation editor, **AUTOSAR** tab, with **Policy** set to `QUEUED`

3. Add a Pport `SRI_queued_P` and an Rport `SRI_queued_R` to an SWC.

   If you need help, see sections 6.1.1.1 *Sender Port* and 6.1.2.1 *Receiver Port*.

4. Add a cont variable with implementation data type `sint16`.
5. Specify the functionality, e.g., as shown in the following figure.



6. Add the SWC to a project that is set up for AUTOSAR.

   If you need help, see section 3.1.2 *Code Generation Settings for AUTOSAR*.

The `<PROVIDED-COM-SPECS>` element in the Pport definition in `<SWC_name>.arxml` contains a `<QUEUED-SENDER-COM-SPEC>` element for each queued element.

```
<P-PORT-PROTOTYPE>
  <SHORT-NAME>SRI_queued_P</SHORT-NAME>
    <PROVIDED-COM-SPECS>
      ...
      <QUEUED-SENDER-COM-SPEC>
        <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
          /ASCET_Interfaces/Impl/SRI_queued/cont_queued
        </DATA-ELEMENT-REF>
      </QUEUED-SENDER-COM-SPEC>
      ...
  </PROVIDED-COM-SPECS>
  <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
    /ASCET_Interfaces/Impl/SRI_queued
  </PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```

Listing 55: ARXML code – Pport `SRI_queued_P` definition with queued elements

The <Required-COM-SPECS> element in the RPort definition in *<SWC_name>*.arxml contains a <QUEUED- RECEIVER-COM-SPEC> element for each queued element.

```
<R-PORT-PROTOTYPE>
  <SHORT-NAME>SRI_queued_R</SHORT-NAME>
    <REQUIRED-COM-SPECS>
      ...
      <QUEUED-RECEIVER-COM-SPEC>
        <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
          /ASCET_Interfaces/Impl/SRI_queued/cont_queued
        </DATA-ELEMENT-REF>
        <QUEUE-LENGTH>1</QUEUE-LENGTH>
      </QUEUED-RECEIVER-COM-SPEC>
      ...
    </REQUIRED-COM-SPECS>
  <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
    /ASCET_Interfaces/Impl/SRI_queued
  </REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Listing 56: ARXML code – Rport SRI_queued_R definition with queued elements

7.7 Responding to a Server Request on a Port

In software components that provide a client-server interface, ASCET defines one runnable entity for each operation in the interface. These runnable entities are the servers for the client-server Pports on the software component.

The runnable entity to be regarded by the RTE as a server must be tied to an <OPERATION-INVOKED-EVENT>. This RTE event allows the RTE to call the runnable entity at runtime in response to client requests. The <OPERATION-INVOKED-EVENT> must specify what operation request on the server interface will result in the runnable entity being activated.

The following example shows how ASCET configures the runnable Server_MaximumValue to be executed when the operation called MaximumValue is called on the Pport Server of interface type CSInterface. See also:

A. section 5.3, *Client-Server*, on page 70 for the creation of the client interface CSInterface
B. section 6.1.1, *Provided Ports*, on page 86 for the creation of the Server Pport
C. section 7.1.2, *Operation-Invoked Events*, on page 108 for a detailed description of Operation-Invoked events

```
<SWC-INTERNAL-BEHAVIOR>
  <SHORT-NAME>bSWC</SHORT-NAME>
  <DATA-TYPE-MAPPING-REFS>
    <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
    /ASCET_Mappings/DataMappings/Impl/SWC</DATA-TYPE-MAPPING-REF>
  </DATA-TYPE-MAPPING-REFS>
  <EVENTS>
    ...
    <OPERATION-INVOKED-EVENT>
      <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
      /ASCET_ComponentTypes/SWC/bSWC/Server_MaximumValue
      </START-ON-EVENT-REF>
      <OPERATION-IREF>
        <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
        /ASCET_ComponentTypes/SWC/Server</CONTEXT-P-PORT-REF>
        <TARGET-PROVIDED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
        /ASCET_Interfaces/Impl/CSInterface/MaximumValue
        </TARGET-PROVIDED-OPERATION-REF>
      </OPERATION-IREF>
    </OPERATION-INVOKED-EVENT>
    ...
  </EVENTS>
  ...
  <RUNNABLES>
    ...
    <RUNNABLE-ENTITY>
      <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
      <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
      </SW-ADDR-METHOD-REF>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <SYMBOL>SWC_Impl_Server_MaximumValue</SYMBOL>
    </RUNNABLE-ENTITY>
    ...
  </RUNNABLES>
</SWC-INTERNAL-BEHAVIOR>
```

Listing 57: ARXML code – internal behavior responding to a server request

An `<OPERATION-INVOKED-EVENT>` must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the event. The short-name does not need to be a valid C identifier.

7.7.1 Concurrent Invocation of Servers

When a runnable acting as a server is written to be invoked concurrently, then the RTE can optimize invocation by clients on the same ECU to a direct function call. This means that no queuing is required (or possible) and therefore multiple invocations of the server can occur concurrently.

The RTE generator needs to know which runnable entities can be called in this way.

To enable concurrent invocation of a server

1. In the "Outline" tab of the software component editor, double-click the server runnable `Server_MaximumValue`.

   The server runnable `Server_MaximumValue` was automatically inserted by ASCET when the Pport `Server` was created in section 6.1.1, *Provided Ports*, on page 86.

The "Runnable Signature Editor for: Server_MaximumValue" window opens.

2. Select the "Settings" tab.
3. Activate the **Can be Invoked Concurrently** option.

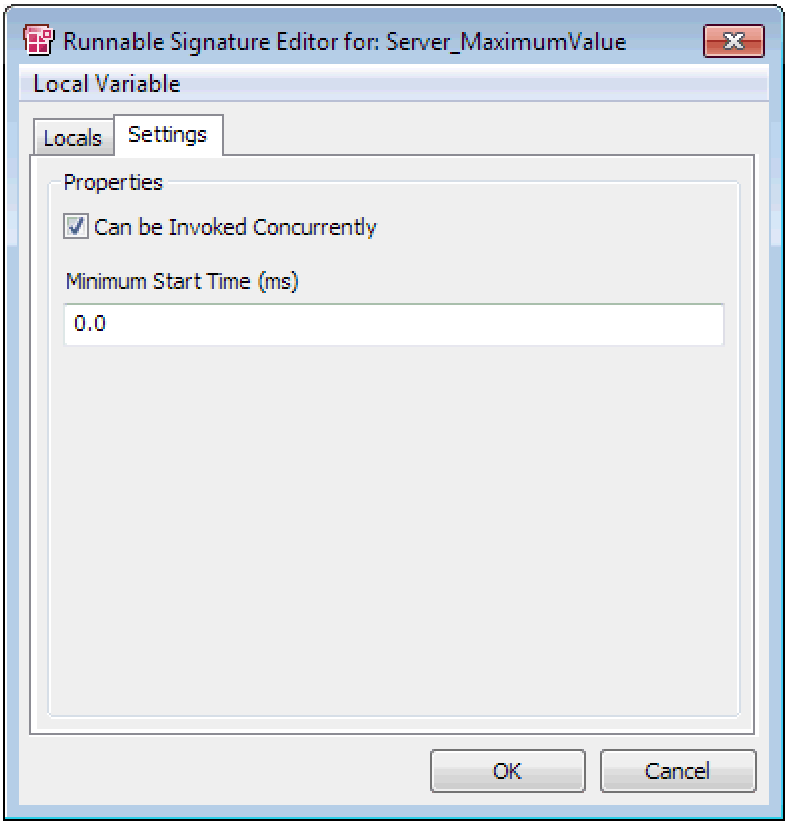


Figure 55: Setting **Can be Invoked Concurrently** for the runnable `Server_MaximumValue`

4. Close the runnable signature editor with **OK**.

Concurrent invocation is defined within the server's runnable entity definition as follows:

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
      /ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
   <SYMBOL>Swc_Impl_Server_MaximumValue</SYMBOL>
</RUNNABLE-ENTITY>
```

Listing 58: ARXML code – server runnable with concurrent invocation

> **NOTE**
>
> The runnable must be written to be invoked concurrently. If this is not the case, then data consistency is not guaranteed when there is more than one client simultaneously requesting the server.

7.8 Making a Client Request on a Port

Similarly, if your software component requires a client-server interface, then you must define at least one runnable entity that acts as the client.

In ASCET, clients can access servers synchronously, which means that the client is blocked while the server processes the request. When the server has processed the request, the result is passed back to the client and the client continues the execution. You have to ensure that the client is triggered by an RTE event.

To make a client request on a port

1. Add an Rport `Client` to `Swc`, as described in section *To create a client port* on page 99.
2. Insert a signed discrete variable, name it `A`, and implement it as a `sint16` with implementation range `[-32768, 32767]`.
3. Create the signed discrete variables `B` and `C` with the same implementation as `A`.
4. Choose the runnable **RunnableEntity** in the tree pane.
5. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Drag the Rport `Client` from the "Outline" tab and drop it in the drawing area of the software component editor.
   ii. Deactivate the method `Notification`, as described on page 100.
   iii. Connect `A` and `B` to the arguments `InputA` and `InputB` of the Rport `Client`.
   iv. Connect `C` to the argument `OutputMaximum` of `Client`.
   v. Use the **RTE Invoke** button to create an RTE Invoke operator and place it in the drawing area.
   vi. Connect the return value of the operation `MaximumValue` to the RTE Invoke operator.
   vii. Double-click the empty sequence call **InvokeOp**.

   ASCET will automatically assign a sequence number to **InvokeOp** within the runnable `RunnableEntity`, i.e. the sequence 5.
6. If you are working in the *ESDL editor* for SWC, proceed as follows:
   i. Enter the following code for a client request without status inquiry:

```
Client.MaximumValue(A, B, C);
```

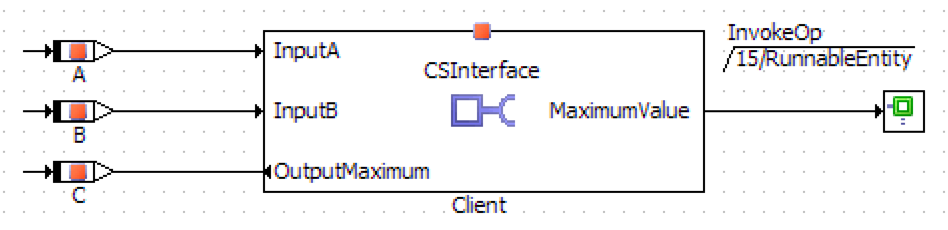


Figure 56: Request on Rport `Client` to compute `MaximumValue(A,B)` and store it in `C` (block diagram editor for SWC)

Runnable entities that need to call a server synchronously must define a synchronous server call point. The `<SYNCHRONOUS-SERVER-CALL-POINT>` element defines which operations the client can call, and specifies a global `<TIMEOUT>` value for all called operations. The `<TIMEOUT>` specifies the maximum time that the client will wait for any of the servers providing an operation.

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   ...
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
      /ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <SERVER-CALL-POINTS>
      <SYNCHRONOUS-SERVER-CALL-POINT>
         <SHORT-NAME>ServerCallPoint1</SHORT-NAME>
         <OPERATION-IREF>
            <CONTEXT-R-PORT-REF DEST="R-PORT-PROTOTYPE">
               /ASCET_ComponentTypes/Swc/Client
            </CONTEXT-R-PORT-REF>
            <TARGET-REQUIRED-OPERATION-REF
                                        DEST="CLIENT-SERVER-OPERATION">
               /ASCET_Interfaces/Impl/CSInterface/MaximumValue
            </TARGET-REQUIRED-OPERATION-REF>
         </OPERATION-IREF>
         <TIMEOUT>0</TIMEOUT>
      </SYNCHRONOUS-SERVER-CALL-POINT>
   </SERVER-CALL-POINTS>
   <SYMBOL>Swc_Impl_RunnableEntity</SYMBOL>
   ...
</RUNNABLE-ENTITY>
```

Listing 59: ARXML code – runnable entity with client request (AUTOSAR R4.2.2)

A `<SYNCHRONOUS-SERVER-CALL-POINT>` must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the call point. The short-name does not need to be a valid C identifier, but it must pass the syntactic checks imposed by the AUTOSAR schema.

> **NOTE**
>
> The global `<TIMEOUT>` value for all the called operations is always set to 0 in ASCET.

For the ASCET-generated C code, refer to section 9.4.2, *Making a Client Request on a Port*, on page 184.

The same runnable entity can be used as a server on one interface and client on another interface. For example, you may create a runnable entity that handles a server request for sorting on a Pport and uses an auxiliary operation on an Rport.

7.9 Interrunnable Variables

In non-AUTOSAR projects, ASCET messages can be used for inter-process communication. These messages are not available in AUTOSAR software components. Instead, interrunnable variables are used for communication between different runnable entities.

Communication via interrunnable variables is equivalent in semantics to implicit/explicit sender-receiver communication (see also section 5.1, *Sender-Receiver*, on page 63), but within the scope of the software component instance.

7.9.1 Scalar Interrunnable Variables

To specify scalar interrunnable variables

1. In the software component editor, use the **Interrunnable Variable** button to add an interrunnable variable.

   The "Properties for Scalar Element: interrunnable" dialog window opens.
2. Name the interrunnable variable `IRV_explicit`.
3. Set the "Internal Access" to **Explicit**.
4. Select a "Basic Type", e.g., `Signed Discrete`.
5. Close the properties editor with **OK**.
6. Create a second interrunnable variable `IRV_implicit` with **Implicit** internal access.
7. Implement both interrunnable variables as `sint8` (see Figure 12).

In AUTOSAR R4.*, a scalar interrunnable variable is described in a `<VARIABLE-DATA-PROTOTYPE>` element. Explicit and implicit interrunnable variables are stored in different elements of the `<SWC-INTERNAL-BEHAVIOR>`, i.e. `<EXPLICIT-INTER-RUNNABLE-VARIABLES>` (see Listing 60) and `<IMPLICIT-INTER-RUNNABLE-VARIABLES>` (see Listing 61).

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_explicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt8</TYPE-TREF>
    <INIT-VALUE>
      <NUMERICAL-VALUE-SPECIFICATION>
        <VALUE>0</VALUE>
      </NUMERICAL-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 60: ARXML code – explicit scalar interrunnable variable

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_implicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt8</TYPE-TREF>
    <INIT-VALUE>
      <NUMERICAL-VALUE-SPECIFICATION>
        <VALUE>0</VALUE>
      </NUMERICAL-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 61: ARXML code – implicit scalar interrunnable variable

Each interrunnable variable must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the interrunnable variable.

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see Listing 36. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values* on page 89 for an instruction how to select the specification means.

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>msg_1</SHORT-NAME>
    ...
    <INIT-VALUE>
      <APPLICATION-VALUE-SPECIFICATION>
        <CATEGORY>VALUE</CATEGORY>
        <SW-VALUE-CONT>
          <UNIT-REF DEST="UNIT">/ASCET_Units/NoUnit</UNIT-REF>
          <SW-VALUES-PHYS>
            <V>3.1415</V>
          </SW-VALUES-PHYS>
        </SW-VALUE-CONT>
      </APPLICATION-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 62: ARXML code – explicit scalar interrunnable variable with `<APPLICATION-VALUE-SPECIFICATION>`

7.9.2 Complex Interrunnable Variables

In AUTOSAR R4.*, interrunnable variables can also be of a complex type:

- *Interrunnable Variables of Array Type* on page 133
- *Interrunnable Variables of Matrix Type* on page 137
- *Interrunnable Variables of Record Type* on page 142

Read access to a complex interrunnable variable, or to an element inside a complex interrunnable variable, can be explicit or implicit.

Write access to an element in a complex interrunnable variable (e.g., an array element, or an element inside a record) must be implicit (Figure 61, sequence call `/20/ModeRunnable`). Explicit write access to elements in complex interrunnable variables causes an error:

- `MMdl1190 – %1 access to an array element` *<array>*[11]`[`*<x>*`,`*<y>*[12]`] (not mapped) cannot be explicit`
- `MMdl119 – Write access to a record element` *<record>*`.`*<record_element>* `(not mapped) cannot be explicit`

Write access to a complex interrunnable variable as a whole, i.e. via its Set port, can be explicit (Figure 63, sequence call `/30/ModeRunnable`) or implicit (Figure 61, sequence call `/10/RunnableEntity`).

[11] includes arrays and matrices
[12] only for matrices

7.9.2.1 Interrunnable Variables of Array Type

To create an interrunnable of array type

1. In the software component editor, use the **Array** button to create an array.

   The "Properties for Array Element: *" dialog window opens.

2. In that dialog, select the kind `Interrunnable Variable`.

   > **NOTE**
   >
   > This is the only way to create an interrunnable variable of array type.
   >
   > You cannot convert an array interrunnable variable into a scalar interrunnable variable, or vice versa.

3. Enter the name `IRV_array`, the desired dimension, and other properties, then click **OK** to close the window.

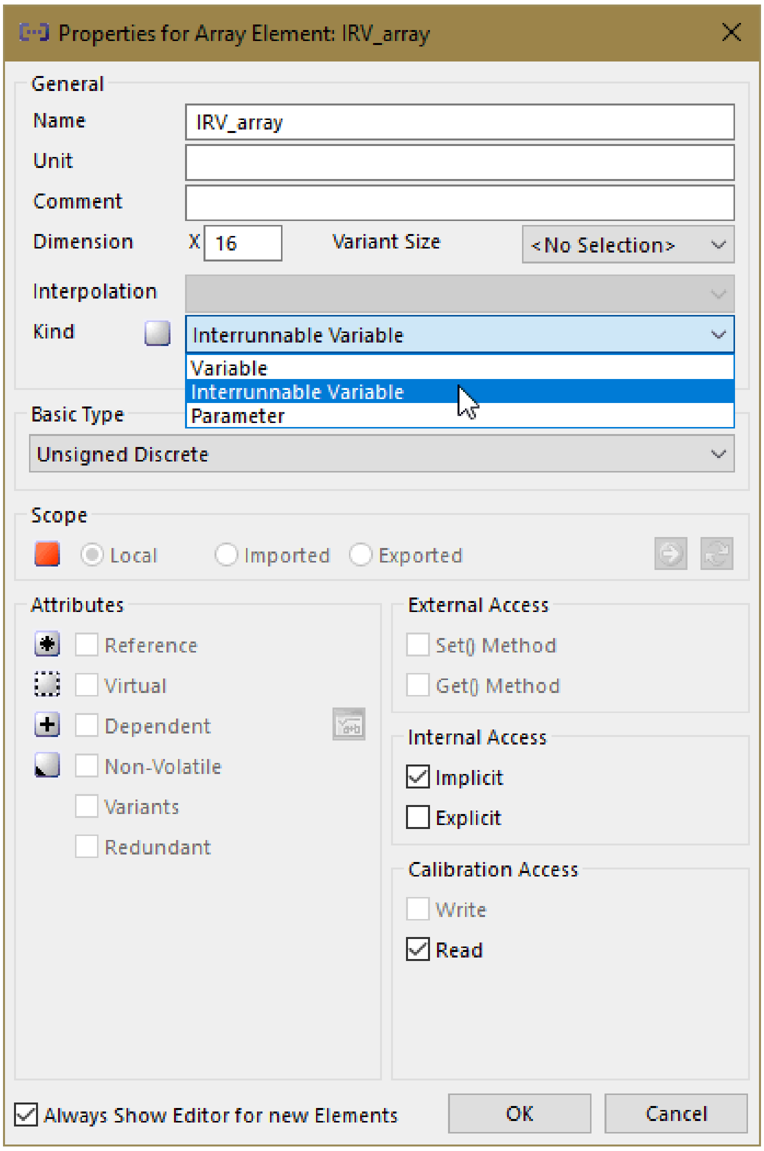


Figure 57: Interrunnable variable of array type

In AUTOSAR R4.*, an interrunnable variable of array type is described in a `<VARIABLE-DATA-PROTOTYPE>` element below `<IMPLICIT-INTER-RUNNABLE-VARIABLES>` or `<EXPLICIT-INTER-RUNNABLE-VARIABLES>` in the `<SWC-INTERNAL-BEHAVIOR>` (see Listing 63 and Listing 64).

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
  ...
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_array</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt32_ident_n32768_p32767_4</TYPE-TREF>
    <INIT-VALUE>
      <ARRAY-VALUE-SPECIFICATION>
        <ELEMENTS>
          <NUMERICAL-VALUE-SPECIFICATION>
            <VALUE>0</VALUE>
          </NUMERICAL-VALUE-SPECIFICATION>
          <NUMERICAL-VALUE-SPECIFICATION>
            <VALUE>0</VALUE>
          </NUMERICAL-VALUE-SPECIFICATION>
          ...
        </ELEMENTS>
      </ARRAY-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
  ...
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 63: ARXML code – implicit interrunnable variable of array type

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  ...
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_array_e</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt32_ident_n32768_p32767_4</TYPE-TREF>
    <INIT-VALUE>
      <ARRAY-VALUE-SPECIFICATION>
        <ELEMENTS>
          <NUMERICAL-VALUE-SPECIFICATION>
            <VALUE>0</VALUE>
          </NUMERICAL-VALUE-SPECIFICATION>
          <NUMERICAL-VALUE-SPECIFICATION>
            <VALUE>0</VALUE>
          </NUMERICAL-VALUE-SPECIFICATION>
          ...
        </ELEMENTS>
      </ARRAY-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
  ...
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 64: ARXML code – explicit interrunnable variable of array type

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see, e.g., Listing 63. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values* on page 89 for an instruction how to select the specification means.

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
   <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>array</SHORT-NAME>
      ...
      <INIT-VALUE>
         <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
               <APPLICATION-VALUE-SPECIFICATION>
                  <CATEGORY>VALUE</CATEGORY>
                  <SW-VALUE-CONT>
                     <UNIT-REF DEST="UNIT">/ASCET_Units/NoUnit
                         </UNIT-REF>
                     <SW-VALUES-PHYS>
                        <V>5.0</V>
                     </SW-VALUES-PHYS>
                  </SW-VALUE-CONT>
               </APPLICATION-VALUE-SPECIFICATION>
               ...
            </ELEMENTS>
         </ARRAY-VALUE-SPECIFICATION>
      </INIT-VALUE>
   </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 65: ARXML code – interrunnable variable of array type with `<APPLICATION-VALUE-SPECIFICATION>`

7.9.2.2 Interrunnable Variables of Matrix Type

> **NOTE**
>
> If you use matrices as interrunnable variables, make sure that the “Number of dimensions for fixed matrixes” target option is set to `Two-dimensional`.
>
> Otherwise, a warning `WMd1653` is issued during AUTOSAR code generation. By default, this warning is promoted to an error.

