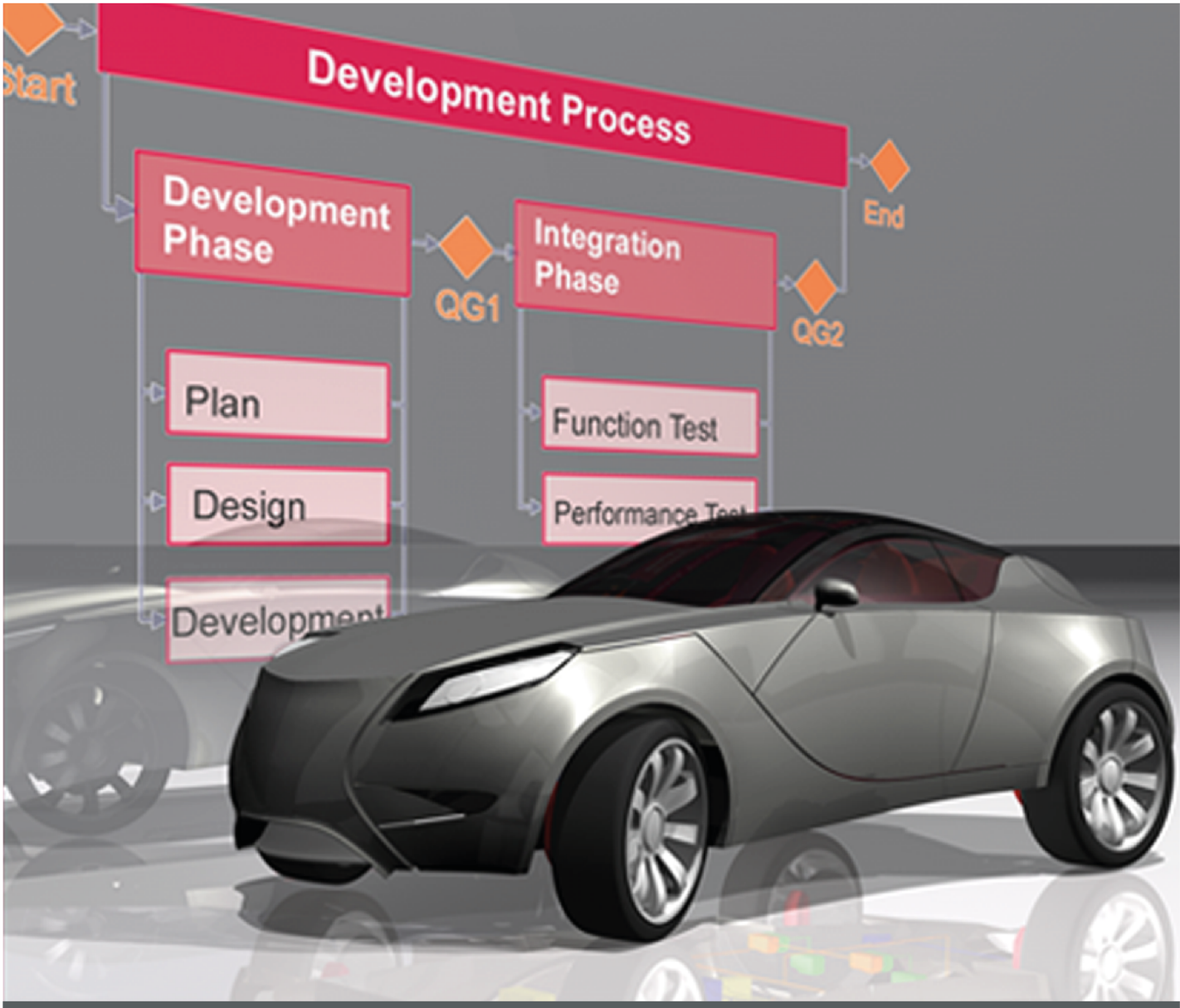
VECTOR



Startup with GM SLP2

Version 4.0.0 for MICROSAR 4 Release 16

User Manual - A Step by Step Introduction

Content

1 About This Manual

1.1 History Information

Cross Reference

For detailed information about what's new, what's changed in current version, refer to section What's New, What's Changed? on page 203.

1.2 Finding Information Quickly

The user manual provides the following access help:

- You can see in the footer to which version the user manual refers,
- At the end of the user manual you will find an index to find information quickly,
- Also at the end of the user manual you will find a glossary of the used technical terms

1.3 Conventions

In the following two charts you will find the conventions used in the user manual regarding utilized spellings and symbols.

Style	Style Utilization
bold	Blocks, surface elements, window- and dialog names of the software. Accentuation of warnings and advices. **[OK]** Push buttons in brackets **File|Save** Notation for menus and menu entries
MICROSAR	Legally protected proper names and side notes.
`Source Code`	File name and source code.
Hyperlink	Hyperlinks and references.
<CTRL>+<S>	Notation for shortcuts.

Symbol	Symbol Utilization
	Here you can obtain supplemental information.
	This symbol calls your attention to warnings.
	Here you can find additional information.
	Here is an example that has been prepared for you.
	Instructions on editing files are found at these points.
	This symbol warns you not to edit the specified file.
	This icon indicates multimedia files like e.g. video clips.
	This icon indicates an introduction into a specific topic.
	This icon indicates text areas containing basic knowledge.
	This icon indicates text areas containing expert knowledge.
	This icon indicates that something has changed.