To create an interrunnable of matrix type

1. In the software component editor, use the **Matrix** button to create a matrix.

   The "Properties for Matrix Element: *" dialog window opens.

2. In that dialog, select the kind `Interrunnable Variable`.

   > **NOTE**
   >
   > This is the only way to create an interrunnable variable of matrix type.
   >
   > You cannot convert a matrix interrunnable variable into a scalar interrunnable variable, or vice versa.

3. Enter the name `IRV_matrix`, the desired dimension, and other properties, then click **OK** to close the window.

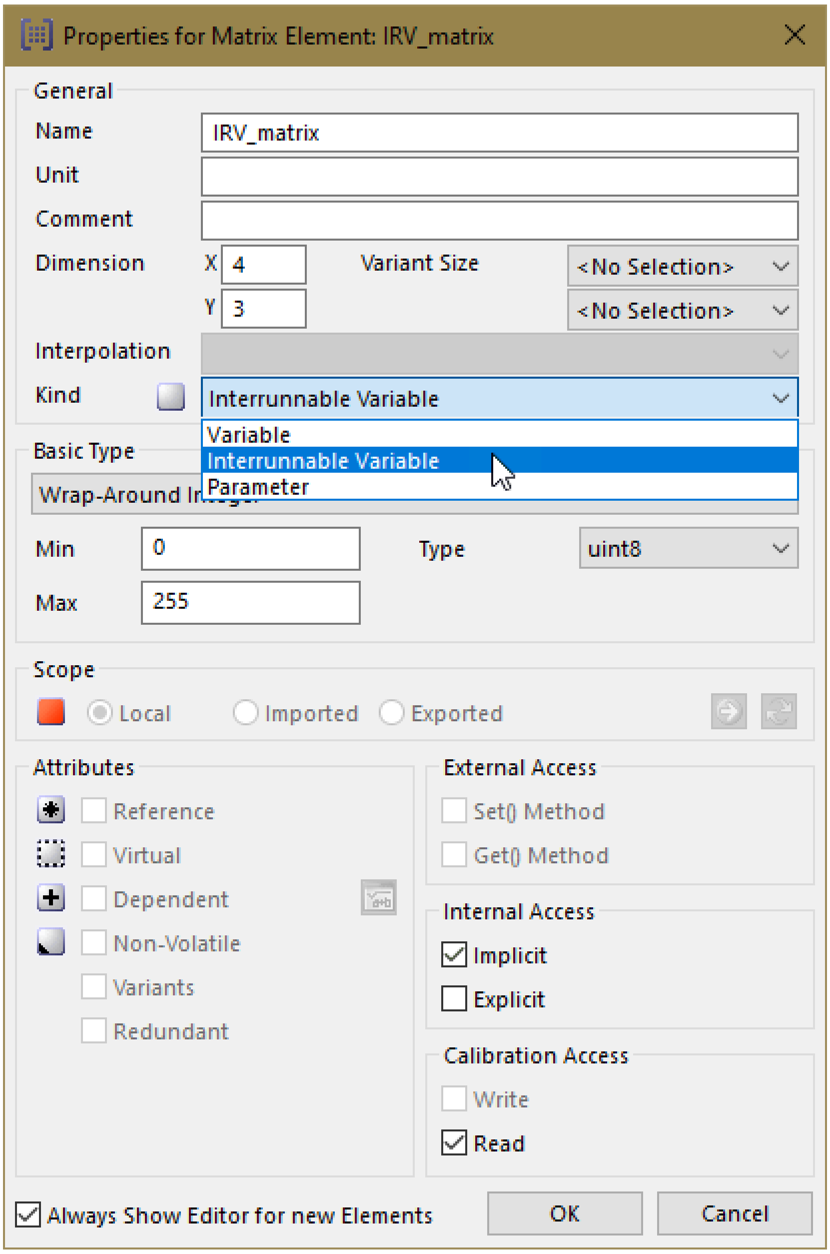


Figure 58: Interrunnable variable of matrix type

An interrunnable variable of matrix type is described in a `<VARIABLE-DATA-PROTOTYPE>` element below `<IMPLICIT-INTER-RUNNABLE-VARIABLES>` or `<EXPLICIT-INTER-RUNNABLE-VARIABLES>` in the `<SWC-INTERNAL-BEHAVIOR>` (see Listing 66 and Listing 67).

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
   ...
   <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>IRV_matrix_i</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
         <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
               <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
               /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
               <SW-CALIBRATION-ACCESS>READ-ONLY
               </SW-CALIBRATION-ACCESS>
               <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
         </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/SInt32_2_6</TYPE-TREF>
      <INIT-VALUE>
         <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
               <ARRAY-VALUE-SPECIFICATION>
                  <ELEMENTS>
                     <NUMERICAL-VALUE-SPECIFICATION>
                        <VALUE>0</VALUE>
                     </NUMERICAL-VALUE-SPECIFICATION>
                     ...
                  </ELEMENTS>
               </ARRAY-VALUE-SPECIFICATION>
               <ARRAY-VALUE-SPECIFICATION>
                  <ELEMENTS>
                     <NUMERICAL-VALUE-SPECIFICATION>
                        <VALUE>0</VALUE>
                     </NUMERICAL-VALUE-SPECIFICATION>
                     ...
                  </ELEMENTS>
               </ARRAY-VALUE-SPECIFICATION>
               ...
            </ELEMENTS>
         </ARRAY-VALUE-SPECIFICATION>
      </INIT-VALUE>
   </VARIABLE-DATA-PROTOTYPE>
   ...
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 66: ARXML code – implicit interrunnable variable of matrix type

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  ...
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_matrix_e</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_INIT</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY
          </SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/UInt8_4_3</TYPE-TREF>
    <INIT-VALUE>
      <ARRAY-VALUE-SPECIFICATION>
        <ELEMENTS>
          <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
              <NUMERICAL-VALUE-SPECIFICATION>
                <VALUE>1</VALUE>
              </NUMERICAL-VALUE-SPECIFICATION>
              ...
            </ELEMENTS>
          </ARRAY-VALUE-SPECIFICATION>
          <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
              <NUMERICAL-VALUE-SPECIFICATION>
                <VALUE>2</VALUE>
              </NUMERICAL-VALUE-SPECIFICATION>
              ...
            </ELEMENTS>
          </ARRAY-VALUE-SPECIFICATION>
          ...
        </ELEMENTS>
      </ARRAY-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 67: ARXML code – explicit interrunnable variable of matrix type

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see, e.g., see Listing 66. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-`

`SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values* on page 89 for an instruction how to select the specification means.

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>matrix</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY
          </SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-ARRAY-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt32_2_6</TYPE-TREF>
    <INIT-VALUE>
      <ARRAY-VALUE-SPECIFICATION>
        <ELEMENTS>
          <ARRAY-VALUE-SPECIFICATION>
            <ELEMENTS>
              <APPLICATION-VALUE-SPECIFICATION>
                <CATEGORY>VALUE</CATEGORY>
                <SW-VALUE-CONT>
                  <UNIT-REF DEST="UNIT">/ASCET_Units/NoUnit
                  </UNIT-REF>
                  <SW-VALUES-PHYS>
                    <V>0.0</V>
                  </SW-VALUES-PHYS>
                </SW-VALUE-CONT>
              </APPLICATION-VALUE-SPECIFICATION>
            ...
          </ARRAY-VALUE-SPECIFICATION>
          ...
        </ELEMENTS>
      </ARRAY-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 68: ARXML code – interrunnable variable of matrix type with `<APPLICATION-VALUE-SPECIFICATION>`

7.9.2.3 Interrunnable Variables of Record Type

To create an interrunnable variable of record type

1. In the software component editor, select **Insert > Component**.

   The "Select Item" window opens. It shows the content of the current database/workspace.

2. From the "Database" or "Workspace" list, select the record you want to use as interrunnable variable.

   > **NOTE**
   >
   > Records are the only ASCET components that can be used as complex interrunnable variable.

3. Click **OK** to add the record.

   The properties editor for the record opens.

4. Name the record instance `Record_IRV`.
5. Open the "Kind" combo box and select `Interrunnable Variable`.

   > **NOTE**
   >
   > This is the only way to create an interrunnable variable of record type.
   >
   > You cannot convert this interrunnable variable into a scalar interrunnable variable.

6. Adjust the properties according to your needs and click **OK**.

   The complex interrunnable variable is listed in the "Outline" tab. Its elements are available for message mapping; they appear in the "Internal Access" tab.

   See section 9.5.2, *Accessing ASCET Messages*, on page 190 for more details on mapping.

A complex interrunnable variable is described in a `<VARIABLE-DATA-PROTOTYPE>` element below `<IMPLICIT-INTER-RUNNABLE-VARIABLES>` or `<EXPLICIT-INTER-RUNNABLE-VARIABLES>` in the `<SWC-INTERNAL-BEHAVIOR>` (see Listing 69 and Listing 70).

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
   ...
   <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>Record_IRV</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
         <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
               <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
               /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
               <SW-CALIBRATION-ACCESS>NOT-ACCESSIBLE
               </SW-CALIBRATION-ACCESS>
               <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
         </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/Record_Impl</TYPE-TREF>
      <INIT-VALUE>
         <RECORD-VALUE-SPECIFICATION>
            <!--
               APPLICATION-RECORD-DATA-TYPE: Record_Impl
            -->
            <FIELDS>
               <NUMERICAL-VALUE-SPECIFICATION>
                  <VALUE>0</VALUE>
               </NUMERICAL-VALUE-SPECIFICATION>
               ...
            </FIELDS>
         </RECORD-VALUE-SPECIFICATION>
      </INIT-VALUE>
   </VARIABLE-DATA-PROTOTYPE>
   ...
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 69: ARXML code – implicit interrunnable variable of record type

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  ...
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>Record_IRV_expl</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
          /ASCET_AddrMethods/RAM_CLEARED</SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>NOT-ACCESSIBLE
          </SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/Record_Impl</TYPE-TREF>
    <INIT-VALUE>
      <RECORD-VALUE-SPECIFICATION>
        <!--
          APPLICATION-RECORD-DATA-TYPE: Record_Impl
        -->
        <FIELDS>
          <NUMERICAL-VALUE-SPECIFICATION>
            <VALUE>0</VALUE>
          </NUMERICAL-VALUE-SPECIFICATION>
          ...
        </FIELDS>
      </RECORD-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
  ...
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 70: ARXML code – explicit interrunnable variable of record type

By default, initial values in ASCET are given as `<NUMERICAL-VALUE-SPECIFICATION>`; see, e.g., Listing 69. However, AUTOSAR R4.* offers another specification means for physical initial values, `<APPLICATION-VALUE-SPECIFICATION>`, associated with elements typed using application data types (see section 4.1, *Application Data Types*, on page 31).

See section *To select the specification means for initial values on page* 89 for an instruction how to select the specification means.

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
   <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>Record</SHORT-NAME>
      ...
      <TYPE-TREF DEST="APPLICATION-RECORD-DATA-TYPE">
          /ASCET_Types/ApplicationDataTypes/Record_Impl</TYPE-TREF>
      <INIT-VALUE>
         <RECORD-VALUE-SPECIFICATION>
            <FIELDS>
               <APPLICATION-VALUE-SPECIFICATION>
                  <CATEGORY>VALUE</CATEGORY>
                  <SW-VALUE-CONT>
                     <UNIT-REF DEST="UNIT">/ASCET_Units/NoUnit
                         </UNIT-REF>
                     <SW-VALUES-PHYS>
                        <V>19</V>
                     </SW-VALUES-PHYS>
                  </SW-VALUE-CONT>
               </APPLICATION-VALUE-SPECIFICATION>
               ...
            </FIELDS>
         </RECORD-VALUE-SPECIFICATION>
      </INIT-VALUE>
   </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

Listing 71: ARXML code – interrunnable variable of record type with `<APPLICATION-VALUE-SPECIFICATION>`

7.9.3 Read and Write Access

Each runnable entity must explicitly specify whether it reads or writes an interrunnable variable at runtime.

Read access to a complex interrunnable variable, or to an element inside a complex interrunnable variable, can be explicit or implicit.

Write access to an element in a complex interrunnable variable (e.g., an array element, or an element inside a record) must be implicit (Figure 61, sequence call `/20/ModeRunnable`). Explicit write access to elements in complex interrunnable variables causes an error:

- `MMdl1190 – %1 access to an array element` *<array>*[13]`[`*<x>*`,`*<y>*[14]`] (not mapped) cannot be explicit`
- `MMdl119 – Write access to a record element` *<record>*`.`*<record_element>* `(not mapped) cannot be explicit`

Write access to a complex interrunnable variable as a whole, i.e. via its Set port, can be explicit (Figure 63, sequence call `/30/ModeRunnable`) or implicit (Figure 61, sequence call `/10/RunnableEntity`).

[13] includes arrays and matrices
[14] only for matrices

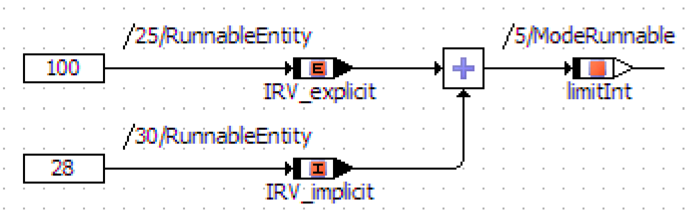


Figure 59: Scalar interrunnable variables used by two runnable entities (block diagram editor for SWC)

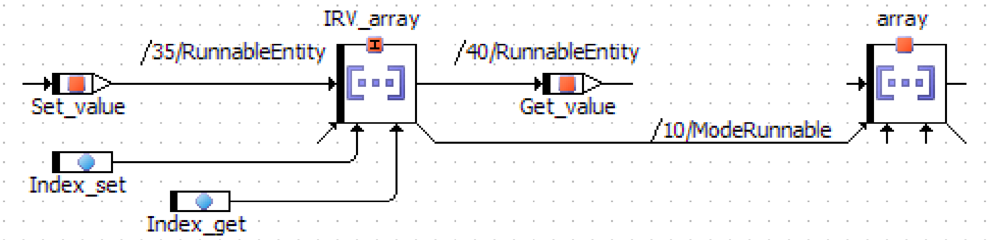


Figure 60: Complex interrunnable variable (implicit, array) used by two runnable entities (block diagram editor for SWC)

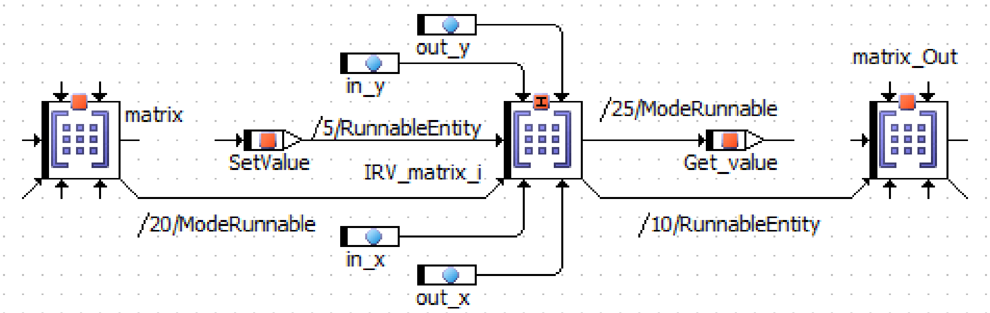


Figure 61: Complex interrunnable variable (implicit, matrix) used by two runnable entities (block diagram editor for SWC)

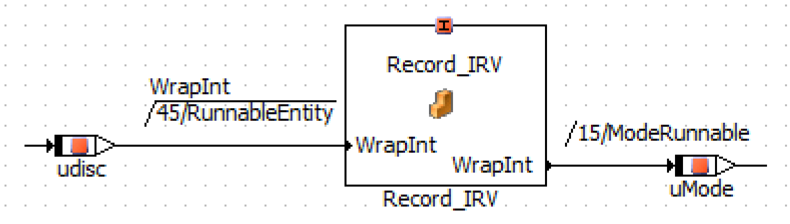


Figure 62: Complex interrunnable variable (implicit, record) used by two runnable entities (block diagram editor for SWC)

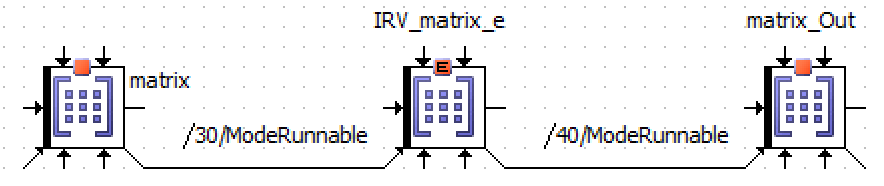


Figure 63: Explicit read and write access to a complex interrunnable variable (explicit, matrix)

The corresponding ESDL code for Figure 59 – Figure 63 is given in Table 4.

	ESDL code	Runnable
Figure 59	IRV_explicit = 100; IRV_implicit = 28;	RunnableEntity
	limitInt = IRV_explicit + IRV_implicit;	ModeRunnable
Figure 60	IRV_array[Index_set] = Set_value; GetValue = IRV_array[Index_get];	RunnableEntity
	array = IRV_array;	ModeRunnable
Figure 61	IRV_matrix_i[in_x][in_y] = SetValue; matrix_Out = IRV_matrix_i;	RunnableEntity
	IRV_matrix_i = matrix; Get_value = IRV_matrix_i[out_x][out_y];	ModeRunnable
Figure 62	Record_IRV.WrapInt = udisc;	RunnableEntity
	uMode = Record_IRV.WrapInt;	ModeRunnable
Figure 63	IRV_explicit = Record_IRVe.wrapInt; matrix_Out = IRV_matrix_e;	ModeRunnable

Table 4: ESDL code for access to interrunnable variables

Access to scalar interrunnable variables is declared within `<READ-LOCAL-VARIABLES>` and `<WRITTEN-LOCAL-VARIABLES>` elements. The example shown in Figure 59 results in the following description for the runnables `RunnableEntity` and `ModeRunnable`:

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>ModeRunnable</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <READ-LOCAL-VARIABLES>
      ...
      <VARIABLE-ACCESS>
         <SHORT-NAME>Read_IRV_explicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/bSWC/IRV_explicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>Read_IRV_implicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/bSWC/IRV_implicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
      ...
   </READ-LOCAL-VARIABLES>
   <SYMBOL>SWC_Impl_ModeRunnable</SYMBOL>
</RUNNABLE-ENTITY>
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <WRITTEN-LOCAL-VARIABLES>
      ...
      <VARIABLE-ACCESS>
         <SHORT-NAME>Write_IRV_explicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/bSWC/IRV_explicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>Write_IRV_implicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/bSWC/IRV_implicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
      ...
   </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

Listing 72: ARXML code – runnable entities with read (top) and write (bottom) access to scalar interrunnable variables

Access to interrunnable variables of array type is declared within `<READ-LOCAL-VARIABLES>` and `<WRITTEN-LOCAL-VARIABLES>` elements. The example shown in Figure 60 results in the following description for the runnables `ModeRunnable` and `RunnableEntity`:

```
<RUNNABLE-ENTITY>
    <SHORT-NAME>ModeRunnable</SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
    <READ-LOCAL-VARIABLES>
        <VARIABLE-ACCESS>
            <SHORT-NAME>Read_IRV_array</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/IRV_array</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
        ...
    </READ-LOCAL-VARIABLES>
    <SYMBOL>SWC_Impl_ModeRunnable</SYMBOL>
</RUNNABLE-ENTITY>
<RUNNABLE-ENTITY>
    <SHORT-NAME>RunnableEntity</SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
    ...
    <READ-LOCAL-VARIABLES>
        <VARIABLE-ACCESS>
            <SHORT-NAME>Read_IRV_array</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/IRV_array</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
        ...
    </READ-LOCAL-VARIABLES>
    ...
    <SYMBOL>SWC_Impl_RunnableEntity</SYMBOL>
    <WRITTEN-LOCAL-VARIABLES>
        <VARIABLE-ACCESS>
            <SHORT-NAME>Write_IRV_array</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/IRV_array</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
        ...
    </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

Listing 73: ARXML code – runnable entities with read (top, middle) and write (bottom) access to an implicit interrunnable variable of array type

Access to interrunnable variables of matrix type is declared in the same way as access to interrunnable variables of array type. The example shown in Figure 61 adds the following blocks to the descriptions of the runnables `ModeRunnable` and `RunnableEntity`:

```
<READ-LOCAL-VARIABLES>
   ...
   <VARIABLE-ACCESS>
      <SHORT-NAME>Read_IRV_matrix_i</SHORT-NAME>
      <ACCESSED-VARIABLE>
         <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/bSWC/IRV_matrix_i
         </LOCAL-VARIABLE-REF>
      </ACCESSED-VARIABLE>
   </VARIABLE-ACCESS>
   ...
</READ-LOCAL-VARIABLES>
...
<WRITTEN-LOCAL-VARIABLES>
   ...
   <VARIABLE-ACCESS>
      <SHORT-NAME>Write_IRV_matrix_i</SHORT-NAME>
      <ACCESSED-VARIABLE>
         <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/bSWC/IRV_matrix_i
         </LOCAL-VARIABLE-REF>
      </ACCESSED-VARIABLE>
   </VARIABLE-ACCESS>
   ...
</WRITTEN-LOCAL-VARIABLES>
```

Listing 74: ARXML code – read (top) and write (bottom) access to an implicit interrunnable variable of matrix type

The example shown in Figure 63 adds the following blocks to the descriptions of the runnable `ModeRunnable`:

```
<READ-LOCAL-VARIABLES>
   ...
   <VARIABLE-ACCESS>
      <SHORT-NAME>Read_IRV_matrix_e</SHORT-NAME>
      <ACCESSED-VARIABLE>
         <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/bSWC/IRV_matrix_e
         </LOCAL-VARIABLE-REF>
      </ACCESSED-VARIABLE>
   </VARIABLE-ACCESS>
</READ-LOCAL-VARIABLES>
...
<WRITTEN-LOCAL-VARIABLES>
...
<VARIABLE-ACCESS>
      <SHORT-NAME>Write_IRV_matrix_e</SHORT-NAME>
      <ACCESSED-VARIABLE>
         <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/bSWC/IRV_matrix_e
         </LOCAL-VARIABLE-REF>
      </ACCESSED-VARIABLE>
   </VARIABLE-ACCESS>
...
</WRITTEN-LOCAL-VARIABLES>
```

Listing 75: ARXML code – read (top) and write (bottom) access to an explicit interrunnable variable of matrix type

Access to interrunnable variables of record type is declared within `<READ-LOCAL-VARIABLES>` and `<WRITTEN-LOCAL-VARIABLES>` elements. The example shown in Figure 62 results in the following description for the runnables `ModeRunnable` and `RunnableEntity`:

```xml
<RUNNABLE-ENTITY>
    <SHORT-NAME>ModeRunnable</SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
    <READ-LOCAL-VARIABLES>
        ...
        <VARIABLE-ACCESS>
            <SHORT-NAME>Read_Record_IRV</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/Record_IRV</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </READ-LOCAL-VARIABLES>
    <SYMBOL>SWC_Impl_ModeRunnable</SYMBOL>
</RUNNABLE-ENTITY>
<RUNNABLE-ENTITY>
    <SHORT-NAME>RunnableEntity</SHORT-NAME>
    <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
    </SW-ADDR-METHOD-REF>
    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
    <DATA-READ-ACCESSS>
        ...
    </DATA-READ-ACCESSS>
    <DATA-WRITE-ACCESSS>
        ...
    </DATA-WRITE-ACCESSS>
    <READ-LOCAL-VARIABLES>
        ...
        <VARIABLE-ACCESS>
            <SHORT-NAME>Read_Record_IRV</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/Record_IRV</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </READ-LOCAL-VARIABLES>
    <SERVER-CALL-POINTS>
        ...
    </SERVER-CALL-POINTS>
    <SYMBOL>RteRunnable</SYMBOL>
    <WRITTEN-LOCAL-VARIABLES>
        ...
        <VARIABLE-ACCESS>
            <SHORT-NAME>Write_Record_IRV</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/bSWC/Record_IRV</LOCAL-VARIABLE-REF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

Listing 76: ARXML code – runnable entities with read (top, middle) and write (bottom) access to an interrunnable variable of record type

7.10 Exclusive Areas

Software components that need to provide mutual exclusion over data shared by two (or more) of their runnable entities do so by configuring exclusive areas.