1.4 Certification

Vector Informatik GmbH has ISO 9001:2008 certification. The ISO standard is a globally recognized standard.

1.5 Warranty

We reserve the right to modify the contents of the documentation or the software without notice. Vector disclaims all liabilities for the completeness or correctness of the contents and for damages

which may result from the use of this documentation.

1.6 Support

Germany And all other countries, not listed here.	
☎+ 49 711 80670 3900 ✉ support@de.vector.com	
BR Brazil, Argentina, Bolivia, Chile, Ecuador, Colombia, Paraguay, Peru, Uruguay, Venezuela	**CN** China
☎ +55 11 5180 2350 ✉ support@br.vector.com	☎ +86 21 6432 5353 ✉ support@cn.vector.com
FR France	**IN** India
☎ +33 1 42 31 40 10 ✉ support@fr.vector.com	☎ +91 20 6673 6673 ✉ support@in.vector.com
IT Italia	**JP** Japan
☎ +39 2 678171 10 ✉ support@de.vector.com	☎ TOKYO +81 3 5769 6972 ☎ NAGOYA +81 52 238 5014 ✉ support@jp.vector.com
KR South Korea	**Scandinavia** Sweden, Denmark, Finland, Iceland, Norway
☎ +82 2 807 0600 ✉ support@kr.vector.com	☎ +46 31 764 76 00 ✉ support@se.vector.com
UK United Kingdom & Ireland	**North America** USA, Canada, Mexico
☎ +44 121 7887 902 ✉ support@uk.vector.com	☎ +1 248 449 9290, Option 2 ✉ support@vectorcantech.zendesk.com

1.7 Registered Trademarks

All brand names in this documentation are either registered or non registered trademarks of their respective owners.

1.8 Errata Sheet of Hardware Manufacturers

Caution! Vector only delivers software!

Your hardware manufacturer will provide you with the necessary errata sheets concerning your used hardware. In case of errata dealing with CAN, please provide us with the relevant erratas and we will examine whether this hardware problem is already known to us or whether to get a possible workaround.

Note

Because of many NDAs with different hardware manufacturers or because we are not informed about them, we are not able to provide you with information concerning hardware errata of the hardware manufacturers.

1.9 Example Code

Caution!

All application code in any of the Vector User Manuals is for training purposes only. They are slightly tested and designed only to understand the basic idea of using a certain component or set of components. Vector gives consent that you use it as a basis for developing your own code and modify it (Vector waives its copyright to forbid you the use), however, with regard to the fact that the code is designed for training purposes only and not for the use by you, usability of the code is not warranted and Vector's liability shall be expressly excluded in cases of normal negligence, to the extent admissible by law or statute. Of course you have to test the software developed on the basis of the code with diligent care before using the software.

1.10 What Do You Learn from This Manual

Use this document to

- Set up your project in just a few steps
- Learn how to use the Vector tools
- Learn how to work with the AUTOSAR means like software components, runnables, etc.
- Work with the Vector AUTOSAR Solution on a basic level
- Get a feeling of what you have to do when you start working for your actual project

The document is split in two parts, a **STEPbySTEP** description and an **Expert Knowledge (Concepts)** section.

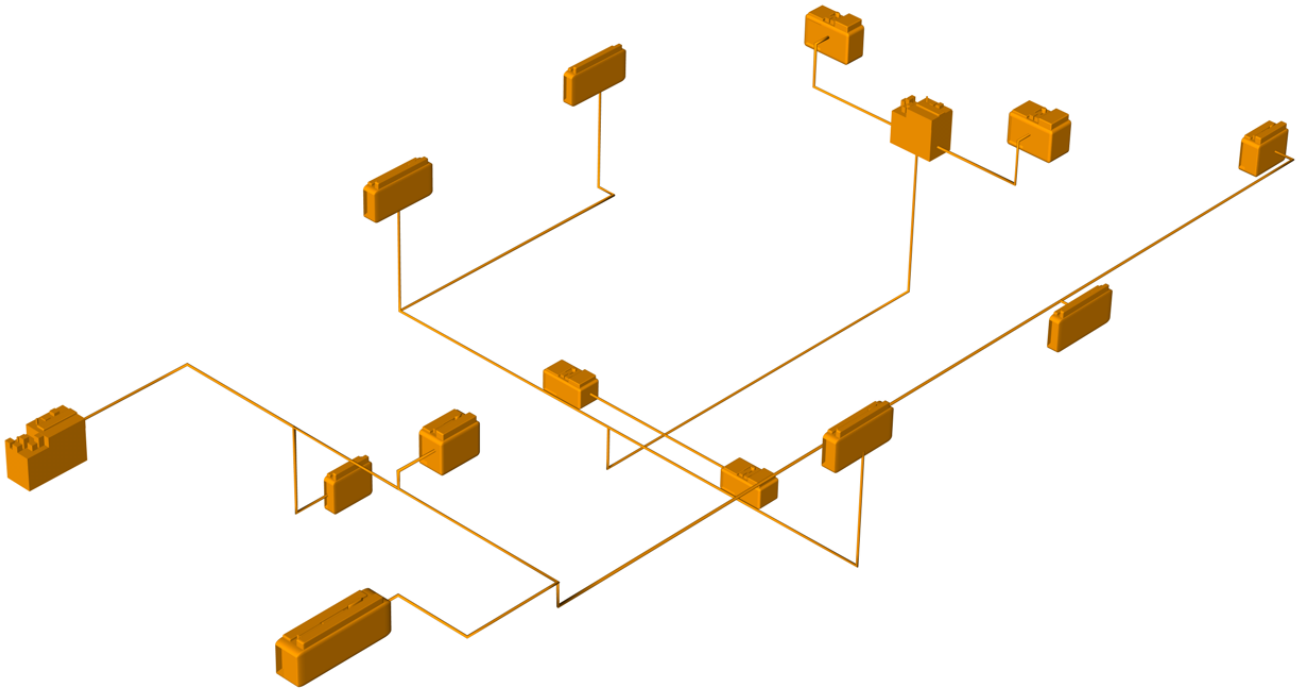
- **STEPbySTEP**
  This section gives you a short overview how to set up your project in just a few steps. Use the step by step description to start up with basic information and get your project basically running.
- **Expert Knowledge**
  This section gives you a detailed information about working with Vector AUTOSAR Solution. It contains expert knowledge based on STEPbySTEP description sections and information about tool use and general AUTOSAR concepts.

2 Basics

2.1 An Overall View

What we are talking about is an ECU, a module to be built-in a vehicle . The ECUs have to communicate with other ECUs and therefore almost every ECU participates in a bus system like e.g. CAN, FlexRay, LIN, MOST or IP. Any ECU within one bus system has to provide an identical interface to this bus system, because all ECUs have to share information via this bus system.

The illustration below shows an example of a CAN network that connects the ECUs of a vehicle.



All ECUs are built-up in a very similar way. There is a software part to perform the main job (application) of the ECU, e.g. to control the engine or a door. The other part handles the network communication and diagnostics.

2.2 MICROSAR - Vector's AUTOSAR Solution

The Vector MICROSAR Solution provides efficient and scalable basic software modules and an RTE for AUTOSAR ECUs.

The individual AUTOSAR basic software modules are grouped into MICROSAR products that of related functions. The configuration of the MICROSAR basic software with DaVinci Configurator Pro is simple, user-friendly, and consistent. The included command-line-based generators create optimized code for your ECU based on the configuration.

Cross Reference

To get more information about the MICROSAR concept refer to General Overview on page 76.

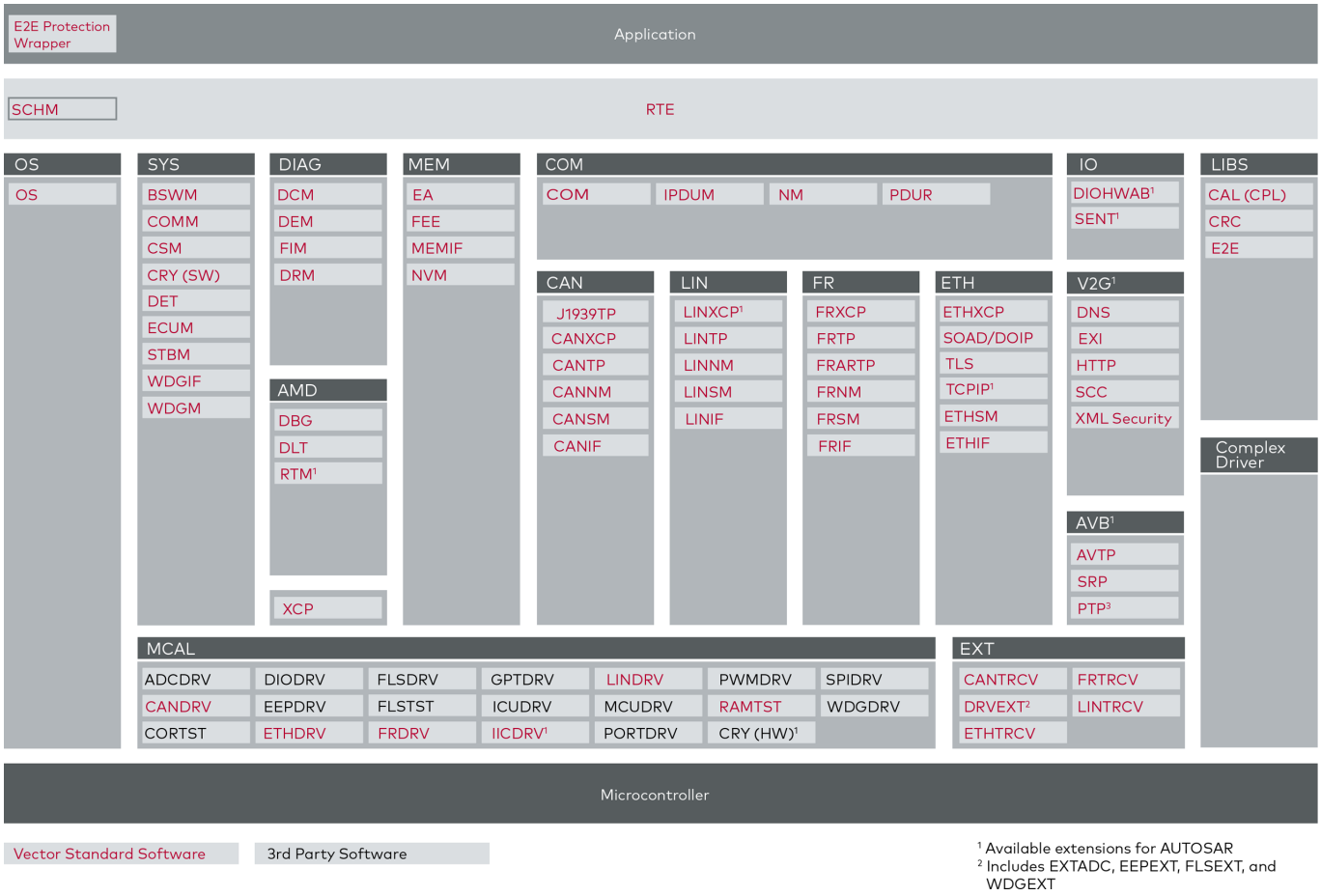
Cross Reference vVIRTUALtarget

Find more about vVIRTUALtarget usage here:

- Online Help vVIRTUALtarget
- TechnicalReference_vVIRTUALtarget.pdf

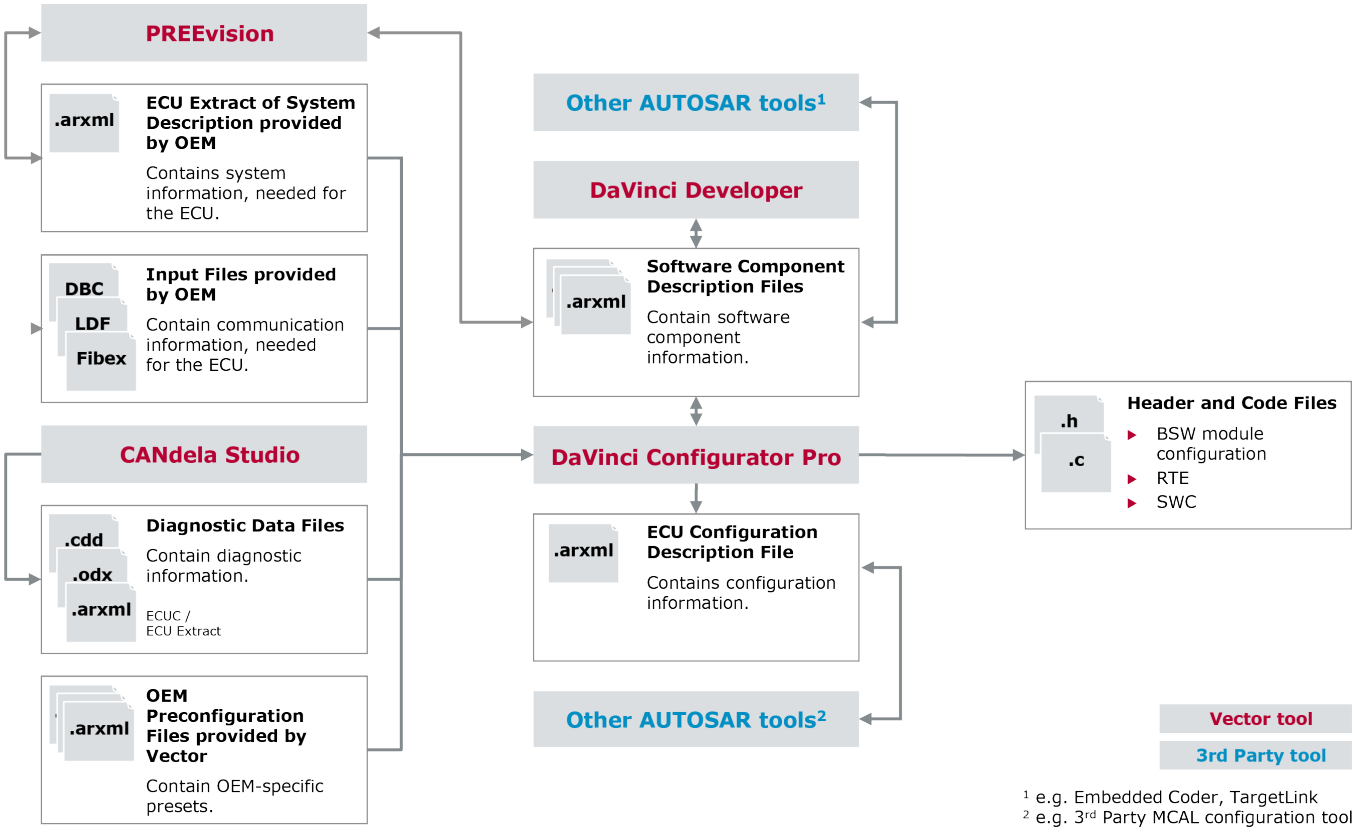
2.3 AUTOSAR Layer Model GM

The following figure shows the AUTOSAR Layer model for GM .



2.4 Configuration Workflow GM SLP2

The following figure shows the configuration workflow for GM SLP2:



I STEPbySTEP

This section shows you the few steps that are necessary to set up your project. Use it to start with basic information and get your project basically running.