The RTE generator uses exclusive area configuration to create operating system configuration files and to optimize exclusive areas. For example, if the only components that access a region are mapped to the same task, then the entire region can be elided.

Exclusive areas are defined in the XML configuration and are associated with the runnable entities that use them.

7.10.1 Configuration

Exclusive areas are created by means of ASCET resources.

To create an exclusive area

1. In the software component editor, use the **Resource** button to create a resource.
2. If you are working in the block diagram editor for SWC, place the resource in the drawing area.
3. In the "Outline" tab, right-click the resource, select **Rename** from the context menu and rename the resource to `SwcExclusiveArea`.

When the newly created exclusive area `SwcExclusiveArea` is used in the software component (see section 7.10.2), then the `<SWC-INTERNAL-BEHAVIOR>` declaration names the `<EXCLUSIVE-AREAS>` it uses.

```
<SWC-INTERNAL-BEHAVIOR>
   <SHORT-NAME>bSWC</SHORT-NAME>
   <DATA-TYPE-MAPPING-REFS>
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
      /ASCET_Mappings/DataMappings/Impl/SWC</DATA-TYPE-MAPPING-REF>
   </DATA-TYPE-MAPPING-REFS>
   <EXCLUSIVE-AREAS>
      <EXCLUSIVE-AREA>
         <SHORT-NAME>SwcExclusiveArea</SHORT-NAME>
      </EXCLUSIVE-AREA>
   </EXCLUSIVE-AREAS>
   <EVENTS>
      ...
   </EVENTS>
   <EXPLICIT-INTER-RUNNABLE-VARIABLES>
      ...
   </EXPLICIT-INTER-RUNNABLE-VARIABLES>
   <IMPLICIT-INTER-RUNNABLE-VARIABLES>
      ...
   </IMPLICIT-INTER-RUNNABLE-VARIABLES>
   <RUNNABLES>
      ...
   </RUNNABLES>
</SWC-INTERNAL-BEHAVIOR>
```

Listing 77: ARXML code – exclusive area definition

This means that the scope of any exclusive areas that you define is the software component instance. It is not possible to define exclusive areas that cross software component boundaries. Data that is shared between multiple software-component instances, which can potentially be accessed concurrently, should be encapsulated in its own component and then normal sender-receiver or client-server communication used to access the data.

Each exclusive area defined within an internal behavior definition must be named using the `<SHORT-NAME>` element. The name is used within other elements to reference the exclusive area and to form the "handle" by which the exclusive area is accessed at run-time. The short-name of an exclusive area should be a valid C identifier.

Additionally, the RTE can be informed how to implement the exclusive area with an *ExclusiveAreaImplementation* element within the ECU description.

> **NOTE**
>
> If the definition of the ExclusiveAreaImplementation for an exclusive area is omitted, then the RTE defaults to "OS resource" implementation strategy.

A different exclusive area implementation method can be set for each exclusive area and SWC instance.

> **NOTE**
>
> The InterruptBlocking method will cause all OS interrupts to be blocked in the worst case for the longest execution time of the protected critical section.

7.10.2 Usage

Each runnable in the `<SWC-INTERNAL-BEHAVIOR>` section can declare if it uses one of the named exclusive areas and how it uses the area at runtime.

ASCET defines exclusive areas with explicit access. The `<RUNNABLE-ENTITY-CAN-ENTER-EXCLUSIVE-AREA>` element determines that the exclusive area is accessed using an explicit API. The area's name forms part of the generated API (explicit access is similar to a standard resource in OSEK OS).

Since ASCET V6.2, exclusive areas can only be accessed by assigning sequences of a runnable entity in a user-defined exclusive area.

> **NOTE**
>
> Since ASCET V6.2, messages and the automatically generated exclusive area `ASCET_exclusive_area` are no longer available in software components.

<u>To assign sequences of a runnable in an exclusive area</u>

1. In the *block diagram editor* for SWC, proceed as follows:

i. Edit the sequence call `reserve` of the `SwcExclusiveArea` and provide the sequence number **8** in the `RunnableEntity` method.

ii. Edit the sequence call `release` of the `SwcExclusiveArea` and provide the sequence number **22** in the `RunnableEntity` method.

2. In the *ESDL editor* for SWC, proceed as follows:

i. In the "Outline" tab, select the runnable `RunnableEntity`.

ii. Enter the code you want to place in the exclusive area between the following lines:

```
SwcExclusiveArea.reserve();
   // your code inside the exclusive area, e.g.,
   SpeedSwc = Receiver.Speed;
   Client.MaximumValue(A, B, C);
SwcExclusiveArea.release();
```

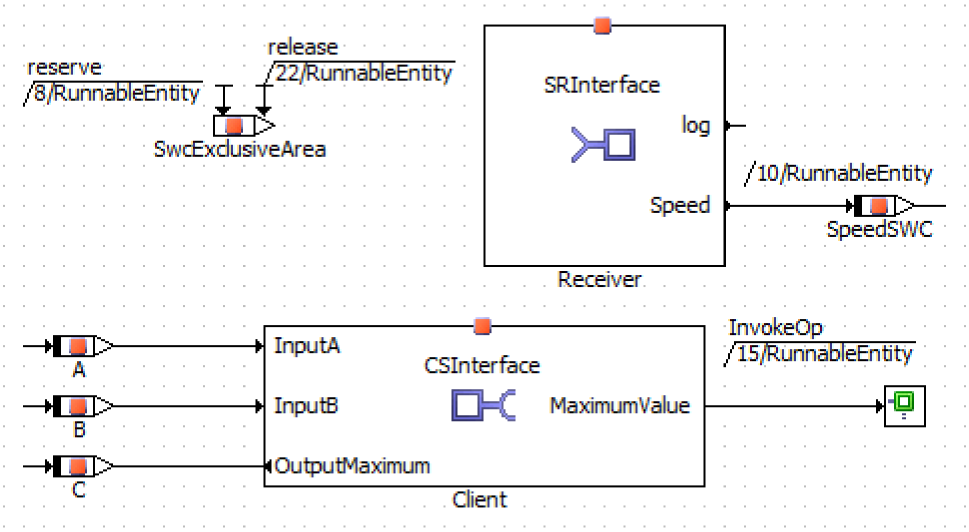


Figure 64: Use of the exclusive area `SwcExclusiveArea` in `RunnableEntity` (block diagram editor for SWC)

In the definition of the `<RUNNABLE-ENTITY>` element, the reference to the `SwcExclusiveArea` is generated as shown in Listing 78.

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-ENTER-EXCLUSIVE-AREA-REFS>
      <CAN-ENTER-EXCLUSIVE-AREA-REF DEST="EXCLUSIVE-AREA">
      /ASCET_ComponentTypes/SWC/bSWC/SwcExclusiveArea
      </CAN-ENTER-EXCLUSIVE-AREA-REF>
   </CAN-ENTER-EXCLUSIVE-AREA-REFS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
   </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-READ-ACCESSS>
      ...
   </DATA-READ-ACCESSS>
   <DATA-WRITE-ACCESSS>
      ...
   </DATA-WRITE-ACCESSS>
   <READ-LOCAL-VARIABLES>
      ...
   </READ-LOCAL-VARIABLES>
   <SERVER-CALL-POINTS>
      ...
   </SERVER-CALL-POINTS>
   <SYMBOL>RteRunnable</SYMBOL>
   <WRITTEN-LOCAL-VARIABLES>
      ...
   </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

Listing 78: ARXML code – runnable entity with reference to exclusive area

For the ASCET-generated C code, refer to section 9.6, *Concurrency Control with Exclusive Areas*, on page 205.

7.11 Variant Handling

Since AUTOSAR R4.0, the standard describes variant handling. Elements in the component description can be annotated with information when an element is present or which value an attribute has, depending on the value of so-called *system constants*. These annotations are called *variation points*. The value of a system constant can also be used in the C code, if a corresponding `#define` directive is generated based on a *variation point proxy*.

Variant handling in ASCET is done by specifying elements with kind “System constant” and then adding conditions with such elements to the behavior specification. The code generator then analyzes which code needs to be executed for each variant, which data is needed, etc. The same approach is used for variant handling with AUTOSAR. The conditions of the variation points are derived from the usage in the model.

ASCET knows three different binding times: generation time, compile time, and run time. For AUTOSAR variant handling, only compile time is supported. The corresponding binding time for AUTOSAR is `PRE-COMPILE-TIME`.

To enable variant handling, the AUTOSAR options for system constant generation must be set as shown in Figure 65. See section 3.1.3, *Settings for the AUTOSAR XML Output*, on page 21 to find this option.

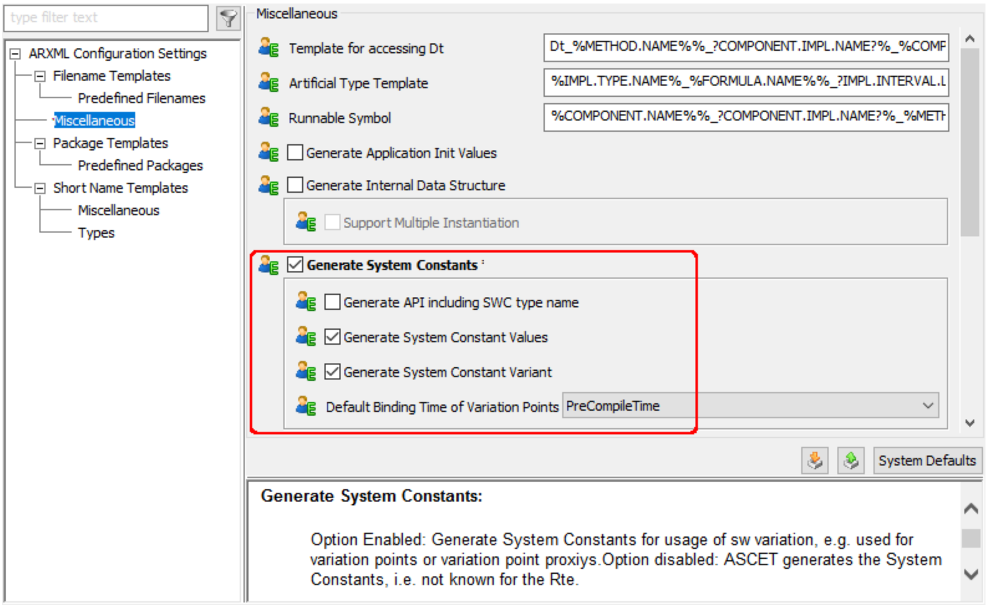


Figure 65: Variant handling - required settings for system constant generation

See also topic *Constants and System Constants* in the ASCET online help and section *Variant-Coded Data Structures* in the ASCET-SE User Guide.

7.11.1 Deriving the Conditions from the Model

ASCET includes a comprehensive analysis for the variant conditions, which takes the following information into account:

- Conditions with system constants in the behavior specification that affect the usage of AUTOSAR elements (data elements of data interfaces, operations of client/server interfaces, interrunnable variables)
- Conditions with system constants in the behavior specification that affect calls to methods or processes
- The mapping of AUTOSAR elements to messages and parameters

In the example below, the runnable calls a process only for a specific variant (Figure 66). The messages in that process are therefore only used conditionally, and this is propagated by the mapping to the AUTOSAR elements (Figure 67):

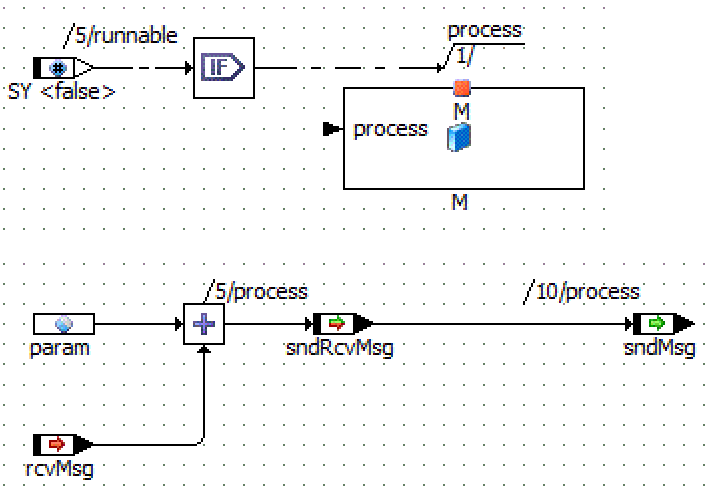


Figure 66: Conditional call of a process in a runnable (top) and usage of messages in the process (bottom)

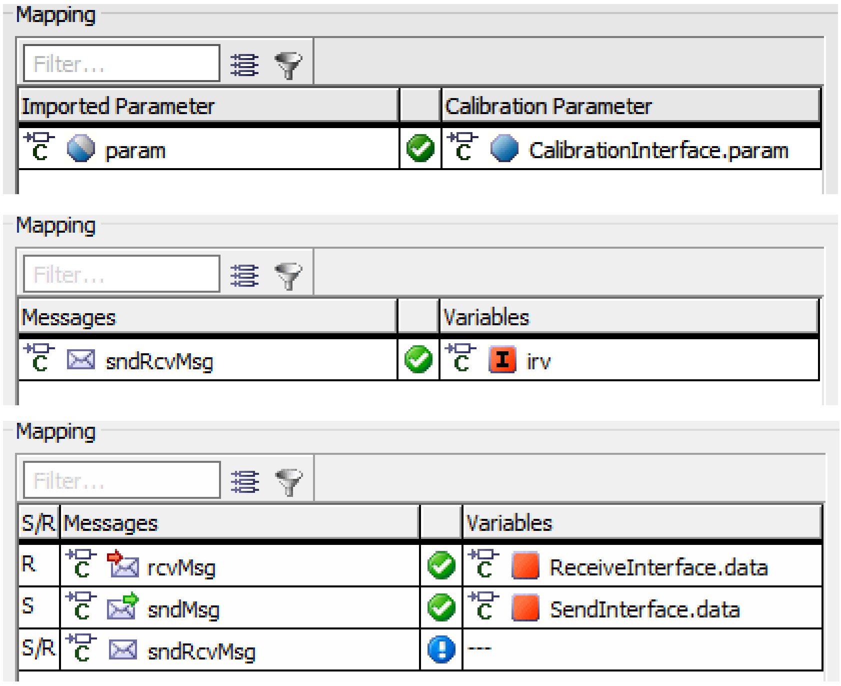


Figure 67: Mapping of messages and parameters for variant handling (from top to bottom: parameter mapping, internal message mapping, external message mapping)

7.11.2 System Constants

All system constants that are used in the conditions of variation points are defined in the AUTOSAR XML, in the `<swc name>_systemconstants.arxml` file. The package name, file name and short name are configured as described in section 3.1.3, *Settings for the AUTOSAR XML Output*, on page 21.

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_SystemConstants</SHORT-NAME>
    <ELEMENTS>
        <SW-SYSTEMCONST>
            <SHORT-NAME>SY_SWC_IMPL</SHORT-NAME>
            <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                    <SW-DATA-DEF-PROPS-CONDITIONAL>
                        <SW-CALIBRATION-ACCESS>READ-ONLY
                        </SW-CALIBRATION-ACCESS>
                        <COMPU-METHOD-REF DEST="COMPU-METHOD">
                        /ASCET_CompuMethods/ident</COMPU-METHOD-REF>
                        <DATA-CONSTR-REF DEST="DATA-CONSTR">
                        /ASCET_DataConstrs/Internal/dc_0to1</DATA-CONSTR-REF>
                        <IMPLEMENTATION-DATA-TYPE-REF DEST=
                        "IMPLEMENTATION-DATA-TYPE">
                        /AUTOSAR_Platform/ImplementationDataTypes/uint8
                        </IMPLEMENTATION-DATA-TYPE-REF>
                        <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
                    </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
        </SW-SYSTEMCONST>
    </ELEMENTS>
</AR-PACKAGE>
```

Listing 79: ARXML code – System Constant (AUTOSAR R4.2.2)

In addition, the default values for the system constants are also generated in `<swc name>_systemconstants.arxml` to provide values for the system constants:

```
<AR-PACKAGE>
    <SHORT-NAME>ASCET_SystemConstantValues</SHORT-NAME>
    <ELEMENTS>
        <SW-SYSTEMCONSTANT-VALUE-SET>
            <SHORT-NAME>Data</SHORT-NAME>
            <SW-SYSTEMCONSTANT-VALUES>
                <SW-SYSTEMCONST-VALUE>
                    <SW-SYSTEMCONST-REF DEST="SW-SYSTEMCONST">
                    /ASCET_SystemConstants/SY_SWC_IMPL
                    </SW-SYSTEMCONST-REF>
                    <VALUE>false</VALUE>
                </SW-SYSTEMCONST-VALUE>
            </SW-SYSTEMCONSTANT-VALUES>
        </SW-SYSTEMCONSTANT-VALUE-SET>
    </ELEMENTS>
</AR-PACKAGE>
```

Listing 80: ARXML code – System Constant value (AUTOSAR R4.2.2)

In addition to the built-in system constants, you can add user-defined system constants. A template file for the creation of user-defined system constants is given in the ASCET installation directory, `Tools\System Constant` subdirectory. See the ASCET online help for details on how to create user-defined system constants.

User-defined constants appear like an additional kind in the properties dialog (e.g., **My System Constant** in Figure 68).

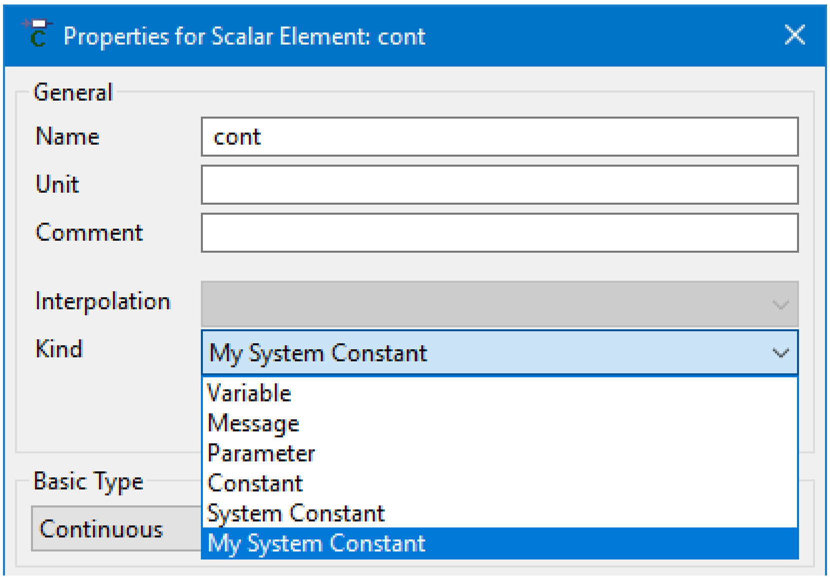


Figure 68: User-defined system constant **My System Constant** available for selection in the properties editor

Each user-defined system constant has its own binding time, which can deviate from the binding time selected in the ASCET options, "Targets\ ‹*target name*›\Build" node.

The binding time of a user-defined system constant can be set in the creation file and in the ASCET options window, "Modeling\User-defined System Constants\ ‹*system constant name*›" node. The available binding times are described in the description field of the ASCET options window.

7.11.3 Variation Points for Interrunnable Variables

If an interrunnable variable is only used for a specific variant, the corresponding variation point is added to the definition of the interrunnable variable in `<swc name>.arxml` (see also section 7.9, *Interrunnable Variables*, on page 130):

```
<VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
        ...
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/SInt32</TYPE-TREF>
    <INIT-VALUE>
        ...
    </INIT-VALUE>
    <VARIATION-POINT>
        <SHORT-LABEL>VP_IRV</SHORT-LABEL>
        <SW-SYSCOND BINDING-TIME="PRE-COMPILE-TIME">
            <SYSC-REF DEST="SW-SYSTEMCONST">
            /ASCET_SystemConstants/SY_SWC_IMPL</SYSC-REF>
        </SW-SYSCOND>
    </VARIATION-POINT>
</VARIABLE-DATA-PROTOTYPE>
```

Listing 81: ARXML code – variation point for an interrunnable variable (AUTOSAR R4.2.2)

7.11.4 Variation Points for Data Access

If a data element of a data interface (calibration, sender/receiver, NV data) is used conditionally, the corresponding access is annotated with a variation point in `<swc name>.arxml`:

```
<DATA-READ-ACCESSS>
    <VARIABLE-ACCESS>
        <SHORT-NAME>DataReadAccess1</SHORT-NAME>
        <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
                <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/ReceiveInterface
                </PORT-PROTOTYPE-REF>
                <TARGET-DATA-PROTOTYPE-REF DEST=
                "VARIABLE-DATA-PROTOTYPE">
                /ASCET_Interfaces/Impl/ReceiveInterface/data
                </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
        </ACCESSED-VARIABLE>
        <VARIATION-POINT>
            <SHORT-LABEL>VP_DataReadAccess1</SHORT-LABEL>
            <SW-SYSCOND BINDING-TIME="PRE-COMPILE-TIME">
            <SYSC-REF DEST="SW-SYSTEMCONST">
                /ASCET_SystemConstants/SY_SWC_IMPL</SYSC-REF>
            </SW-SYSCOND>
        </VARIATION-POINT>
    </VARIABLE-ACCESS>
</DATA-READ-ACCESSS>
```

Listing 82: ARXML code – variation point for a data access (AUTOSAR R4.2.2)

7.11.5 Variation Point Proxies

To make the value of a system constant available in the C code, a variation point proxy in `<swc name>.arxml` instructs the RTE generator to add a `#define` directive in the generated header files. The name of the preprocessor symbol follows the AUTOSAR naming convention and is therefore different from the regular ASCET system constant names.

```
<SWC-INTERNAL-BEHAVIOR>
  <SHORT-NAME>bSWC_var</SHORT-NAME>
  <DATA-TYPE-MAPPING-REFS>
    <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
    /ASCET_Mappings/DataMappings/Impl/SWC</DATA-TYPE-MAPPING-REF>
  </DATA-TYPE-MAPPING-REFS>
  <EVENTS>
    ...
  </EVENTS>
  <EXPLICIT-INTER-RUNNABLE-VARIABLES>
    ...
  </EXPLICIT-INTER-RUNNABLE-VARIABLES>
  <IMPLICIT-INTER-RUNNABLE-VARIABLES>
    ...
  </IMPLICIT-INTER-RUNNABLE-VARIABLES>
  <RUNNABLES>
    ...
  </RUNNABLES>
  <VARIATION-POINT-PROXYS>
    <VARIATION-POINT-PROXY>
      <SHORT-NAME>SY_SWC_IMPL</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <VALUE-ACCESS>
        <BOOLEAN-VALUE-VARIATION-POINT BINDING-TIME="PRE-COMPILE-TIME">
          <SYSC-REF DEST="SW-SYSTEMCONST">
          /ASCET_SystemConstants/SY_SWC_IMPL</SYSC-REF>
        </BOOLEAN-VALUE-VARIATION-POINT>
      </VALUE-ACCESS>
    </VARIATION-POINT-PROXY>
  </VARIATION-POINT-PROXYS>
</SWC-INTERNAL-BEHAVIOR>
```

Listing 83: ARXML code – variation point proxy (AUTOSAR R4.2.2)

If you generate an RTE with RTA-RTE V6.8, this results in the following generated code:

```
/* BEGIN: Condition Value Macros */
#define Rte_SysCon_SWC_SY_SWC_IMPL                (FALSE)
...
/* END: Condition Value Macros */
```

Listing 84: C code – `Rte_Cfg.h` with definition of system constant

```
/*******************************************************
 ***
 *** Constants
 ***
 *******************************************************/

#define Rte_SysCon_SY_SWC_IMPL Rte_SysCon_SWC_SY_SWC_IMPL
```

Listing 85: C code – `Rte_SWC.h` with definition of system constant

> **NOTE**
>
> Creating an RTE is not part of ASCET. See the RTA-RTE documentation for information.

As shown in this example, the data access is conditional, which means that the RTE access macros are also defined conditionally. As a consequence, the processes where these macros are used are also defined conditionally:

```
#if (Rte_SysCon_SY_SWC_IMPL) /* M_IMPL_process */
extern FUNC(void, SWC_CODE) M_IMPL_process (void);
#endif /* M_IMPL_process */
```

Listing 86: C code – `M.h` with conditional declaration of function

```
#if (Rte_SysCon_SY_SWC_IMPL) /* M_IMPL_process */
/* messages used by this process */
/* public process [] */

FUNC(void, SWC_CODE) M_IMPL_process (void)
{
    /* process: sequence call #5 */
    /* assignment to irv: min=-oo, max=+oo, hex=phys, limit=false,
    zero incl.=true */
    Rte_IrvIWrite_runnable_IRV(
        RTE_Prm_CalibrationInterface_param() +
        Rte_IRead_runnable_ReceiveInterface_data());
    /* process: sequence call #10 */
    /* assignment to SendInteface.data: min=-oo, max=+oo, hex=phys,
    limit=false, zero incl.=true */
    Rte_IWrite_runnable_SendInteface_data(Rte_IrvIRead_runnable_irv());
}
#endif /* M_IMPL_process */
```

Listing 87: C code – `M.c` with conditional definition of function

7.11.6 Variants

The description of one component does not require values for the system constants, but they are required to compile the software component. In the AUTOSAR model, all components on one ECU share the same variant, and the selection of the variant (in terms of values for the system constants) is therefore done in the ECU configuration.

To simplify testing, ASCET also generates this information; it is stored in the `<swc name>_variants.arxml` file. It is not intended to replace a proper variant management tool, which has the ability to define valid combinations of values of system constants, define constraints, and also track the test status for each variant.

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Variants</SHORT-NAME>
   <ELEMENTS>
      <PREDEFINED-VARIANT>
         <SHORT-NAME>Data</SHORT-NAME>
         <SW-SYSTEMCONSTANT-VALUE-SET-REFS>
            <SW-SYSTEMCONSTANT-VALUE-SET-REF DEST=
            "SW-SYSTEMCONSTANT-VALUE-SET">/ASCET_SystemConstantValues/Data
            </SW-SYSTEMCONSTANT-VALUE-SET-REF>
         </SW-SYSTEMCONSTANT-VALUE-SET-REFS>
      </PREDEFINED-VARIANT>
      <EVALUATED-VARIANT-SET>
         <SHORT-NAME>EV_Data</SHORT-NAME>
         <EVALUATED-ELEMENT-REFS>
            <EVALUATED-ELEMENT-REF DEST="APPLICATION-SW-COMPONENT-TYPE">
            /ASCET_ComponentTypes/SWC_var</EVALUATED-ELEMENT-REF>
         </EVALUATED-ELEMENT-REFS>
         <EVALUATED-VARIANT-REFS>
            <EVALUATED-VARIANT-REF DEST="PREDEFINED-VARIANT">
            /ASCET_Variants/Data</EVALUATED-VARIANT-REF>
         </EVALUATED-VARIANT-REFS>
      </EVALUATED-VARIANT-SET>
      <ECUC-MODULE-CONFIGURATION-VALUES>
         <SHORT-NAME>EMCV_Data</SHORT-NAME>
         <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcuDefs/EcuC
         </DEFINITION-REF>
         <CONTAINERS>
            <ECUC-CONTAINER-VALUE>
               <SHORT-NAME>ECV_Data</SHORT-NAME>
               <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">
               /AUTOSAR/EcuDefs/EcuC/EcucVariationResolver</DEFINITION-REF>
               <PARAMETER-VALUES></PARAMETER-VALUES>
               <REFERENCE-VALUES>
                  <ECUC-REFERENCE-VALUE>
                     <DEFINITION-REF DEST="ECUC-FOREIGN-REFERENCE-DEF">
                     /AUTOSAR/EcuDefs/EcuC/EcucVariationResolver/Predefined
                     VariantRef</DEFINITION-REF>
                     <VALUE-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">
                     /ASCET_Variants/Data</VALUE-REF>
                  </ECUC-REFERENCE-VALUE>
               </REFERENCE-VALUES>
            </ECUC-CONTAINER-VALUE>
         </CONTAINERS>
      </ECUC-MODULE-CONFIGURATION-VALUES>
   </ELEMENTS>
</AR-PACKAGE>
```

Listing 88: ARXML code – Variant (AUTOSAR R4.2.2)

8 Modes

The previous chapters have explored how an AUTOSAR software-component type can be defined and configured. In this chapter, you will learn how to define application modes that can be used by software components to control the execution of runnable entities.

This chapter summarizes the topics related with modes of the following sections:

- section 5.2, *Mode Switch*, on page 67
- section 7.1.3, *Mode-Switch Events*, on page 109

8.1 Defining Modes

Modes are declared within a <MODE-DECLARATION-GROUP> element contained in the AUTOSAR package ASCET_types. The ASCET_types package contains software-component-specific types.

In AUTOSAR R4.*, the ASCET_Types package is stored in the application types file of the software component, i.e. the generated file SWC_appltypes.arxml.

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Types</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
         <ELEMENTS>
            <MODE-DECLARATION-GROUP>
               ...
            </MODE-DECLARATION-GROUP>
            ...
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

Listing 89: ARXML code – mode declaration group

The <MODE-DECLARATION-GROUP> element is used to declare one or more modes that are subsequently used by interface declarations.

<u>**To create a mode group**</u>

1. In the component manager, select **Insert > AUTOSAR > Mode Group**.
2. Name the mode group OnOffMode.
3. Create two modes, off and on, as described on page 67.

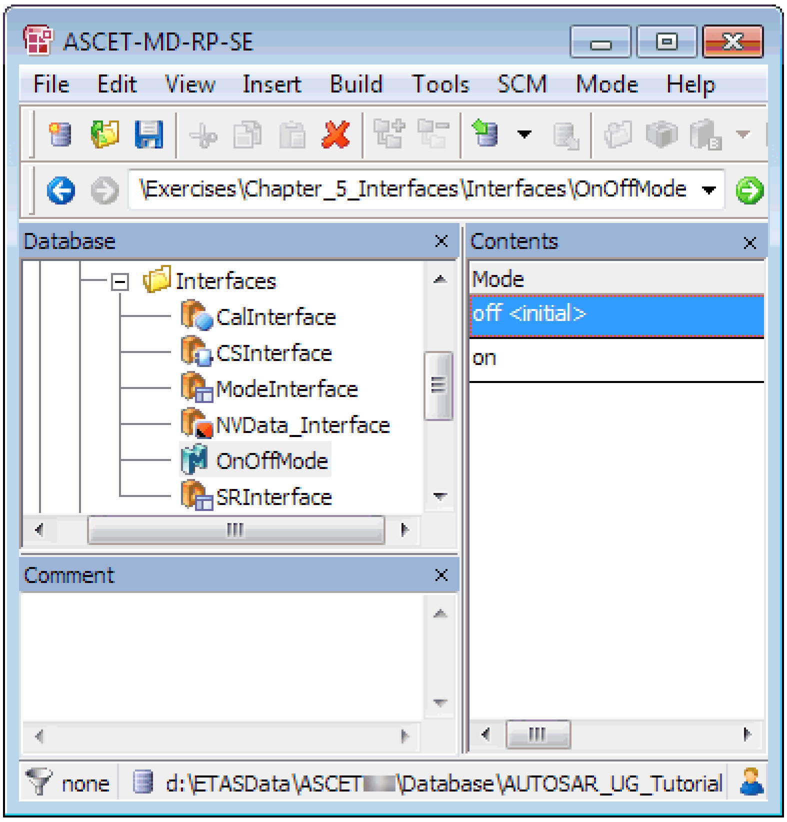


Figure 69: Mode declaration group `OnOffMode`

> **NOTE**
>
> A mode declaration group resembles an ASCET enumeration. In contrast to enumerations, the representing value cannot be set explicitly.

ASCET declares the `<MODE-DECLARATION-GROUP>` in the AUTOSAR package `ASCET_types`. See Listing 25 on page 68 for an AUTOSAR R4.0.2 ARXML example.

One mode within a `<MODE-DECLARATION-GROUP>` element is marked as the group's initial mode through the `<INITIAL-MODE-REF>`. Mode-Switch events that are attached to the `ENTRY` of an initial mode are triggered by the RTE when this is started using `Rte_Start`.

A `<MODE-DECLARATION-GROUP>` can be used (referenced) by multiple mode-switch interfaces and therefore inherently used by multiple software-components.

8.2 Mode Communication

Modes are communicated over a mode-switch interface (see section 5.2, *Mode Switch*, on page 67).

In ASCET, mode-switch interfaces are realized as sender-receiver interface components that contain mode groups.

In AUTOSAR R4.*, each mode-switch interface must specify one mode declaration group prototype.

Each mode declaration group prototype defines a prototype of a specific mode declaration group.

To create a mode group interface

1. In the component manager, select **Insert > AUTOSAR > SenderReceiver Interface**.
2. Name the sender-receiver interface `ModeInterface`.
3. Double-click `ModeInterface`.

   The "Sender Receiver Interface Editor for: ModeInterface" editor opens.
4. Select **Insert > Component**.

   The "Select Item ..." window opens.
5. In the "Database" or "Workspace" field of the "Select Item" window, select the mode group `OnOffMode` (see also Figure 21 on page 69).
6. Click **OK** to close the "Select Item" window and insert `OnOffMode` into `ModeInterface`.

   The "Properties for Element: OnOffMode" window opens.
7. Click **OK** to use the default name and comment.

   The mode group interface `ModeInterface` now looks as shown in Figure 22 on page 69.

The declaration of mode declaration group prototypes within a mode-switch interface definition has the structure shown in Listing 26 on page 70.

In AUTOSAR R4.*, a mode group is defined using the `<MODE-GROUP>` element.

Each `<MODE-GROUP>` must specify the following:

– the `<SHORT-NAME>` that you will use to refer to the item
– the `<TYPE-TREF>` reference to mode declaration group

In AUTOSAR R4.*, a sender-receiver interface component that contains a mode group must not contain data elements, and vice versa. Mixing both kinds of elements leads to a code generation error.

8.3 Using Modes

A software component can be a mode user activated in response to a mode switch. In this section, you learn how to use modes in a software component.

To insert a mode group interface in a software component

1. Create and set up a project as shown in section 3.1.2, *Code Generation Settings for AUTOSAR*, on page 17.
2. Insert a software component `Swc` in the project, as described in *To insert an AUTOSAR software component in a project* on page 23.
3. In the "Outline" tab of the project editor, double-click `Swc` to open the software component editor.
4. In the software component editor, select **Insert > Component**.

   The "Select item..." window opens.

5. In the "Database" or "Workspace" field of the "Select Item" window, select the interface `ModeInterface` and click **OK**.

   The "Properties for complex element: ModeInterface" window opens.

6. Make sure that **Required** is activated in the "Internal Access" area.

   > **NOTE**
   >
   > Mode interfaces can only be used as Rports. If you insert a mode interface as Pport, an error (`MMd11285`) is issued during code generation.

7. Click **OK** to add the mode interface.

8.3.1 Software Component Initialization and Finalization

AUTOSAR modes can be used to execute code when the RTE is started, e.g., to initialize internal data structures, etc. Similarly, when a system is shut down your software component may need to store data, log operational details, etc.

Each mode declaration group describes an initial mode – to activate a runnable when the system is started created by a `<SWC-MODE-SWITCH-EVENT>` for entry to the initial mode.

To create a mode-switch event

1. In the "Software Component Editor for: Swc", go to the "Event Specification" tab.
2. Select **Event > Add Event** and name the event `ModeEvent`.
3. In the "Event Kind" combo box, select `ModeSwitch`.
4. Set the following mode switch settings (see also Figure 45 on page 109):
   - Activation: **entry**
   - Assigned Mode: `on::OnOffMode`

A runnable entity within a software component can be started when the RTE is started by declaring a `<SWC-MODE-SWITCH-EVENT>` for entry to an initial mode.

8.3.2 Triggering a Runnable Entity on a Mode-Switch

A runnable entity can be activated on either entry to or exit from a mode using a Mode-Switch event configured, like all other events, in the `<SWC-INTERNAL-BEHAVIOR>` element of a software component.

To create a runnable entity:

1. In the "Software Component Editor for: Swc", select a diagram (e.g., `Main`) in the "Outline" tab.
2. Select **Insert > Runnable** and name it `ModeRunnable`.

For details on runnable entities, refer to section 7.2, *Runnable Entities*, on page 110.

If `ModeRunnable` is defined for entry, the runnable entity must be of Category 1. This means that it must not make any (blocking) RTE calls nor access other application components.

Similarly, when a system is defined for exit, your software component may need to store data, log termination etc. The principle is the same as initialization, except that finalization is simply a transition to a new mode that is associated with shutdown.

To add a Mode-Switch event to a runnable

1. Go to the "Event Specification" tab of the "Software Component Editor for: Swc".
2. In the "Events" field, select the event `ModeSwitch`.
3. In the "Runnables" field, select the runnable `ModeRunnable`.
4. Select **Event > Assign Event** or click the **>>** button.

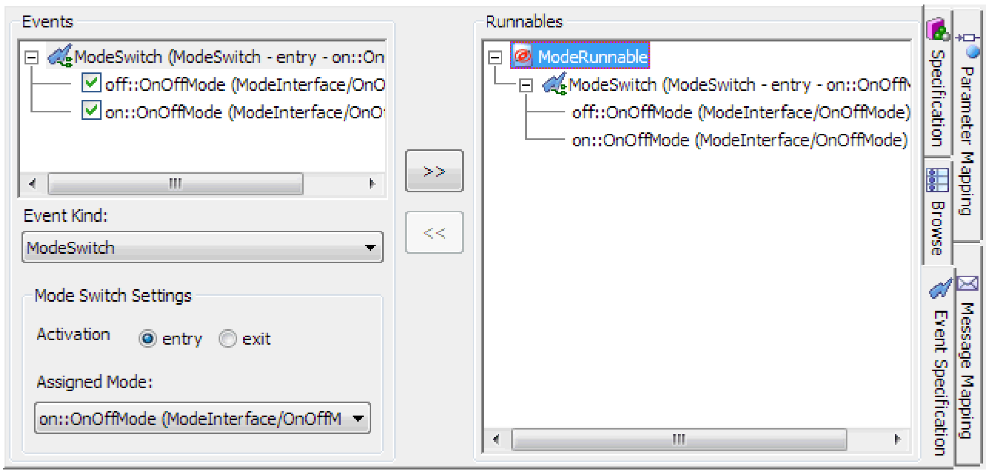


Figure 70: `ModeSwitch` is assigned to `ModeRunnable`

When the Mode-Switch event is mapped to a runnable entity, then ASCET generates the `<SWC-MODE-SWITCH-EVENT>` element in the configuration language as shown in Listing 48 on page 110.

A `<SWC-MODE-SWITCH-EVENT>` element defines the following things:

A. The `<START-ON-EVENT-REF>` element defines the runnable entity to be activated. The reference must be to a runnable entity within the same software component type.

B. The `<ACTIVATION>` element defines whether the runnable entity is triggered on entry to, or exit from, the mode. ASCET supports the text `ENTRY` or `EXIT`. A Mode-Switch event can apply either to entry to a mode or exit from a mode, but not to both. If runnable activation is required for entry and exit, then two Mode-Switch events must be defined.

C. The `<MODE-IREF>` element defines the mode associated with the Mode-Switch event. The `<MODE-IREF>` element must contain three references (the port prototype, the mode declaration group prototype, and the mode declaration group that types the declaration group prototype).

One mode within a `<MODE-DECLARATION-GROUP>` element is marked as the group's initial mode. Any Mode-Switch events that are attached to the *entry* of an initial mode within any group are triggered by the RTE when this is started using `Rte_Start`.

> **NOTE**
>
> When more than one runnable entity is triggered by the same mode entry (or exit), the order of execution of runnable entities is not defined. For portability, therefore, a system should not rely on a particular execution order.

8.3.3 Disabling Modes

A `<DISABLED-MODE-IREFS>` element permits the behavior of an event to be different in different modes. This allows such use cases as the activation of a runnable entity to be suppressed/permitted when a certain mode is active.

To disable the activation of a runnable

1. In the software component editor, go to the "Event Specification" tab.
2. In the "Events" pane, select the event `ModeEvent`.
3. Disable the mode `off`.

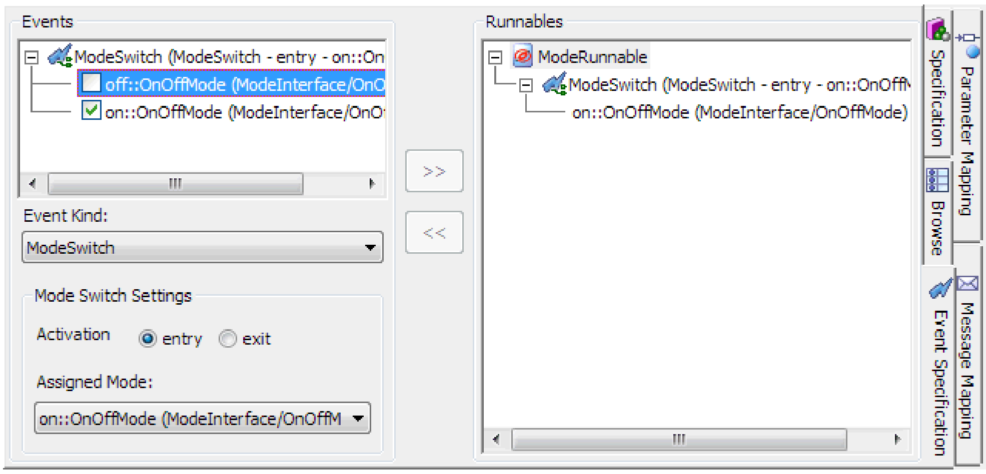


Figure 71: Mode `off` disabled in `ModeEvent`

The `<DISABLED-MODE-IREFS>` element specifies the disabled modes:

```
<SWC-MODE-SWITCH-EVENT>
    <SHORT-NAME>ModeEvent</SHORT-NAME>
    <DISABLED-MODE-IREFS>
        <DISABLED-MODE-IREF>
            <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/ModeInterface</CONTEXT-PORT-REF>
            <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
            "MODE-DECLARATION-GROUP-PROTOTYPE">
            /ASCET_Interfaces/Impl/ModeInterface/OnOffMode
            </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
            <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
            /ASCET_Types/ApplicationDataTypes/OnOffMode/off
            </TARGET-MODE-DECLARATION-REF>
        </DISABLED-MODE-IREF>
    </DISABLED-MODE-IREFS>
    <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
    /ASCET_ComponentTypes/SWC/bSWC/ModeRunnable</START-ON-EVENT-REF>
    <ACTIVATION>ON-ENTRY</ACTIVATION>
    <MODE-IREFS>
        <MODE-IREF>
            <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/ModeInterface</CONTEXT-PORT-REF>
            <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
            "MODE-DECLARATION-GROUP-PROTOTYPE">
            /ASCET_Interfaces/Impl/ModeInterface/OnOffMode
            </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
            <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
            /ASCET_Types/ApplicationDataTypes/OnOffMode/on
            </TARGET-MODE-DECLARATION-REF>
        </MODE-IREF>
    </MODE-IREFS>
</SWC-MODE-SWITCH-EVENT>
```

Listing 90: ARXML code – definition of a Mode-Switch event with disabled mode

When the mode specified within the `<DISABLED-MODE-IREFS>` element is active, the RTE will not activate the runnable (the activation is discarded).