1 STEP1 Setup Your Project

This section includes all information for setting up a new project. It starts with the situation after installation and provides you with all information about relevant tools.

Expert Knowledge

You need to know more? See section New Project on page 85.

1.1 Situation after Installation

- Before you start your first steps with GM SLP2 check whether you have installed all necessary tools and software.
- Have you read the **Installation Guide**? Have you received and installed all the necessary tools mentioned there?
- Have you installed the MICROSAR SIP?

NO? Then please read the installation guide carefully and follow the installation guidelines given there.

Cross Reference

You will find the installation guide document at the Vector Knowledge Base:
https://vector.com/kbp/entry/640/

YES? You are ready to continue and to start with your first steps with the GM SLP2 solution.

1.2 Setup Project via DaVinci Configurator Pro

Start the DaVinci Configurator Pro to set up your project.

Note

The DaVinci Configurator is located in your SIP Folder:
<SIPFolder>\DaVinciConfigurator\Core\DaVinciCFG.exe

1. Open **File|New**.
   A set of four configuration windows will open;
   General Settings,
   Target,
   Project Folder Structure, and
   Tools
2. Enter the necessary Information for each window and click **[Next>]** to get to the next window.

Note for No-Communication Projects

For projects without communication modules like e.g. DEM only, just confirm the windows **Target** and **Tools** as this information is not relevant.

Especially the **Target** window will show pull-down menus where **(undefined)** is preset and nothing else can be selected. This is not relevant for your use case and can be confirmed with **[Next]**.

1.2.1 General Settings

The following settings are required for project setup.

- **SIP root path**
  The SIP root path will be defined automatically based on the DaVinci Configurator Pro location and cannot be changed by the user!
  The DaVinci Configurator Pro is always located within the SIP.
- **Project Name**
  The name of your project (e.g. MyECU). It will be used as name of the generated ECUC files and for the **DaVinci Developer workspace**.
- **Project Folder**
  This is the root path of your project. Select an existing one or define a new one.
- **Create entries in the start menu**
  The checkbox **Create entries in the start menu** is set as default. This is a comfortable feature that enables you to open your tools and project folder directly from the Windows start menu.

1.2.2 Target

Check **Derivative**, **Compiler** and **Pin layout** (automatically set, based on SIP information).

If you want to use **Postbuild-Loadable** or **Postbuild-Selectable** for your project, activate the necessary support option.

1.2.3 Project Folder Structure

Define your output paths here. You can use the given defaults or change paths to fit your project's needs.

Note

Default paths are based on the location of the defined project folder.

ECUC File Granularity

Additionally, you can specify whether the ECUC should be saved as one file (Single file) or is split into several files (One file per module).

Note

Split to several files is necessary to realize a multi user concept. Refer to Multiple User Concept.

1.2.4 Tools

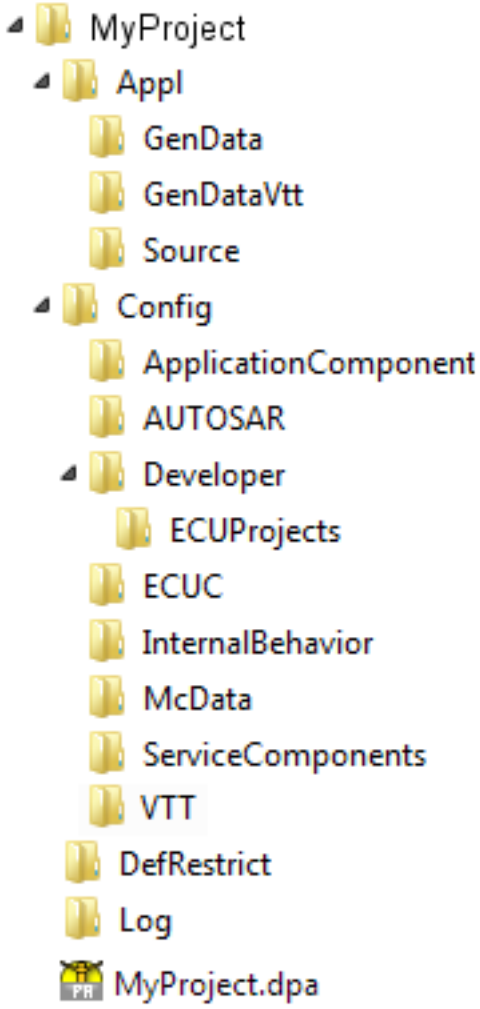
> **DaVinci Developer Workspace**
If you have a RTE and you use the DaVinci Developer select the checkbox **Create DaVinci Developer Workspace** and define the path to the DaVinci Developer executable the path to the workspace.

> **Automatic Update of DaVinci Developer Workspace**
Option used for updating the DaVinci Developer workspace during the project update process. Details see DaVinci Developer help.

3. Click the button **[Finish]** to create your project.

1.3 Result Project Folder - Result of the Project Setup

The DaVinci Configurator Pro automatically creates and stores all relevant files to the project folder. The content of the folder looks like shown and described below .



1.3.1 Appl folder

The **Appl** folder contains further folders **GenData**, **GenDataVtt** and **Source**. In **GenData** the configuration tools generate their output files, if vVIRTUALtarget is used, the generated output will be stored in **GenDataVtt**. The **Source** folder is meant to carry your source code files.