For more information about the implementation of mode instances, please refer to the RTA-RTE User Guide.

9 Implementing Software Components

This chapter shows how to model software components in ASCET so that the objects required by the RTE are declared, and how to use the RTE API generated by the RTE generator.

> **NOTE**
>
> The generated C code may differ for different `RTE-AUTOSAR R*` versions used as OS (cf. page 18), depending on the settings in the respective memory sections definition file (see *To define a memory sections definition file* on page 20).
>
> All C code examples in this chapter have been generated with `RTE-AUTOSAR R4.2.2`, unless explicitly stated otherwise.

9.1 Basic Concepts

9.1.1 Namespace

All RTE symbols (e.g., function names, global variables etc.) that are visible in the global namespace use either the prefix `Rte_` or the prefix `RTE_`.

> **NOTE**
>
> You must not create symbols that use either the prefix `Rte_` or the prefix `RTE_`, to avoid the possibility of namespace clashes.

9.1.2 Runnable Naming Convention

The RTE generator generates code that activates your runnable entities. To do this, the RTE's internal mechanisms need to be able to access your code through defined interfaces.

Each of the named runnable entities defined in your runnable entity `<SYMBOL>` declarations must be implemented. Failure to define all runnable entities will be detected at compile time when your application is linked to form the ECU's executable image. The linker error message will reference the missing runnable entity entry point.

Runnable entities are executed by RTE-generated code when required. The function providing an entry point for a runnable entity should not be invoked directly by an application software component.

Like for all executable elements in the model, a symbol can also be specified in the implementation settings for a runnable and will then take precedence over the default naming scheme.

9.1.3 API Naming Convention

The RTE API calls are generated for each software component using names derived from the RTE generator's input. The RTE API provides a consistent interface to each software component, but allows the RTE generator to provide different implementations of the API functionality.

Each API call name is formed from the following parts:

– prefix `Rte_`
– call functionality (read, write, etc.)
– Either
    - port name and data item name (sender-receiver) or operation name (client-server) through which the call operates

  or

    - name of the object (e.g., exclusive area) upon which the call operates

Thus, RTE API calls involving communication through ports have the format:

```
Rte_StatusType
        Rte_<API call name>_<port>_<dataitem/operation>
```

Whereas other RTE APIs have the format:

```
Rte_StatusType
        Rte_<API call name>_<object name>
```

9.1.4 API Parameter Passing Mechanisms

The RTE API calls may have one or more parameters. The API parameters (if any) fall into one of three classes:

– *"In" Parameters* – All "in" parameters that are AUTOSAR primitive data types (with the exception of a string) are passed by value. Strings and other "in" parameters that are of a complex data type (i.e. a record, array, or matrix) are passed by reference.

  Note that while AUTOSAR defines a string as a primitive data type, its inherent size makes it inefficient to pass by value and is therefore treated the same as a complex data type.

  "In" parameters are strictly read-only.

– *"Out" Parameters* – All "out" parameters are passed to RTE API functions by reference. This is required to ensure that the API functions can modify the parameter.

  "Out" parameters are strictly write-only.

– *"In/Out" Parameters* – All "in/out" parameters are passed to the RTE API functions by reference except for an asynchronous client-server call when primitive data types (other than strings) are passed by value to Rte_Call and by reference to Rte_Result.

  "In/out" parameters can be read and written by the API function being called.

NOTE

ASCET configures the identifiers of the API parameters in the XML configuration file specified in the project properties (see *To define a memory sections definition file* on page 20). The standard configuration of the AUTOSAR memory sections is provided in the exemplary files `memorySections_Autosar.xml` and `memorySections_Autosar4.xml`.

When generating code in an AUTOSAR project, ASCET loads the memory sections defined in the specified XML file. Changes in the `*.xml` file will only be considered if you perform **Build › Touch › Recursive** before the code generation is started.

9.2 Application Source Code

ASCET is a C code generator, and the RTE also generates C code. ASCET V6.4 supports, at present, single-instance software components.

9.2.1 Application Header Files

Each software component generated in ASCET includes the relevant application header file `<SWCname>.h` created during RTE configuration.

```
/*----------------------------------------------------
 *    Include files
 *----------------------------------------------------*/

#include "a_basdef.h"
#include "Rte_SWC.h"
#include "chartab.h"
```

Listing 91: C code – include application header file (`<SWCname>.h`)

The RTE API is specific to each software component type. Therefore, it must be included only in the component's application header file for each source code file that defines a component (whether completely or partially).

NOTE

ASCET includes the header files in the application software when exporting the generated code into a storage directory (see how to generate code in a project in section 3.1.4, *Code Generation*, on page 22). The user shall not use intermediate files taken from the code generation directory.

A single source module must not include multiple application header files, as the API mappings they contain may be different for different software components. The header files generated by the RTE generator protect against such multiple file inclusion.

The component type specific header file defines the component's RTE API.

9.2.2 Entry Point Signature for Runnable Entities

The user models in ASCET the implementation of the runnables in the software component. ASCET generates the source code of all the runnable entities required to make a software component work at runtime.

ASCET provides an entry point (i.e. a C function) for each <RUNNABLE-ENTITY> declared in the component description.

```
/*********************************************************************
 * BEGIN: DEFINITION OF RUNNABLE 'RteRunnable'
 * -------------------------------------------------------------------
 * model name:...............................'RunnableEntity'
 * memory class:.............................'CODE'
 * ------------------------------------------------------------------*/
/* messages used by this runnable */
/* public RunnableEntity_ ()  */

FUNC(void, SWC_CODE) RteRunnable (void)
{
   ...
}
/* -------------------------------------------------------------------
 * END: DEFINITION OF RUNNABLE 'RteRunnable'
 ********************************************************************/
```

Listing 92: C code – entry point for runnable entity

The signature of a runnable entity's entry point function follows the following implementation rules:

- There are no user-defined parameters.
- There is no return value (i.e. a return type of `void` must be specified).
- The memory class must be `CODE`.

All RTE events other than Operation-Invoked events use the same basic signature for runnable entity entry points, irrespective of the event that actually triggers the runnable entity.

If the runnable entity responds to an <OPERATION-INVOKED-EVENT>, then additional parameters may be required.

```
/**********************************************************************
*
 * BEGIN: DEFINITION OF RUNNABLE 'SWC_Impl_Server_MaximumValue'
 *
 ----------------------------------------------------------------------
 * model name:..................................'Server_MaximumValue'
 * memory class:................................'CODE'
 *
 ---------------------------------------------------------------------*/
/* messages used by this runnable */
/* public Server_MaximumValue
          (InputA::sdisc;InputB::sdisc;OutputMaximum::sdisc)  */

FUNC(void, SWC_CODE) SWC_Impl_Server_MaximumValue (
   /* IN     */ VAR(sint16, AUTOMATIC)                        InputA,
   /* IN     */ VAR(sint16, AUTOMATIC)                        InputB,
   /* OUT    */ CONSTP2VAR(sint16, AUTOMATIC, RTE_APPL_DATA) OutputMaximum
   )
{
   /* Server_MaximumValue: sequence call #5  */
   /* assignment to OutputMaximum: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: false, assign: true), zero incl.=true  */
   (*OutputMaximum) = ((InputA >= InputB) ? InputA : InputB);
}
```

Listing 93: C code – server runnable entity

The signature of a runnable entity entry point function invoked as a result of an Operation-Invoked event follows the following implementation rules:

- There is a return value when a server specifies application errors, in which case `Std_ReturnType` is used.
- Formal parameters are the operations' IN, IN/OUT and OUT parameters. These parameters are passed by value or reference depending on the type.
- The memory class must be `CODE`.

9.3 Sender-Receiver Communication

The RTE API calls for handling non-queued sender-receiver communication differ for the type of data access.

- Non-queued communication with explicit access
  - Send with `Rte_Write`
  - Receive with `Rte_Read`
  - Receive with `Rte_DRead`

  Non-queued communication with explicit access can be optionally implemented with status.
- Non-queued communication with implicit access
  - Send with `Rte_IWrite`
  - Receive with `Rte_IRead`

  The implicit API uses a locally cached copy of data to preserve consistency over a calling runnable entity invocation. Data is read into a global cache before the runnable entity starts executing and is written from the global cache after the runnable entity terminates. Data writes are done once, no matter how many times it is written.

The RTE guarantees cached data does not change during execution of the runnable entity.

The implicit API should be used when you need to guarantee that every access to a datum in a runnable entity will provide the same result irrespective of how many times it is accessed during an invocation of the runnable entity.

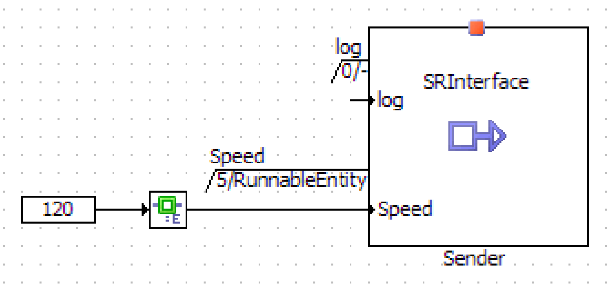
The following sections show how to use these types of data access in your application.

9.3.1 Sending to a Port: Explicit Communication

Components communicate data to other components using the `Rte_Write` call. The call is defined per port and interfaces data item for each component and therefore has the following signature:

```
Rte_StatusType Rte_Write_<Port>_<DataItem>(DataItemType Data)
```

For the example of section 7.4.1, *Explicit Communication*,



(block diagram editor for SWC)

ASCET generates the following C code:

```
FUNC(void, SWC_CODE) RteRunnable (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   _ASCET_RteStatus = Rte_Write_Sender_Speed(120);
   ...
}
```

Listing 94: C code – explicit send (example of section 7.4.1, *Explicit Communication*)

9.3.2 Sending to a Port: Explicit Communication with Status

Explicit access can be optionally implemented with status.

<u>To set explicit communication with status (block diagram editor for SWC)</u>

1. Open the `ARProject` project and `Swc` software component from the example in section 7.4.1, *Explicit Communication*, on page 114.
2. In the drawing area of the software component editor, right-click the `RTE access` operator and select **Access › Explicit with Status** from the context menu (see Figure 72).

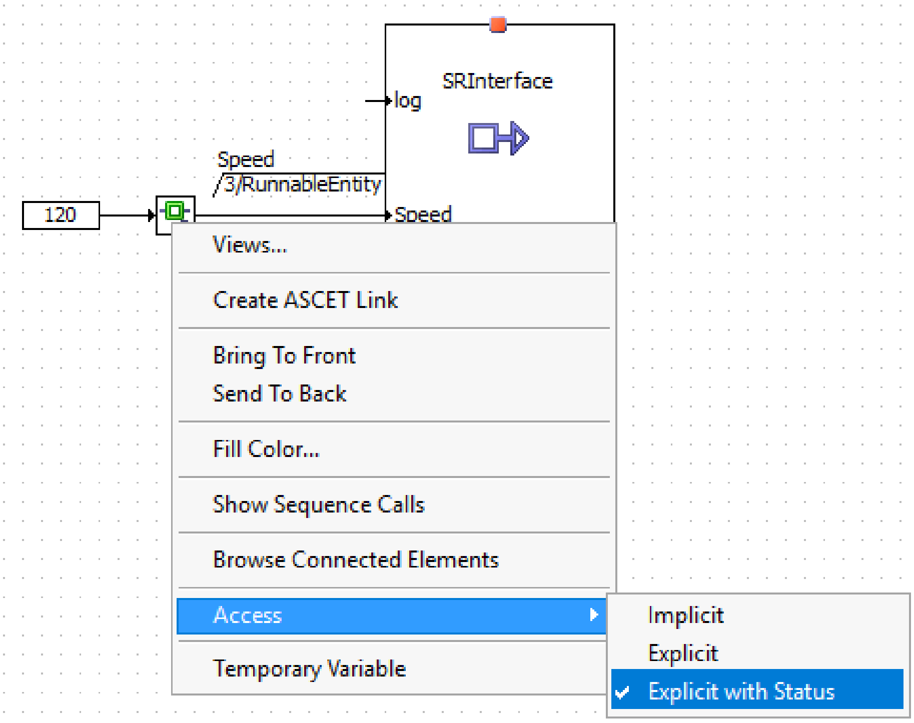


Figure 72: Setting explicit communication with status (block diagram editor for SWC)

3. Use the **RTE Status** button to create an RTE Status operator and place it in the drawing area.
4. Create a literal ( ) and place it in the drawing area.
5. Edit the literal (see the online help for details) and enter one of the status/error values listed in section 4.7.1, *Std_ReturnType*, on page 43.

   This example uses `RTE_E_NO_DATA`.
6. Add a logic variable ( ) named, e.g., `noData`.
7. Convert the variable's sequence call into a connector (see the online help for details).
8. Add an **Equal** operator.
9. Connect literal, RTE status block, operator, and variable as shown in Figure 73.
10. Connect the pin below the RTE Status block with the connector of the `noData` variable

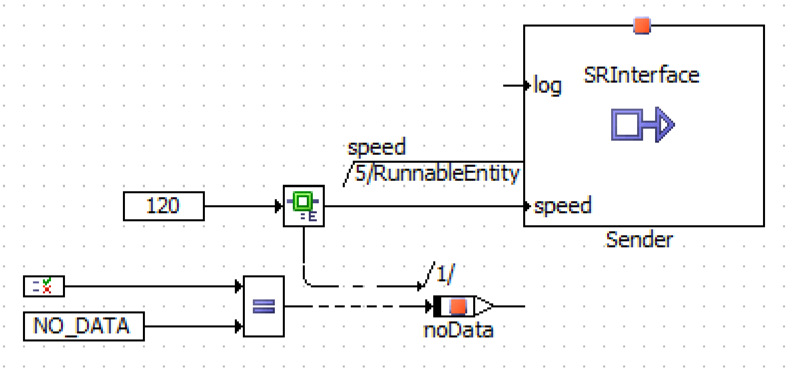


Figure 73: Sending a value 120 to a sender port using explicit communication with status (block diagram editor for SWC)

To set explicit communication with status (ESDL editor for SWC)

1. Open the `ARProject` project and `Swc` software component from the example in section 7.4.1, *Explicit Communication*, on page 114.
2. Declare a method-/runnable-local variable of type `Std_ReturnType`:

   ```
   Std_ReturnType rteStatus;
   ```

   This variable will hold the status from the explicit write procedure.
3. Enter the following code to use explicit write with status information:

   ```
   rt eStatus = Sender.Speed.explicitWrite(120);
   ```

4. Specify the status inquiry as follows:

   ```
   if (rteStatus != RTE_E_OK)
   {
       noData = rteStatus == RTE_E_NO_DATA;
   }
   ```

For the example, ASCET generates the following C code:

```
...
#define _noData (SWC_RAM.noData)

...

FUNC(void, SWC_CODE) RteRunnable (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   _ASCET_RteStatus = Rte_Write_Sender_Speed(120);
   if (_ASCET_RteStatus != RTE_E_OK)
   {
      /* RTE_ExplicitWithStatus-block: sequence call #Explicit write
      error/Status #1  */
      _noData = _ASCET_RteStatus == RTE_E_NO_DATA;
   } /* end if */
   ...
}
```

Listing 95: C code – explicit send with status

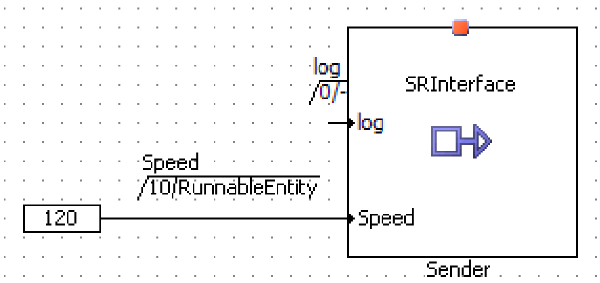
9.3.3 Sending to a Port: Implicit Communication

The implicit API includes a reference to the runnable entity that is declared as accessing the data in the API name. Care should be taken when writing a runnable entity to invoke the correct API. The `Rte_IWrite` API reads data:

```
Rte_StatusType
     Rte_IWrite_<runnable>_<port>_<data>( DataItemType Data)
```

The cache is updated before the runnable entity starts. `Rte_IWrite` writes data to a cached copy and changes are only made visible after the runnable entity terminates irrespective of the number of times the data is written.

For the example of section 7.4.2, *Implicit Communication*,



(block diagram editor for SWC)

ASCET generates the following C code:

```
FUNC(void, SWC_CODE) RteRunnable (void)
{
   ...
   /* RunnableEntity: sequence call #5  */
   /* assignment to Sender.Speed: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: false, assign: true), zero incl.=true  */
   Rte_IWrite_RunnableEntity_Sender_Speed(120);
   ...
}
```

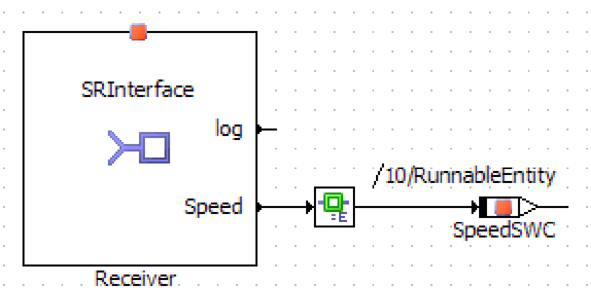
Listing 96: C code – implicit send (example of section 7.4.2, *Implicit Communication)*

9.3.4 Receiving from a Port: Explicit Communication

Components receive communicated data items from other components using the `Rte_Read` call. The call is defined per port and interfaces data item for each component and therefore has the following signature:

```
Rte_StatusType Rte_DRead_<Port>_<DataItem>()
```

For the example of section 7.5.1, *Explicit Data Read Access*,



(block diagram editor for SWC)

ASCET generates the following C code:

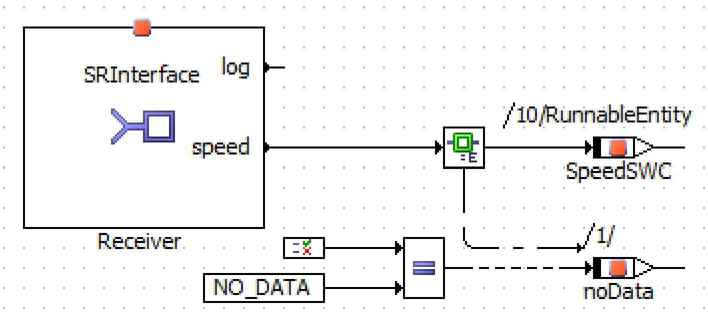
```
...
#define _SpeedSWC (SWC_RAM.SpeedSWC)

...

FUNC(void, SWC_CODE) RteRunnable (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #10  */
   _SpeedSWC = Rte_DRead_Receiver_Speed();
...
}
```

Listing 97: C code – explicit receive (example of section 7.5.1, *Explicit Data Read Access*)

9.3.5 Receiving from a Port: Explicit Communication with Status

Explicit access can be optionally implemented with status. To set explicit communication with status, see the example in section 9.3.2, *Sending to a Port: Explicit Communication with Status*, on page 177.

When setting explicit communication with status to the example of the previous section,



(block diagram editor for SWC)

ASCET generates the following C code:

```
...
#define _noData (SWC_RAM.noData)
...
#define _SpeedSWC_REF_ (&(SWC_RAM.SpeedSWC))

...

FUNC(void, SWC_CODE) RteRunnable (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #10  */
   _ASCET_RteStatus = Rte_Read_Receiver_Speed(_SpeedSWC_REF_);
   if (_ASCET_RteStatus != RTE_E_OK)
   {
      /* RTE_ExplicitWithStatus-block: sequence call #Explicit write
      error/Status #1  */
      _noData = _ASCET_RteStatus == RTE_E_NO_DATA;
   } /* end if */
   ...
}
```

Listing 98: C code – explicit receive with status

`Rte_Read` is non-blocking even if no data is present to read. If no data is present, the return value from the call is `RTE_E_NO_DATA`.

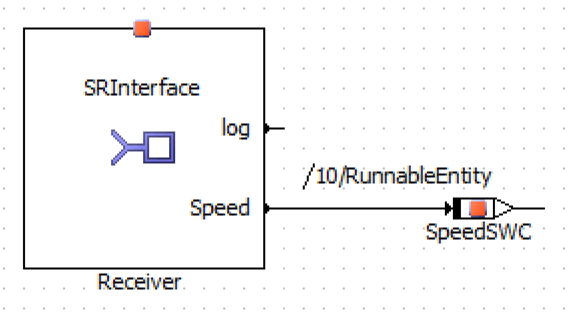
9.3.6 Receiving from a Port: Implicit Communication

The implicit API includes a reference to the runnable entity that is declared as accessing the data in the API name. Care should be taken when writing a runnable entity to invoke the correct API. The `Rte_IRead` API reads data:

`DataItemType Rte_IRead_`*`<runnable>`*`_`*`<port>`*`_`*`<data>`*`()`

The cache is updated before the runnable entity starts and therefore within a single execution of a runnable entity the value returned by `Rte_IRead` is guaranteed not to change.

For the example of section 7.4.2, *Implicit Communication*,



(block diagram editor for SWC)

ASCET generates the following C code:

```
...
#define _SpeedSWC (SWC_RAM.SpeedSWC)

...

FUNC(void, SWC_CODE) RteRunnable (void)
{
   ...
   /* RunnableEntity: sequence call #10  */
   /* assignment to SpeedSWC: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: false, assign: true), zero incl.=true  */
   _SpeedSWC = Rte_IRead_RunnableEntity_Receiver_Speed();
   ...
}
```

Listing 99: C code – implicit receive (example of section 7.4.2, *Implicit Communication)*

9.4 Client-Server Communication

Client-server communication is initiated using the `Rte_Call` API call.

When the `CLIENT_MODE` is set to synchronous, then `Rte_Call` returns after the operation has been completed by the server. This means that your code will not continue to execute until the server returns the result. Once the result has been computed, it is passed back to the component by the return value of the `Rte_Call`.

```
Rte_StatusType Rte_Call_<Port>_<Operation>(  InParam1Type  In_1,
...,
InParamNType  In_N,
OutParam1Type Out_1,
...,
OutParamMType Out_M)
```

9.4.1 Implementing a Server Operation

Each component that defines a server port must implement a runnable entity that responds to an Operation-Invoked event. The signature of the runnable entity must conform to the rules defined in section 9.2.2, *Entry Point Signature for Runnable Entities*, on page 175.

The following instruction explains how to implement the `Server_MaximumValue` runnable of section 7.7 *Responding to a Server Request on a Port*.

.

To implement a server operation

1. Create a Pport `Server` as described in *To create a server port* on page 93.
2. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Load the diagram `Server_CSInterface`.
   ii. Implement the operation `Server_MaximumValue` as shown in Figure 74.
3. If you are working in the *ESDL editor for SWC*, proceed as follows:
   i. Open the `Server_MaximumValue` operation.

ii. Implement the operation as follows:

```
OutputMaximum = InputA.max(InputB);
```

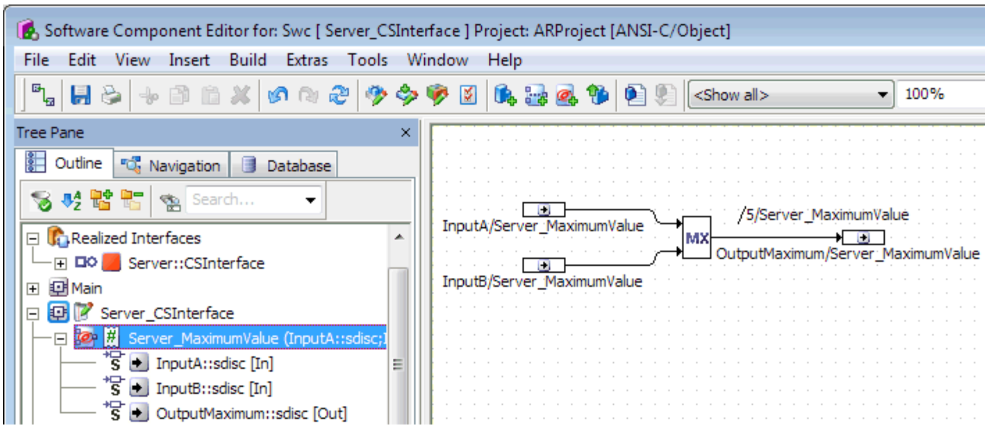


Figure 74: Implementation of the operation `Server_MaximumValue` in the diagram `Server_CSInterface` (block diagram editor for SWC)

For the operation `Server_MaximumValue`, ASCET generates the following server runnable:

```
/***************************************************************************
*
 * BEGIN: DEFINITION OF RUNNABLE 'SWC_Impl_Server_MaximumValue'
 *
 -----------------------------------------------------------------------
 * model name:...................................'Server_MaximumValue'
 * memory class:.................................'CODE'
 *
 ------------------------------------------------------------------------*/
/* messages used by this runnable */
/* public Server_MaximumValue
          (InputA::sdisc;InputB::sdisc;OutputMaximum::sdisc)   */

FUNC(void, SWC_CODE) SWC_Impl_Server_MaximumValue (
   /* IN    */ VAR(sint16, AUTOMATIC)                        InputA,
   /* IN    */ VAR(sint16, AUTOMATIC)                        InputB,
   /* OUT   */ CONSTP2VAR(sint16, AUTOMATIC, RTE_APPL_DATA) OutputMaximum
   )
{
   /* Server_MaximumValue: sequence call #5  */
   /* assignment to OutputMaximum: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: false, assign: true), zero incl.=true  */
   (*OutputMaximum) = ((InputA >= InputB) ? InputA : InputB);
}
```

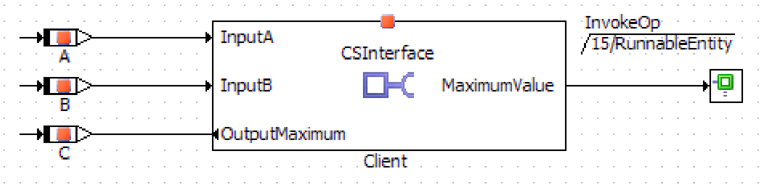
Listing 100: C code – server runnable

Servers may be invoked from multiple sources, for example, through a request from a client received via the communication service or directly via intra-task communication. Unless marked as concurrently executable within the runnable's configuration, the RTE will serialize access to the server, queuing requests on a first-in/first-out basis.

9.4.2 Making a Client Request on a Port

A runnable entity will be invoked by the RTE each time a request is made for an operation on the server's port.

For the example of section 7.8, *Making a Client Request on a Port*, on page 128,



(block diagram editor for SWC)

ASCET generates the following C code:

```
#define _A (SWC_RAM.A)
...
#define _B (SWC_RAM.B)
#define _C_REF_ (&(SWC_RAM.C))
...

FUNC(void, SWC_CODE) RteRunnable (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #15  */
   _ASCET_RteStatus = Rte_Call_Client_MaximumValue(_A,_B,_C_REF_);
   ...
}
```

Listing 101: C code – client request

9.5 Message, NV Variable, and Parameter Mapping

This section describes how ASCET elements, i.e. messages, non-volatile variables (*NV variables*), and imported parameters, are mapped to corresponding AUTOSAR elements.

– 9.5.1, *Accessing Calibration Parameters*, on page 185
– 9.5.2, *Accessing ASCET Messages*, on page 190
– 9.5.3, *Accessing Non-Volatile Variables*, on page 196
– 9.5.4, *Automatic Mapping*, on page 201

To ease reuse of ASCET modules and classes in SWC, it is possible to export mappings from one SWC and import them into another SWC. See the ASCET online help for details.

9.5.1 Accessing Calibration Parameters

If a software component declares calibration parameters, then each characteristic is accessed at runtime using the API call:

```
CalprmElementType Rte_Calprm_<Port>_<CalprmElement>()
```

The call returns either the calibration data (primitive types) or a pointer to the data (complex types).

Calibration data in a function is modeled by means of ASCET imported parameters. In an application software component, the calibration data can be mapped to the calibration parameters of an AUTOSAR calibration component. For this purpose, ASCET provides a special editor in the "Parameter Mapping" view of the software component editor.

In that editor, imported parameters can be mapped to AUTOSAR elements according to the following rules:

– Parameters can be mapped as shown in the following table.

Parameter	Mapping
Scalar	to scalar elements of calibration interfaces
	to scalar elements of complex elements (records) in calibration interfaces
Composite (arrays, matrices)	to composite elements of calibration interfaces
	to composite elements of complex elements (records) in calibration interfaces
Complex (records)	to complex elements (records) in calibration interfaces
	to complex elements of complex elements (records) in calibration interfaces

Table 5: Mapping possibilities for ASCET parameters

– A scalar parameter must be mapped to a scalar element of compatible type.

Imported parameter type	Calibration parameter type
Continuous (cont)	cont / limitInt / wrapInt / sdisc / udisc
Limited Integer (limitInt)	limitInt / wrapInt
Wrap-Around Integer (wrapInt)	wrapInt
Signed Discrete (sdisc)	cont / limitInt / wrapInt / sdisc / udisc
Unsigned Discrete (udisc)	cont / limitInt / wrapInt / sdisc / udisc
Logic (log)	log
Enumeration (enum)	Enumeration of the same type

Table 6: Parameter types and compatible AUTOSAR types

If you map a scalar parameter to an element of compatible, but non-identical type, a warning (`WMdl635`) is issued during code generation.

If you map a scalar parameter to an element of incompatible type, an error (`MMdl635`) is issued during code generation.

– A composite parameter (array, matrix) must be mapped to an array or matrix of identical size, data type, and implementation.

Otherwise, the mapping is indicated as invalid, and an error (`MMdl635`) is issued during code generation.

– A complex parameter (record) must be mapped to a record of identical type and implementation.

Otherwise, the mapping is indicated as invalid, and an error (`MMdl635`) is issued during code generation.

To create a function with parameters

1. In the component manager, select **Insert > Class > Block Diagram** to create a class.
2. Name the class `ClassWithParam`.
3. Open `ClassWithParam` in the block diagram editor.
4. Use the **Logic Parameter** button to create a logic parameter.

   The dialog "Properties for Scalar Element: log" opens.
5. Name the parameter `localLog` and change the scope to **Imported**.

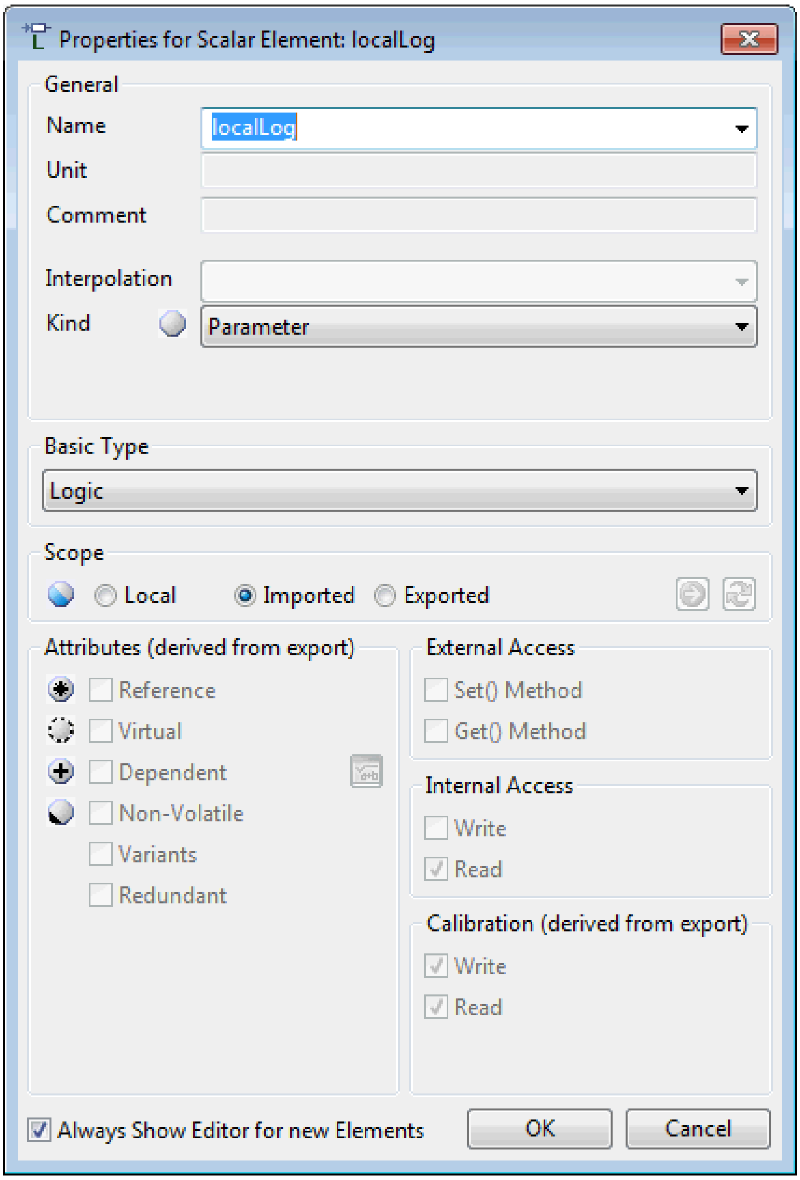


Figure 75: Parameter `localLog` defined as imported

6. Add a wrapInt parameter ( ) with name `localWrapInt` and scope **Imported**.

7. Model the following method:

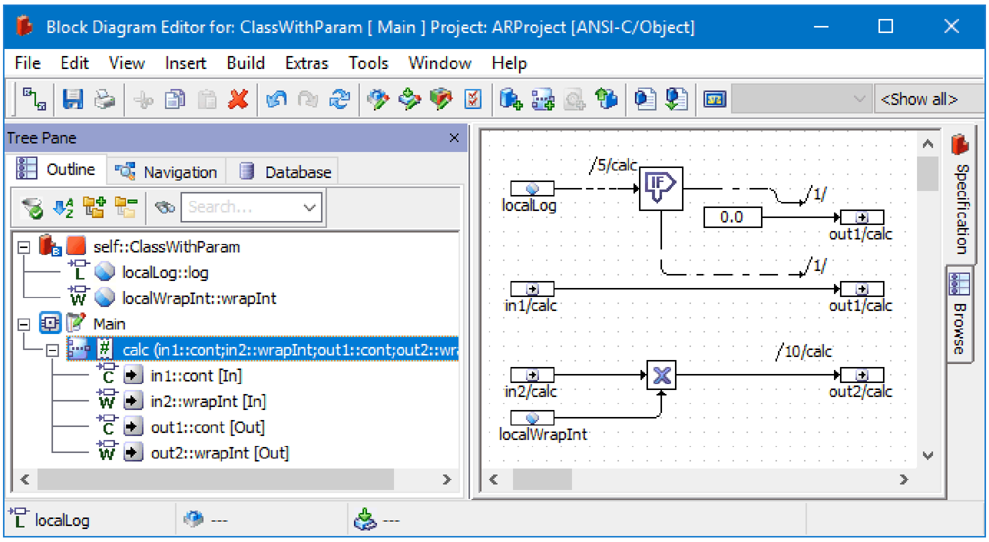


Figure 76: Block diagram of method `calc`

To map internal parameters of a function to AUTOSAR calibration parameters

1. Create a project as described on page 17.
2. Insert `Swc` into the new project as described on page 23.
3. Insert the calibration interface `CalInterface` created in section 5.4, *Calibration*, on page 77 into the software component `Swc`.
4. Insert the class `ClassWithParam` into the software component `Swc`.
5. Insert the cont variables `inValue1` and `outValue1`.
6. Insert the wrapInt variables `inValue2` and `outValue2`, both with type uint16 and default min/max.
7. Create a runnable `Runnable_Entity`.
8. If you are working in the *block diagram editor* for SWC, proceed as follows:
   i. Connect the variables as shown in Figure 77.
   ii. Provide a sequence to the method `calc` within `Runnable_Entity`.

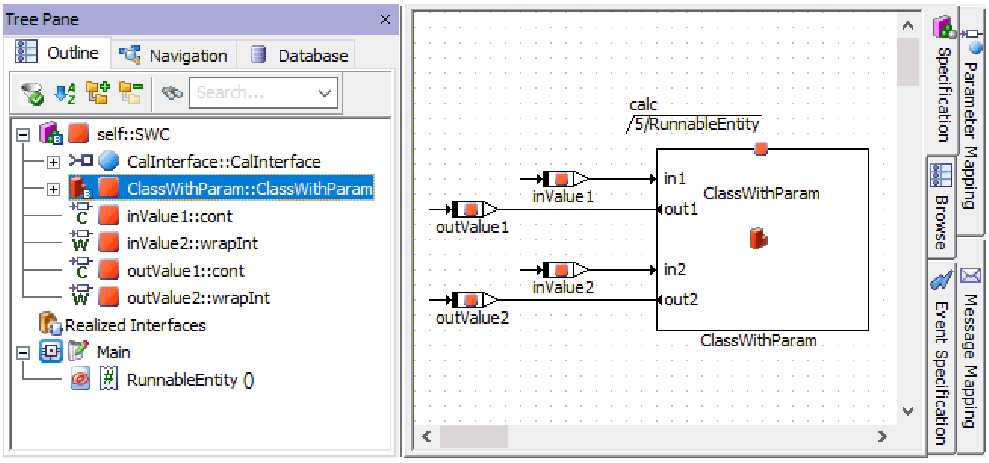


Figure 77: Accessing `ClassWithParam` in the software component (block diagram editor for SWC)

9. If you are working in the *ESDL editor* for SWC, proceed as follows:
   i. Enter the following code:

```
ClassWithParam.calc(inValue1, inValue2, outValue1,
outValue2);
```

10. In both SWC editors, continue as follows:
    i. Go to the "Parameter Mapping" tab.

    The left column of the table lists all imported parameters in modules and classes of the software component.

    The right column of the table contains a drop-down list for each imported parameter. Each list provides the calibration parameters in the software component matching, in type, the imported parameters.

    ii. For the parameter `localLog`, select the calibration parameter `calParam1`.

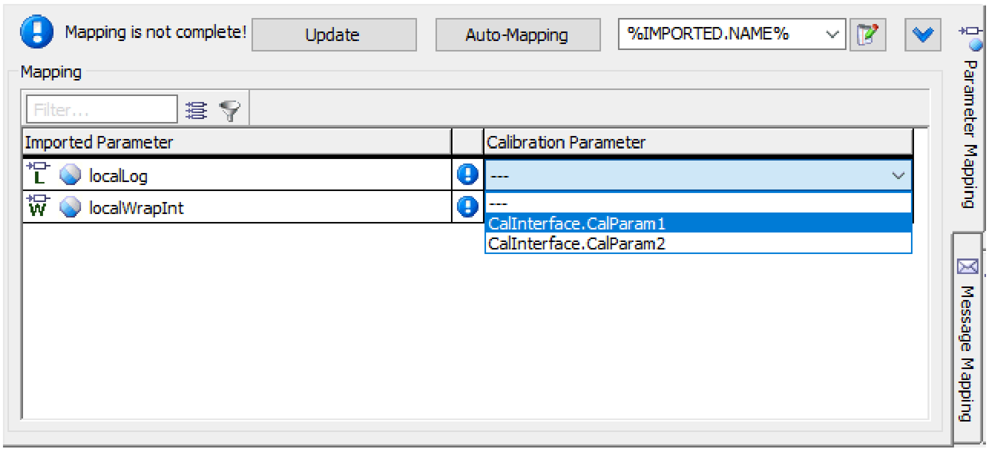


Figure 78: Mapping an imported parameter and a calibration parameter

    iii. For the parameter `localWrapInt`, select the calibration parameter `calParam3`.

    With that, parameter mapping is complete.

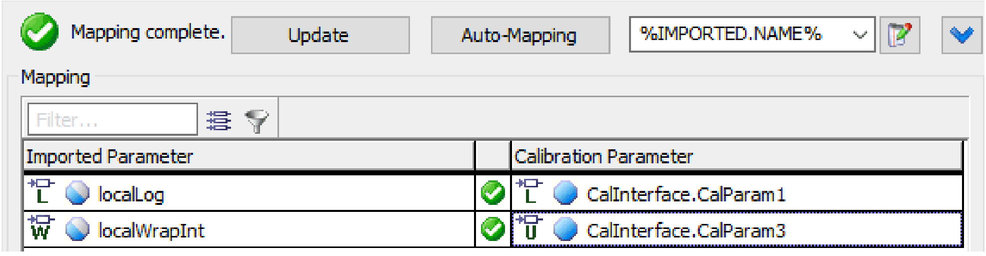


Figure 79: Completed parameter mapping

For the class `ClassWithParam`, ASCET generates the following C code:

```
FUNC(void, SWC_CODE) CLASSWITHPARAM_IMPL_calc (
   /* IN    */ VAR(sint32, AUTOMATIC)                     in1,
   /* IN    */ VAR(uint16, AUTOMATIC)                     in2,
   /* OUT   */ CONSTP2VAR(sint32, AUTOMATIC, RTE_APPL_DATA) out1,
   /* OUT   */ CONSTP2VAR(uint16, AUTOMATIC, RTE_APPL_DATA) out2
   )
{
   /* calc: sequence call #5  */
   if (Rte_Prm_CalInterface_CalParam1())
   {
      /* If-block: sequence call #5/Then #1  */
      (*out1) = 0;
   }
   else
   {
      /* If-block: sequence call #5/Else #1  */
      (*out1) = in1;
   } /* end if */
   /* calc: sequence call #10  */
   (*out2) = (uint16)(in2 * (uint32)Rte_Prm_CalInterface_CalParam3());
}
```

Listing 102: C code – class with mapped parameters

> **NOTE**
>
> If a calibration interface is edited while the software component is open, update the changes in the "Parameter Mapping" tab using the menu option **Mapping › Update**.

9.5.2 Accessing ASCET Messages

AUTOSAR does not know the concept of ASCET messages. If your SWC uses one or more modules that contain ASCET messages, all messages must be mapped to semantically equivalent AUTOSAR elements.

For this purpose, ASCET provides a special editor in the "Message Mapping" view of the software component editor.

In that editor, messages can be mapped to AUTOSAR elements according to the following rules:

– Messages can be mapped according to the following table:

Message	Internal mapping	External mapping
Scalar	to scalar interrunnable variables	to scalar elements of Sender-Receiver interfaces
	to scalar elements of complex interrunnables (records)	to scalar elements of complex elements (records) in Sender-Receiver interfaces
Composite (arrays, matrices)	to composite interrunnable variables	to composite elements (array, matrix) of SenderReceiver interfaces
	to composite elements of complex interrunnables (records)	to composite elements of complex elements (records) in SenderReceiver interfaces
Complex (records)	to complex interrunnable variables (records)	to complex elements (records) in SenderReceiver interfaces
	to complex elements of complex interrunnable variables (records)	to complex elements of complex elements (records) in Sender-Receiver interfaces

Table 7: Mapping possibilities for ASCET messages

– Scalar and composite elements of nested records (record A contains record B, which contains record C, etc.) used as SenderReceiver interface elements or interrunnable variables are available for message mapping.

> **NOTE**
>
> Recursively nested records (record A contains record B, which contains record A) are forbidden; using them leads to a code generation error (`EMake10`).

– Complex elements of nested records (record A contains record B, which contains record C, etc.) used as SenderReceiver interface elements or interrunnable variables are not available for message mapping.

– A scalar message must be mapped to an element of compatible type.

Message type	AUTOSAR element type
Continuous (cont)	cont / limitInt / wrapInt / sdisc / udisc
Limited Integer (limitInt)	limitInt / wrapInt
Wrap-Around Integer (wrapInt)	wrapInt
Signed Discrete (sdisc)	cont / limitInt / wrapInt / sdisc / udisc
Unsigned Discrete (udisc)	cont / limitInt / wrapInt / sdisc / udisc
Logic (log)	log
Enumeration (enum)	Enumeration of the same type

Table 8: Message types and compatible AUTOSAR types

If you map a scalar message to an element of compatible, but non-identical type, a warning (`WMdl635`) is issued during code generation.

If you map a scalar message to an element of incompatible type, an error (`MMdl635`) is issued during code generation.

– A composite message (array, matrix) must be mapped to an array or matrix of identical size, data type and implementation.

  Otherwise, the mapping is indicated as invalid, and an error (`MMdl635`) is issued during code generation.

– A complex message (record) must be mapped to a record of identical type and implementation.

  Otherwise, the mapping is indicated as invalid, and an error (`MMdl635`) is issued during code generation.

– Redundant data storage must not be activated for mapped messages.

  If it is, the following error (`MMdl37`) is issued during code generation:

  ```
  redundant data flag is set for <message>, but redundant data and mapped messages cannot be combined.
  ```

– A *pure send message* can only be mapped to an element of a sender-receiver interface used as Pport (external mapping), since the message value is not used within the SWC and thus provided to be used by another SWC.

  A pure send message is a send message that appears as a send message in all modules of the software component, i.e. it is not received by another module. Its Get method is not activated. Import/export of the pure send message is permitted.

– A *send message with activated Get method* can be mapped to one interrunnable variable (internal mapping) and/or one or more elements of SenderReceiver interfaces used as Pport (external mapping).