1.3.2 Config folder

The **Config** folder contains the following folders:

> **ApplicationComponents**
This is a folder to put in additional SWC-Ts (Software Component Types). The DaVinci Configurator Pro reads in all *.arxml files of this folder.

- **Developer**
  All relevant project data for DaVinci Developer is located in this folder.
- **ECUC**
  This folder contains the created ECUC files. Before you add your input file(s), these files are only placeholders without content.

File Name	Description
<ProjectName>.ecuc.arxml	ECU Configuration Description
<ProjectName>.ecuc.Initial.arxml	Initial ECU Configuration Description for internal tool use only.

- **InternalBehavior**
  This folder contains the bswmd files for all activated modules in the DaVinci Configurator.
- **McData**
  Folder is used for storing measurement and calibration data.
- **ServiceComponents**
  The DaVinci Configurator stores all generated Service Component Prototypes to this folder.

Note

After you have added your Input file(s) and updated the configuration in the DaVinci Configurator, the **System** folder will be added to the **Config** folder.

- **System**
  The input file(s) and an additional created input file extract for communication use are stored to this folder.

Do not edit manually!

These files must not be modified manually!

- **VTT**
  In case of activated Virtual Target option, the vVIRTUALtarget project file is located here.

1.3.3 <ProjectName>.dpa

Project File[1] includes all relevant project data. Additionally the context menu of this file is the control center to open your tools with already loaded projects.

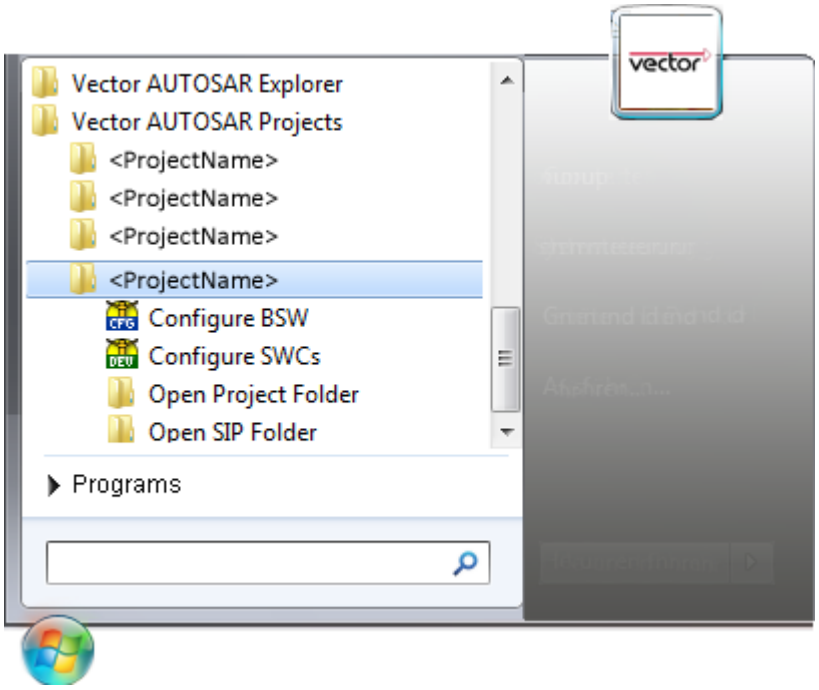
1.3.4 Log Folder

Folder for the generated log files.

1 <ProjectName>.dpa

1.4 Start Menu - Result of the Project Setup

The DaVinci Configurator Pro creates a link to start the tools with already loaded configuration. You will find this link in **Start | Programs | Vector AUTOSAR Projects | <ProjectName>** if **Create entries in the startmenu** was selected while project setup.



Info

You can also use the ***.dpa** file to start your project with already loaded configuration. Therefore right-click on your ***.dpa** file in the Windows Explorer and select **Configure BSW** to open the DaVinci Configurator Pro with already loaded configuration or **Configure SWCs** to open the DaVinci Developer for Software Design and Data Mapping.

If the Virtual Target option is enabled in the project, then **Open vVIRTUALtarget project** is additionally available.

1.5 DaVinci Configurator Pro Project

After finishing project set-up, a new empty project is loaded within the DaVinci Configurator Pro.

Note

Editors to configure modules will be available after input file import (see Add Input Files on page 27) or after module activation (see Activate Your BSW Modules on page 34).

2 STEP2 Define Project Settings

Before you start with the configuration it is necessary to define some project-specific data. This is the definition of project-specific data and the definition of all necessary input file(s) for communication and diagnostic purpose. Additional custom workflow steps, external generation steps and the activation of the necessary Basic Software Modules have to be done in the DaVinci Configurator Pro Project Settings view before.

Expert Knowledge

You need to know more? See section Project Settings on page 91.

Note

In case of OS only projects, skip the next sections **Add Input Files** and **Define Custom Workflow Steps** and **External Generation Steps** and go on with Activate Your BSW Modules on page 34.

2.1 Add Input Files

Open Input Files editor via **Project | Input Files** or use the input file button [1] of DaVinci Configurator Pro toolbar.

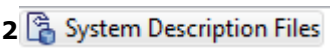
Note

Any change of the input files requires an update of the configuration. Add all relevant files before starting the update process.