  If you map such a message to an element of an Rport, an error (`MMdl271`) is issued during code generation:

  ```
  Invalid external access mapping for element "<message>" in "<component>" - mapping to elements of provide port supported only
  ```

  If you map such a message only to an element of a Pport (i.e. no internal mapping), and use the Get method in the model, an error (`MMdl106`) is issued during code generation:

  ```
  Element "<element>" of provide port "<provide_port>" can not be read
  ```

– A *pure receive message* can only be mapped to an element of a sender-receiver interface used as Rport (external mapping), since the message value is not given within the SWC and must therefore be given by another SWC. Internal mapping is not provided for pure receive messages.

  A pure receive message is a receive message that is not used as send message within the SWC. Its Set method is not activated.

– A *receive message with activated Set method* can be mapped to one interrunnable variable (internal mapping) and/or one element of a sender-receiver interface used as Rport (external mapping).

If you map such a message to an element of a Pport, an error (`MMdl271`) is issued during code generation:

```
Invalid external access mapping for element "<mes-
sage>" in "<component>" - mapping to elements of re-
quire port supported only
```

If you map such a message only to an element of an Rport (i.e. no internal mapping), and use the Set method in the model, an error (`MMdl107`) is issued during code generation:

```
Element "<element>" of require port "<require_port>"
can not be written
```

– All other messages, i.e. SendReceive messages and messages specified as send message in one module and as receive message in another module, can be mapped to an interrunnable variable (internal mapping) or to an element of a sender-receiver interface used as Pport (external mapping).

  If you map a SendReceive message with activated Get and/or Set method to an element of an Rport, an error (`MMdl271`) is issued during code generation.

– Pure receive messages can have one external mapping. Pure send messages and other messages can have multiple external mappings.

– Pure send messages and pure receive messages cannot have an internal mapping. Other messages can have one internal mapping.

– Imported messages must have only one internal mapping. If you apply an external mapping, too, an error (`MMdl274`) is issued during code generation.

Internal mapping is indicated as complete if each mappable message is mapped. External mapping is indicated as complete if each mappable message is mapped once. However, you can still map messages that allow multiple mapping.

To create a module with messages

1. In the component manager, select **Insert > Module > Block Diagram** to create a module.
2. Name the module `ModuleWithMsg`.
3. Open `ModuleWithMsg` in the block diagram editor.
4. Use the **SendReceive Message** button to create a receive message.

   The dialog "Properties for Scalar Element: message" opens.
5. Name the message `SendRecMsg1` and change the basic type to `Signed Discrete`.
6. Click **OK** to close the properties editor.
7. Add a second SendReceive message with name `SendRecMsg2` and basic type `Signed Discrete`.
8. Add a send message ( ) with name `SendMsg` and basic type `Logic`.
9. Implement `SendRecMsg1` and `SendRecMsg2` as `sint8` (see Figure 12).
10. Implement `SendMsg` as `bool`.
11. Model the process as shown in Figure 80.

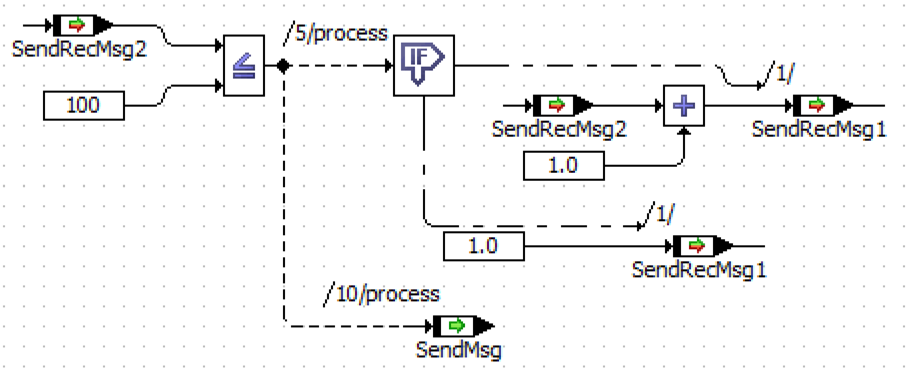


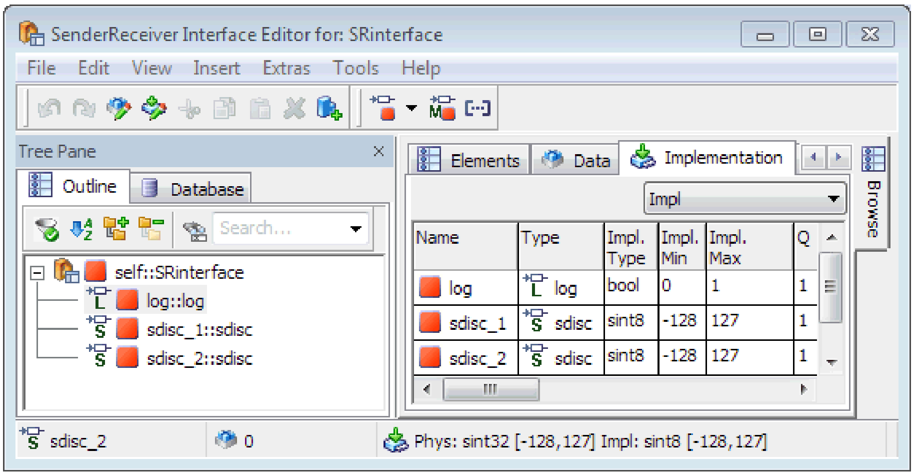
Figure 80: Block diagram of process `process` in `ModuleWithMsg`

To map ASCET messages to AUTOSAR elements

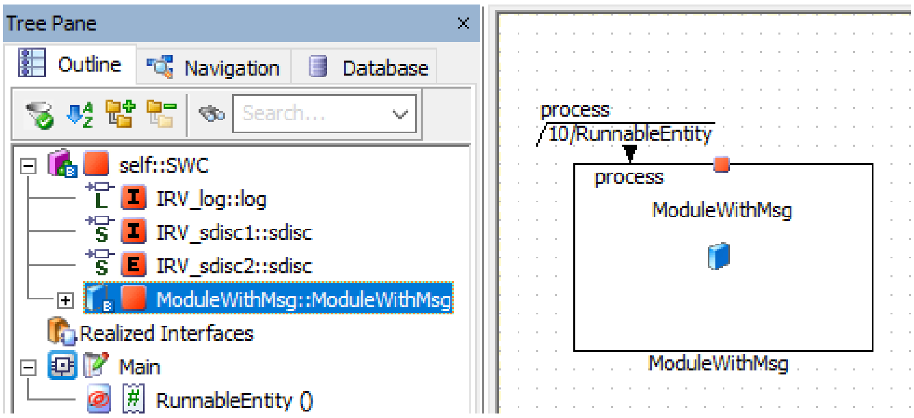
1. Create (cf. page 17) and set up (cf. page 18) a project `ARProject`.
2. Create a software component `Swc` (cf. page 23) with a runnable entity and three interrunnable variables (cf. page 130):

name	`IRV_sdisc1`	`IRV_sdisc2`	`IRV_log`
basic type	`Signed Discrete`		`Logic`
Impl. type	`sint8`		`bool`
Internal access	**implicit**	**explicit**	**implicit**

3. Create a sender-receiver interface `SRinterface` (cf. page 63) with two `sdisc` data elements, implemented as `sint8`, and one `log` element, implemented as `bool`.



4. Add the module `ModuleWithMsg` to `Swc`, as shown below.



5. Use `SRinterface` to create a sender port in the SCW (cf. page 86).
6. Add `Swc` to `ARProject`.
7. In the "Outline" tab of the project editor, double-click `Swc` to open the component in the project context.
8. In the software component editor, go to the "Message Mapping" tab and the "Internal Access" sub-tab.

   The left column of the table lists all messages that can be mapped to interrunnable variables.

   The right column of the table contains a drop-down list for each message. Each list provides the interrunnable variables that can be mapped to the message.
9. Map the messages to interrunnable variables as shown in Figure 81.

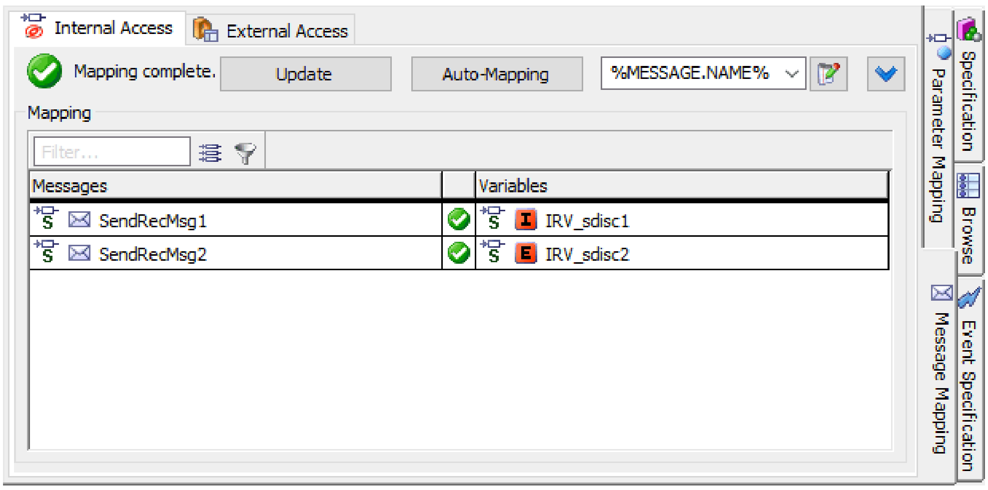


Figure 81: Mapping messages and interrunnable variables

10. Go to the "External Access" sub-tab.

    The left column lists all messages that can be mapped to data elements in sender or receiver ports.

    The right column contains a drop-down list for each message. Each list provides the data elements that can be mapped to the message.
11. Map the messages to data elements as shown in Figure 82.

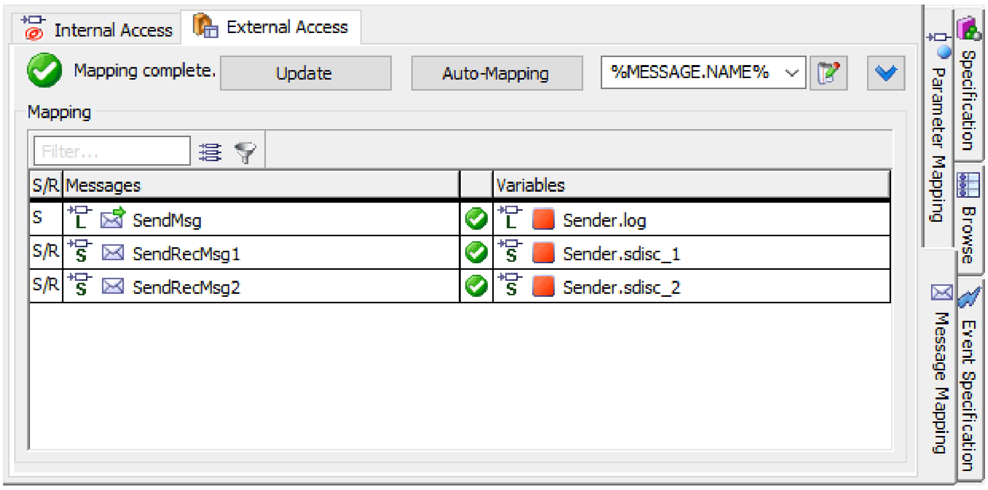


Figure 82: Mapping messages and data elements

With that, message mapping is complete.

For the module `ModuleWithMsg`, ASCET generates the following C code:

```
FUNC(void, SWC_CODE) MODULEWITHMSG_IMPL_process (void)
{
   /* temp. variables */
   VAR(sint8, AUTOMATIC) _t1sint8;
   VAR(sint8, AUTOMATIC) _t2sint8;

   /* process: sequence call #5  */
   if (Rte_IrvRead_RunnableEntity_IRV_sdisc2() <= 100)
   {
      /* If-block: sequence call #5/Then #1  */
      _t1sint8 = Rte_IrvRead_RunnableEntity_IRV_sdisc2();
      _t2sint8 = ((_t1sint8 <= 126) ? (_t1sint8 + 1) : 127);

      Rte_IrvIWrite_RunnableEntity_IRV_sdisc1(_t2sint8);

      Rte_IWrite_RunnableEntity_Sender_sdisc_1(_t2sint8);
   }
   else
   {
      /* If-block: sequence call #5/Else #1  */
      Rte_IrvIWrite_RunnableEntity_IRV_sdisc1(1);

      Rte_IWrite_RunnableEntity_Sender_sdisc_1(1);
   } /* end if */
   /* process: sequence call #10  */
   Rte_IWrite_RunnableEntity_Sender_log(
   Rte_IrvRead_RunnableEntity_IRV_sdisc2() <= 100);
}
```

Listing 103: C code – module with mapped messages

9.5.3 Accessing Non-Volatile Variables

In case an ASCET class, state machine, or module containing non-volatile variables (*NV variables*) is used within an AUTOSAR software component, all NV variables (except local NV variables in classes) can be mapped to semantically equivalent AUTOSAR elements. Unlike messages, however, you do not have to map NV variables, you can access them in the normal way.

For this purpose, ASCET V6.4.6 introduced a special editor in the "NV-Data Mapping" view of the software component editor. In that editor, NV variables can be mapped to AUTOSAR elements according to the following rules:

– NV variables of all scopes – exported, imported, local – can be mapped to NVData interface elements, with one exception:

  Local NV variables in classes cannot be mapped to NVData interface elements.

– You must not map NV variables that are written in your model to NVData interfaces used as Rports.

  If you do, an error `MMdl2761` is issued during code generation.

> **NOTE**
>
> You cannot create NVData interface Pports in ASCET, but you can import ARXML files that contain NVData interface Pports.

– One NV variable can be mapped at most to one NVData interface Rport and one NVData interface Pport.

– NV variables can be mapped as shown in the following table.

NV variable	Mapping
scalar	to scalar elements of NVData interfaces
	to scalar elements of complex elements (records) in NVData interfaces
composite (arrays, matrices)	to composite elements (arrays) of NVData interfaces
	to composite elements of complex elements (records) in NVData interfaces
complex (records)	to complex elements (records) in NVData interfaces
	to complex elements of complex elements in NVData interfaces

Table 9: Mapping possibilities for ASCET NV variables

– Scalar NV variables must be mapped to scalar elements of compatible type.

NV variable type	AUTOSAR element type
Continuous (cont)	cont / limitInt / wrapInt / sdisc / udisc
Limited Integer (limitInt)	limitInt / wrapInt
Wrap-Around Integer (wrapInt)	wrapInt
Signed Discrete (sdisc)	cont / limitInt / wrapInt / sdisc / udisc
Unsigned Discrete (udisc)	cont / limitInt / wrapInt / sdisc / udisc
Logic (log)	log
Enumeration (enum)	Enumeration of the same type

Table 10: NV variable types and compatible AUTOSAR types

If you map a scalar NV variable to an element of compatible, but non-identical type, a warning `WMdl635` is issued during code generation.

If you map a scalar NV variable to an element of incompatible type, an error `MMdl635` is issued during code generation.

- A composite NV variable (array, matrix) must be mapped to an array/matrix of identical size, data type and implementation.

  Otherwise, the mapping is indicated as invalid, and an error `MMdl635` is issued during code generation.

- A complex NV variable must be mapped to a record of identical type and implementation.

  Otherwise, the mapping is indicated as invalid, and an error `MMdl635` is issued during code generation.

- Redundant data storage must not be activated for NV variables.

  If it is, an error `MMdl37` is issued during code generation.

> **NOTE**
>
> SWC that use NV data mapping can only be exported to ASCET V6.4.6 or newer.

NV-Data mapping is indicated as complete if each mappable NV variable is mapped once. However, you can still map NV variables that allow multiple mapping.

To create a module with NV variables

1. In the component manager, select **Insert > Module > Block Diagram** to create a module.
2. Name the module `Module_NVvar`.
3. Open `Module_NVvar` in the block diagram editor.
4. Use the **Continuous Variable** button to create a variable.

   The properties editor for the new variable opens.
5. Name the variable `NV_cont_e`, change the basic type to `Continuous`, set the scope to **Exported**, and activate the **Non-Volatile** option.
6. Click **OK** to close the properties editor.
7. Add the following NV variables:

Name	Type	Scope	Basic type	Min/max
`NV_cont_i`	scalar	imported	continuous	
`NV_cont_l`	scalar	exported	continuous	
`NV_array`	Array [4]	local or exported	Limited integer	-32768 / 32767
`NV_wrapInt`	scalar	local or exported	Wrap-around integer	0 / 255

8. Add the following scalar volatile output variables:

Name	Basic type	Min/max
Out_cont	continuous	
Out_log	logical	
Out_limitInt	Limited integer	-32768 / 32767

9. Add a parameter out_x with basic type Wrap-around integer, type uint8, and min/max of 0/3.
10. Implement the elements as follows:

Name	Impl. Type	Impl. Min	Impl. Max
NV_cont_e	sint16	-32768	32767
NV_cont_i	*not applicable*		
NV_cont_l	sint16	-32768	32767
NV_array	sint16	-32768	32767
out_x	uint8	0	3
NV_wrapInt	uint8	0	255
Out_cont	sint32	-2147483648	2147483647
Out_log	bool	0	1
Out_limitInt	sint32	-2147483648	2147483647

11. Model the following process:

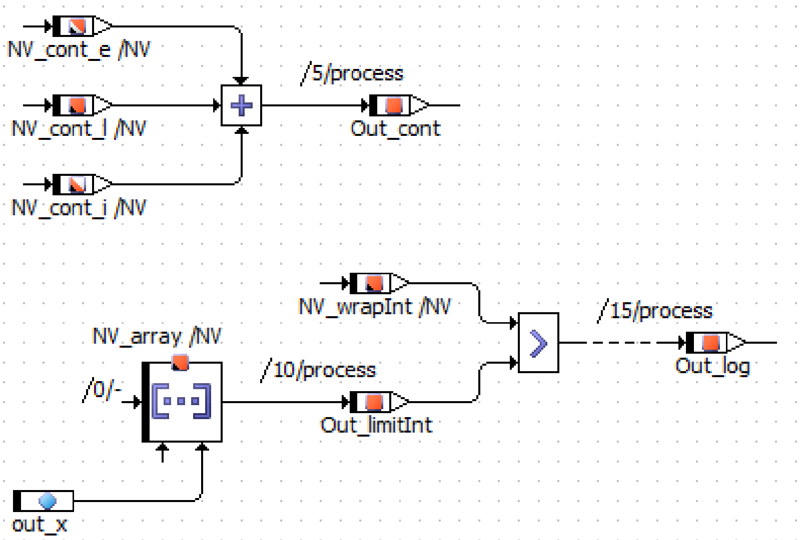
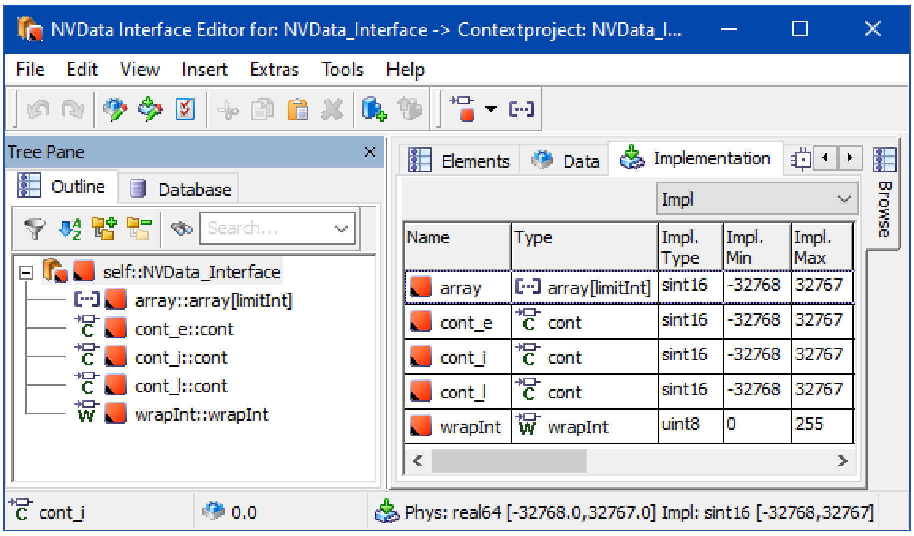


Figure 83: Block diagram of process process in Module_NVvar

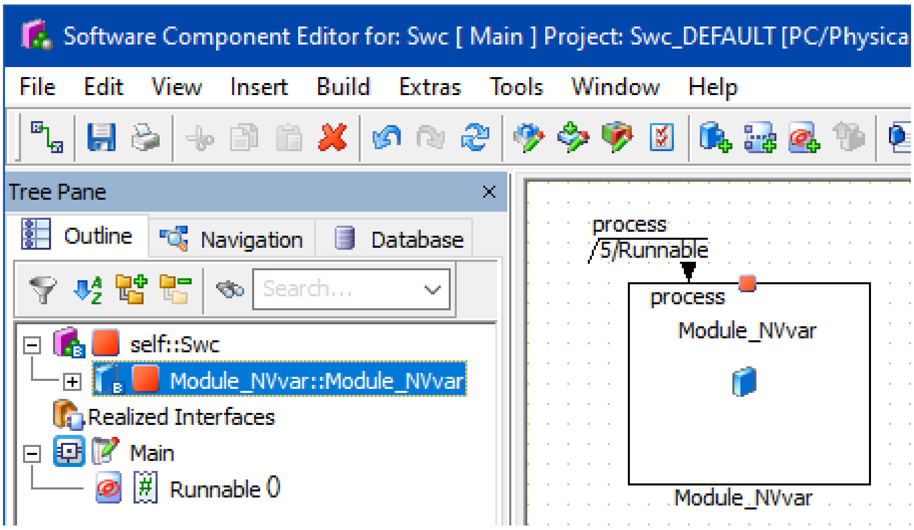
To map NV variables to NVData interface elements

1. Create (cf. page 17) and set up (cf. page 18) a project ARProject.
2. Create a software component Swc (cf. page 23) with a runnable entity.
3. Create an NVData interface NVData_Interface with three scalar cont data elements (implemented as sint16), one scalar wrapInt element

(implemented as `uint8`), and one array element `array` (`limitInt`, implemented as `sint16`).



4. Add the module `Module_NVvar` to `Swc`.



5. Use `NVData_Interface` to create an NVData port in the SCW (cf. page 103).
6. Add `Swc` to `ARProject`.
7. In the "Outline" tab of the project editor, double-click `Swc` to open the component in the project context.
8. In the software component editor, go to the "NV-Data Mapping" tab.

   The left column of the table lists all NV variables that can be mapped to NVData elements.

   The right column of the table contains a drop-down list for each NV variables. Each list provides the NVData elements that can be mapped to the NV variable.
9. Map the NV variables to NVData elements as shown in Figure 84.

   With that, NV Data mapping is complete.

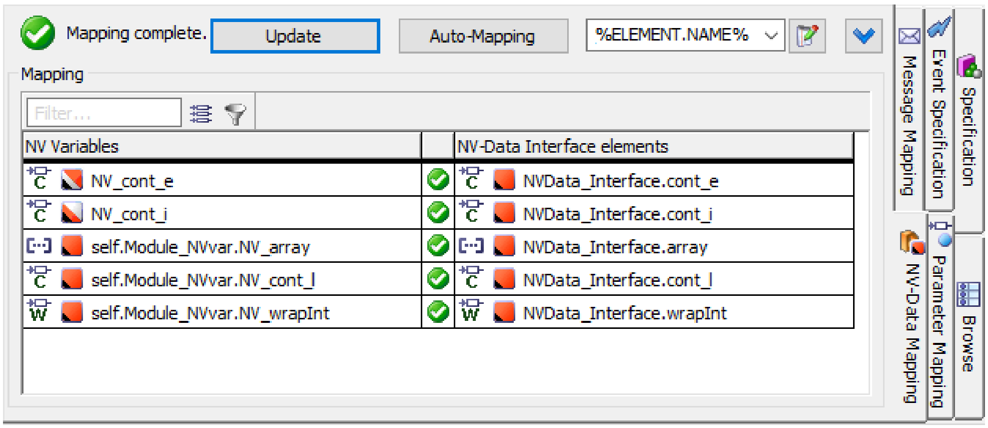


Figure 84: Mapping NV variables and NVData elements

For the module `Module_NVvar`, ASCET generates the following C code:

```
FUNC(void, Swc_CODE) MODULE_NVVAR_IMPL_process (void)
{
   /* process: sequence call #5  */
   _Out_cont
      = (sint32)Rte_IRead_Runnable_NVData_Interface_cont_l()
      + Rte_IRead_Runnable_NVData_Interface_cont_i()
      + Rte_IRead_Runnable_NVData_Interface_cont_e();
   /* process: sequence call #10  */
   _Out_limitInt
      = Rte_IRead_Runnable_NVData_Interface_array()[_out_x];
   /* process: sequence call #15  */
   _Out_log
      = (sint32)Rte_IRead_Runnable_NVData_Interface_wrapInt()
      > _Out_limitInt;
}
```

Listing 104: C code – module with mapped NV variables

9.5.4 Automatic Mapping

Mapping each individual message, NV variable, or imported parameter can be time-consuming if you have a lot of these elements. To simplify the task, the software component editor provides an auto-mapping function.

By default, auto-mapping maps messages, NV variables, or imported parameters and AUTOSAR elements with identical names. To make auto-mapping more effective, you can define patterns that are used to map messages, NV variables, or imported parameters and AUTOSAR elements with different names. With these patterns, you can, for example, take into account prefixes and/or postfixes.

Default patterns are defined in the ASCET options, "Appearance\Editors\Software Component" node. These patterns are available in the combo boxes of the "Parameter Mapping" view, the "Internal Access" and "External Access" tabs of the "Message Mapping" view, and the “NV-Data Mapping” view. You can add your own patterns, either via the ASCET options window or via the combo boxes.

Auto-mapping automatically maps unconnected messages, NV variables, or imported parameters in the software component to AUTOSAR elements with a matching name and type (i.e. scalar, array, or record) according to the following heuristic:

- "Parameter Mapping" tab
  - An imported parameter is mapped to a Calibration interface element with a matching name and type.
- "Internal Access" tab
  - A message with several matching counterparts is mapped to the first counterpart. Other counterparts are ignored.
- "External Access" tab
  - A message labeled as S or S/R in the "Mapping" field with several matching counterparts is mapped to each counterpart.
  - A message labeled as `R` in the "Mapping" field is mapped to the first matching counterpart. Other counterparts are ignored.
- “NV-Data Maping” tab
  - An NV variable is mapped to an element of an NVData interface with a matching name and type.

NOTE

There is no guarantee that a message/imported parameter and an AUTOSAR element with matching names represent the same concept.

For example, a message named `Speed` that represents speed in km/h is not the same as an AUTOSAR element named `Speed` that represents speed in miles/h.

You must therefore verify that any auto-mappings represent valid connections.

Matching names are derived from the currently selected pattern.

- "Parameter Mapping" tab

  `%IMPORTED.NAME%` is the template for names of elements in the "Imported Parameter" column, `%CALIBRATION.NAME%` is the template for names of elements in the "Calibration Parameter" column.
- "Internal Access" tab and "External Access" tab

  `%MESSAGE.NAME%` is the template for names of elements in the respective "Messages" column, `%VARIABLE.NAME%` is the template for names of elements in the respective "Variables" column.
- “NV-Data Mapping” tab

  `%NVVARIABLE.NAME%` is the template for names of NV variables in the “NV Variables” column, `%ELEMENT.NAME%` is the template for names of elements in the “NV-Data Interface elements” column.

If you add a prefix or postfix to one of these templates, auto-mapping works as shown in the following table.

	Pattern	Auto-mapping result
"Parameter Mapping" tab	`prefix_%IMPORTED.NAME%`	An imported parameter `name` and a Calibration parameter `prefix_name` of the same type[15] are mapped.
	`%IMPORTED.NAME%_postfix`	An imported parameter `name` and a Calibration parameter `name_postfix` of the same type[15] are mapped.
	`prefix_%CALIBRATION.NAME%`	An imported parameter `prefix_name` and a Calibration parameter `name` of the same type[15] are mapped.
	`%CALIBRATION.NAME%_postfix`	An imported parameter `name_postfix` and a Calibration parameter `name` of the same type[15] are mapped.
	Names of classes, modules, Calibration interfaces, or records are not considered.	
"Internal Access" tab	`prefix_%MESSAGE.NAME%`	A message `name` and an interrunnable variable `prefix_name` of the same type[15] are mapped.
	`%MESSAGE.NAME%_postfix`	A message `name` and an interrunnable variable `name_postfix` of the same type[15] are mapped.
	`prefix_%VARIABLE.NAME%`	A message `prefix_name` and an interrunnable variable `name` of the same type[15] are mapped.
	`%VARIABLE.NAME%_postfix`	A message `name_postfix` and an interrunnable variable `name` of the same type[15] are mapped.
	Names of modules or records are not considered.	
"External Access" tab	`prefix_%MESSAGE.NAME%`	A message `name` and an interface element `prefix_name` of the same type[15] are mapped.
	`%MESSAGE.NAME%_postfix`	A message `name` and an interface element `name_postfix` of the same type[15] are mapped.
	`prefix_%VARIABLE.NAME%`	A message `prefix_name` and an interface element `name` of the same type[15] are mapped.
	`%VARIABLE.NAME%_postfix`	A message `name_postfix` and an interface element `name` of the same type[15] are mapped.
	Names of modules, SenderReceiver/NVData interfaces, or records are not considered.	

[15] i.e. scalar, array, matrix, or record

"NV-Data Mapping" tab	prefix_ %NVVARIABLE.NAME%	An NV variable name and an interface element prefix_name of the same type[16] are mapped.
	%NVVARIABLE.NAME% _postfix	An NV variable name and an interface element name_postfix of the same type[16] are mapped.
	prefix_ %ELEMENT.NAME%	An NV variable prefix_name and an interface element name of the same type[16] are mapped.
	%ELEMENT.NAME% _postfix	An NV variable name_postfix and an interface element name of the same type[16] are mapped.
	ⓘ Names of classes, modules, NVData interfaces, or records are not considered.	

Table 11: Results of auto-mapping with prefix and postfix patterns

Keep in mind the following rules when you define a template:

– You can add a prefix, or a postfix, or both.

– Prefixes and postfixes are case-sensitive; a pattern PREFIX_*<name>* will not match a message or AUTOSAR element prefix_*<name>*.

– You cannot use %IMPORTED.NAME% and %CALIBRATION.NAME% in the same pattern for parameter mapping.

– You cannot use %MESSAGE.NAME% and %VARIABLE.NAME% in the same pattern for message mapping.

– You cannot use %NVVARIABLE.NAME% and %ELEMENT.NAME% in the same pattern for NV data mapping.

Auto-mapping is accessed via the **Mapping** menu or - in the respective views - the **Auto-Mapping** context menu option or the **Auto-Mapping** button. See also the ASCET online help.

9.5.5 Mapping Conversion

Prior to ASCET V6.4.6, messages used in AUTOSAR SWCs could be mapped to both SenderReceiver interface elements and NVData interface elements. Since ASCET V6.4.6, messages can only be mapped to SenderReceiver interface elements. Existing mappings of messages to NVData interface elements must be converted to NV data mappings.

When you open a database or workspace that contains mappings of messages to NVData interface elements, these mappings become invalid; see the example in Figure 85. Mappings of messages to SenderReceiver interface elements or interrunnable variables remain valid.

[16] i.e. scalar, array, matrix, or record

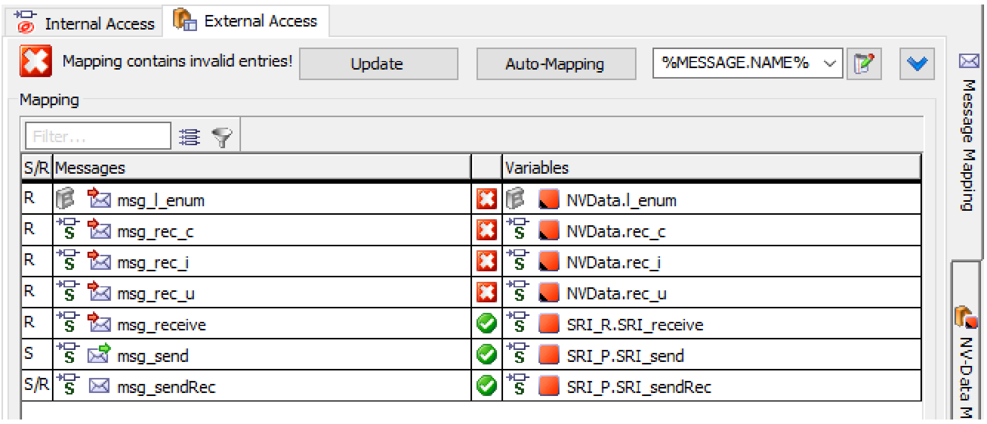


Figure 85: “Message Mapping” tab with invalid mappings of messages to NVData elements

You must convert the invalid message mappings to NV-Data mappings. To make this easier, ASCET offers the **Convert to NV-Data Mapping** option in the **Mapping** menu or the context menu in the lower table of the “External Access” tab.

Details on mapping conversion are given in the ASCET online help.

9.6 Concurrency Control with Exclusive Areas

Where a component has multiple runnable entities that require concurrent write access to the same prototype state, then the `Rte_Enter` and `Rte_Exit` API calls must be used to ensure that data consistency is maintained.

A component includes multiple runnable entities each of which can be active simultaneously. The potential exists for concurrent access to private global data (e.g. elements in the data memory sections) and/or non-reentrant functions.

Operating system concurrency control mechanisms are hidden from components. However, the RTE API implements explicit access to exclusive areas by exposing an appropriate OS mechanism to components:

- `Rte_Enter_`*<exclusive area name>* enters an exclusive area.
- `Rte_Exit_`*<exclusive area name>* exits an exclusive area.

Where components declare exclusive areas, the generated RTE API for the component includes these API calls to allow you to control concurrent access to shared data.

9.6.1 Sequences of a Runnable Assigned to an Exclusive Area

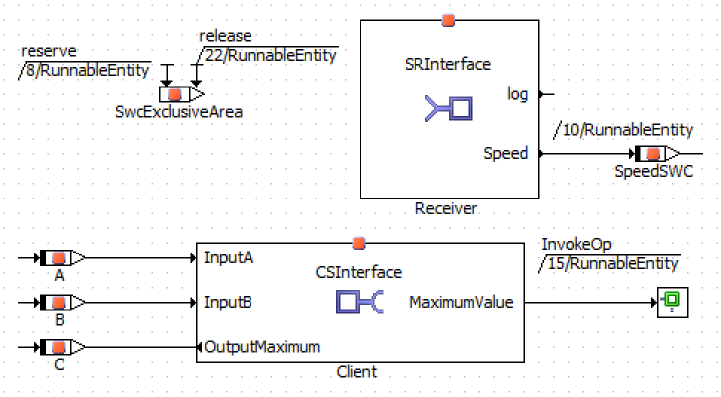
A component can use the `Rte_Enter` and `Rte_Exit` API calls for any exclusive area ID you define at configuration time.

For example, for the exclusive area `SwcExclusiveArea` of section 7.10, *Exclusive Areas*, on page 153, the following C calls are used:

```
Rte_Enter_SwcExclusiveArea();
/* Code protected from concurrent execution */
...
Rte_Exit_SwcExclusiveArea();
```

Listing 105: C code – enter/exit exclusive area

For the example of section 7.10.2 on page 154,



(block diagram editor for SWC)

ASCET generates the following C code:

```
#define _A (SWC_RAM.A)
...
#define _B (SWC_RAM.B)
#define _C_REF_ (&(SWC_RAM.C))
...
#define _SpeedSWC (SWC_RAM.SpeedSWC)

...

FUNC(void, SWC_CODE) SWC_Impl_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #8  */
   Rte_Enter_SwcExclusiveArea();
   /* RunnableEntity: sequence call #10  */
   _SpeedSWC = Rte_IRead_RunnableEntity_Receiver_Speed();
   /* RunnableEntity: sequence call #15  */
   _ASCET_RteStatus = Rte_Call_Client_MaximumValue(_A,_B,_C_REF_);
   /* RunnableEntity: sequence call #22  */
   Rte_Exit_SwcExclusiveArea();
}
```

Listing 106: C code – exclusive area example

NOTE

The scope of an exclusive area is the software component *prototype* and not the software component type or system wide. Therefore, exclusive areas only provide concurrency control within one software component.

Wider scope can be achieved using an AUTOSAR component to broker access to shared data.

9.7 Description of Internal Data Structures

In AUTOSAR, it is possible to add the description of internal data structures to the ARXML code for two main purposes:

A. Generation of an A2L file describing the internal data for measurement and calibration

   The data structures are still generated by ASCET, but described in the ARXML code using compu methods, data constraints and implementation types, so that the description in the A2L is correct and also the addresses of the data can be retrieved from the executable file.

B. Allow multiple instances of a software component

   The data structures are described completely in the ARXML and are generated by the RTE generator. Special Rte macros are used in the generated code to access the data. Also an A2L file can be generated. However, this variant comes with some restrictions.

These variants can be configured in the "ARXML Configuration Settings" window, "Miscellaneous" node.

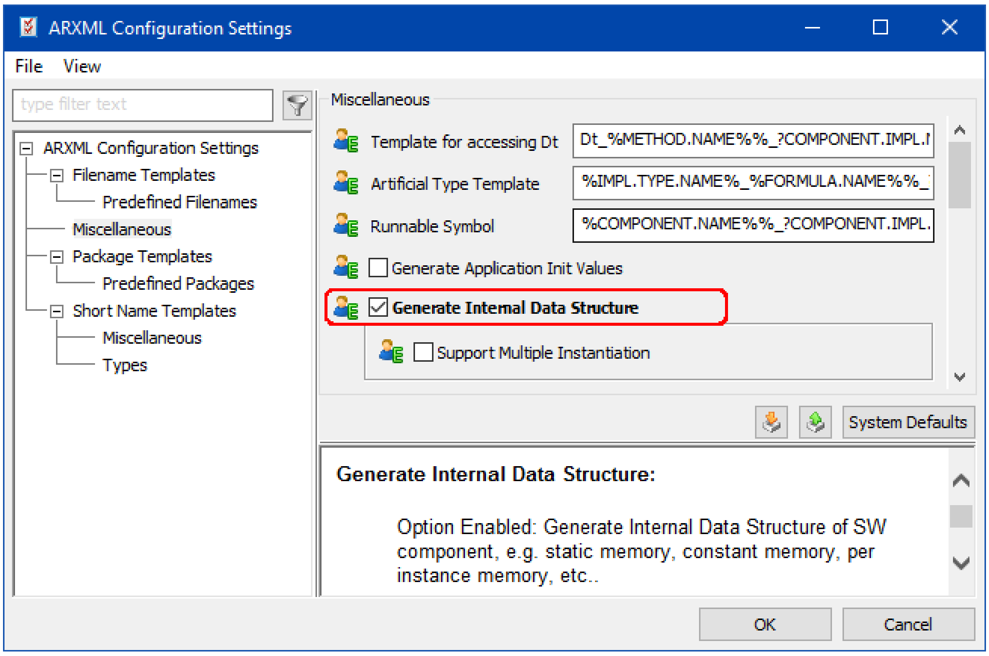


Figure 86: Settings for generation of internal data structure in the "ARXML Configuration Settings" window, "Miscellaneous" node

9.7.1 Measurement and Calibration

If the generation of internal data structures is activated without support of multi-instance software components, the elements in the generated data structures are also described in the ARXML code, with the following exceptions:

- elements with both read and write calibration access disabled (unless they are distributions)
- elements of model type log with implementation type bit

- fixed characteristic lines/maps
- arrays with variant size, where the system constant is not generated by the RTE
- reference elements
- group characteristic lines/maps of kind Variable

The internal behavior contains constant memory with parameter data prototypes, and static memory with variable data prototypes for all non-empty memory structures of the software component, modules, single-instance classes and exported elements. These prototypes may be typed by a generated record application type to reproduce the nesting of structs as generated by ASCET.

Since the calibration access is specified for all data of a data prototype, it may not be possible to reproduce the settings in the model for the ARXML description. If at least one element with read-only calibration access is in the data prototype, the complete prototype is configured as read-only, and a warning `WIle57` is reported for the affected read-write elements.

9.7.2 Multi-Instance Software Components

If the generation of multi-instance software components is activated, all elements are described in the data structures, with the following exceptions:

- elements of model type log with implementation type bit
- fixed characteristic lines/maps
- arrays with variant size, where the system constant is not generated by the RTE
- reference elements
- group characteristic lines/maps of kind Variable
- distributions with a user-defined type for the distribution search results

To keep the data structures generated by the RTE generator as similar as possible to the data structure generated by ASCET, also a substruct pointer is added. Since this substruct pointer cannot be statically initialized, it is initialized dynamically in the init runnable. Therefore, the following restrictions apply if a multi-instance class is used in the model:

- The main structure of the class must be allocated to a writable memory, i.e. the memory location must be assigned to category Variable in the `memorySections*.xml` file.

  If this is not the case, an error (`MMdl3462`) is reported:

  `The memory class of the element` *`<element>`* `is` *`<memClass>`*`, but it has to be a variable memory class (due to dynamic initialization of substruct pointers for multiple instance SWCs.`

- The software component must include exactly on runnable with an init event.

  If this is not the case, an error (`MMdl3461`) is reported:

```
There is no init runnable defined, which is not allowed
for multiple instance SWCs due to dynamic initialization
of substruct pointers.
```

The internal behavior contains per-instance-parameter memory with parameter data prototype and ar-types-per-instance-memory with variable data prototype for all non-empty memory structures of the software component, modules, single-instance classes and exported elements. These prototypes may be typed by a generated record application type to reproduce the nesting of structs as generated by ASCET. These prototypes also contain the initialization.

In the generated C code, all functions have an additional argument named `rteInstance`. This argument is required to invoke Rte macros. All accesses to data is generated using the `Rte_CData` macros for parameters and `Rte_Pim` macros for variables.

```
FUNC(void, SWC_CODE) M_IMPL_process (/* IN   */ VAR(Rte_SelfType_SWC,
AUTOMATIC) rteInstance)
{
    /* process: sequence call #15  */
    (Rte_Pim_M(rteInstance)->v)
       = AddLimit_S16S16_S16(Rte_IRead_runnable_SRIin_in(rteInstance),
        (Rte_CData_M_CAL_MEM(rteInstance)->p));
    /* process: sequence call #15  */
    Rte_IWrite_runnable_SRIout_out(rteInstance,(Rte_Pim_M(rteInstance
    )->v));
}
```

Listing 107: C code – example for accessing items in a multi-instance SWC

10 Contact Information

Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:
www.etas.com/hotlines.

ETAS offers trainings for its products:
www.etas.com/academy

ETAS Headquarters

ETAS GmbH		
Borsigstraße 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	Internet:	www.etas.com

Glossary

ASCET

Development tool for control unit software

ASCET-MD

ASCET Modeling and Design

ASCET-SE

ASCET Software Engineering

AUTOSAR

Automotive **O**pen **S**ystem **Ar**chitecture; see https://www.autosar.org/

AUTOSAR R4.*

All supported AUTOSAR versions with major version number 4, i.e. R4.0.2, R4.0.3, R4.2.2, R4.3.0, R4.3.1.

ARXML

E**X**tensive **M**arkup **L**anguage (XML) used to describe AUTOSAR configurations.

BSW

Basic **s**oft**w**are; provides communications, I/O, and other functionality that all software components are likely to require.

CPU

Central **p**rocessing **u**nit

ECU

Embedded **C**ontrol **U**nit

ESDL

Embedded **s**oftware **d**evelopment **l**anguage developed by ETAS

NV variable

Non-volatile variable

OS

Operating **s**ystem

OSEK

Working group "open systems for electronics in automobiles" (German: Arbeitskreis **O**ffene **S**ysteme für die **E**lektronik im **K**raftfahrzeug)

Pport

Provided port; used by a software component to provide data or services to other software components.

RE

Runnable **e**ntity; a piece of code in an SWC that is triggered by the RTE at runtime. It corresponds largely to the processes known in ASCET.

Rport

Required port; used by a software component to require data or services from other software components.

RTA-OS

ETAS real-time operating system for deeply embedded ECUs with the highest safety level (ISO 26262 ASIL-D). It supports the latest versions of the relevant AUTOSAR, OSEK*/VDX, ISO 26262, and MISRA C standards.

RTA-RTE

AUTOSAR runtime environment by ETAS

RTE

AUTOSAR **r**un**t**ime **e**nvironment; provides the interface between software components, basic software, and operating systems.

SWC

AUTOSAR **s**oft**w**are **c**omponent; the smallest non-dividable software unit in AUTOSAR.

UUID

Universally **U**nique **Id**entifier

VFB

Virtual **F**unction **B**us

Figures

Code Listings

Tables

Index

A

Q

R

S

T

U

V