2.1.1 Add System Description Files

Add your System Description File first. Therefore open the **System Description Files view**[2]. First you have to decide if you add AUTOSAR Files (arxml)[3] or Legacy Files (dbc, xml, ldf)[4].

1 

2 

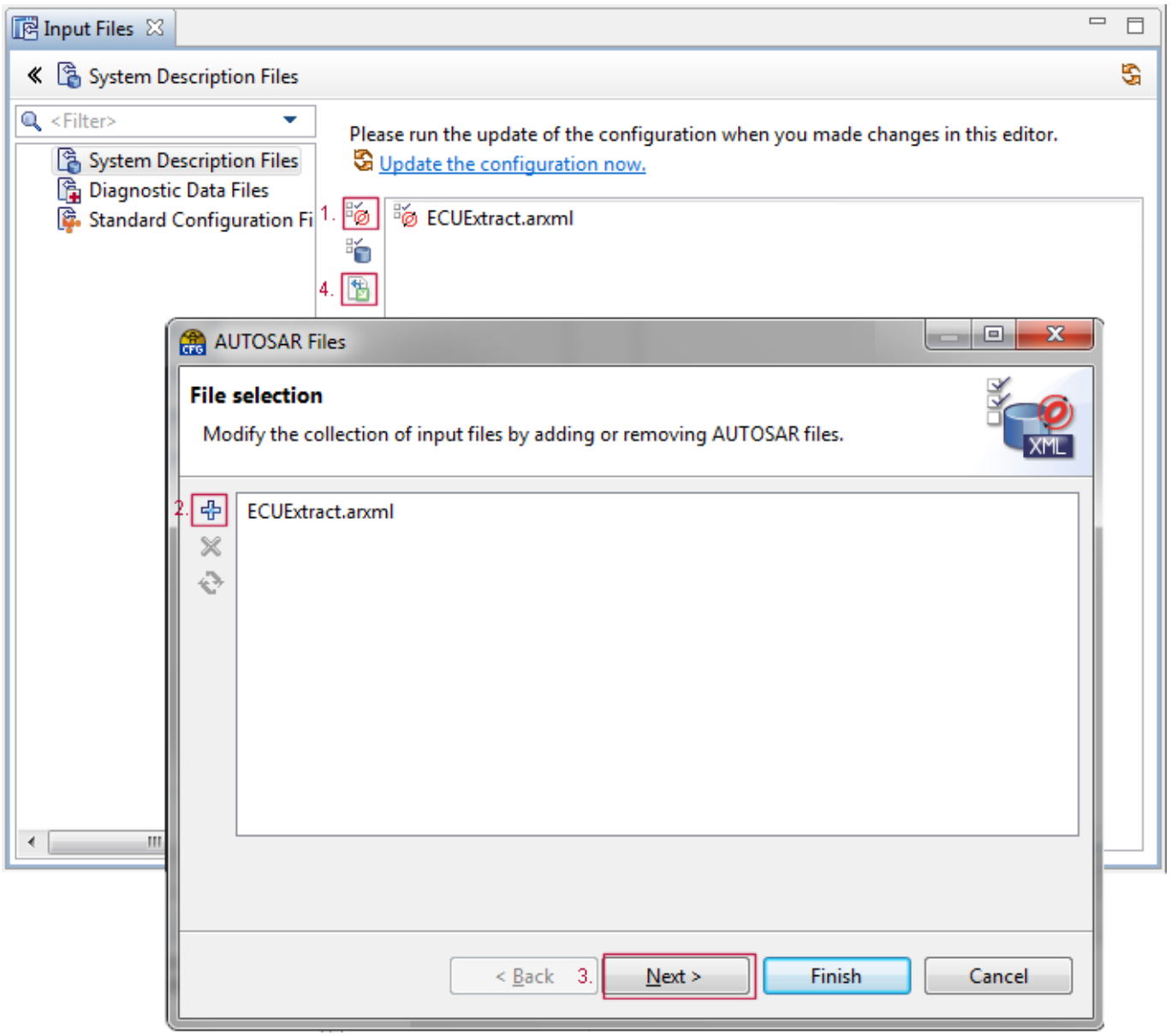
3 

4 

Note

The following section describes the import of AUTOSAR files. If you use legacy file as input, you have to perform nearly the same steps, only for 1. Legacy Files button[1] has to be selected.

1 



1. Use the AUTOSAR Files button[1] to open the window where you can add your AUTOSAR files.
2. Add your Input Files via **[+]**

Note - Only necessary if DBC is used as input file

There is a document TechnicalReference_DbcRules_Vector.pdf that describes necessary DBC rules to help you creating your DBC file. You find this document in the TechnicalReference folder of your delivery.

1

3. Go on with **[Next]** and select the **ECU** instance.
4. Make your input file path relative to your project folder via Make Path Relative button[1]

2.1.2 Add Diagnostic Data Files

Note - Diagnostic Extract as Input!

If your diagnostic input file is a **Diagnostic Extract**, then add this file as if it was a System Description file as described in the previous chapter.

If diagnostic is used, you have to add a diagnostic data file. If not, you can skip this section. There are different use cases to import diagnostic data, decide if you import your data from a diagnostic data file or from an already existing configuration.

Use Case 1: Import Diagnostic Data from ODX, PDX or CDD[2] File
Open your project in the DaVinci Configurator and go on with the following Steps to import your diagnostic data from a diagnostic data file.Open the Input Files editor via **Project | Input Files** or use the input file button [3] of DaVinci Configurator Pro toolbar

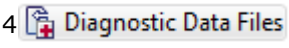
1. Open the Diagnostic Data Files view[4] in the Input File editor
2. Open the Diagnostic Fileset Configuration window via Add, remove or modify diagnostic data button[5]
3. Browse for the **Diagnostic Description File** (*.cdd/*.pdx/*.odx)
4. Select ECU and Variant

---

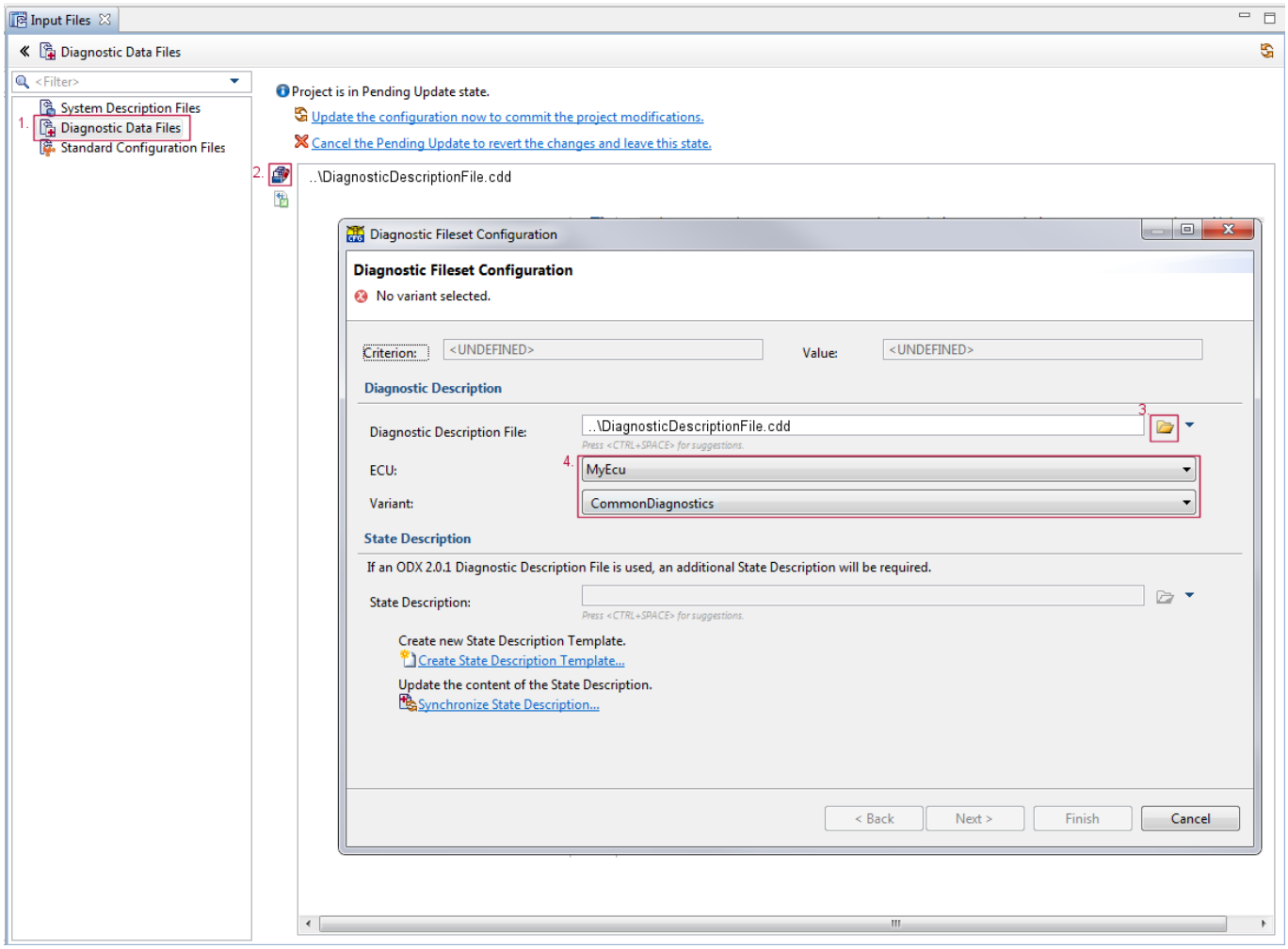
1 

2(Complex Drivers) The Complex Drivers are software modules that are not standardized by AUTOSAR. They have access to other BSW modules, the RTE and direct hardware access. For example, these modules are not standardized communication drivers for SPI or legacy software.

3 

4 

5 



Use Case 2: Import Diagnostic Data From an Existing Configuration
Open your project in the DaVinci Configurator and go on with the following steps to import your diagnostic data from an existing Configuration.

1. Start Import Assistant via **File | Import**
2. Add **[+]** ECUC File and go on with **[Next]**
3. Select modules which should be imported and close the **Modul Configuration Import** dialog via **[Finish]**.

Make your input file path relative to your project folder via Make Path Relative button[1]

2.1.3 Add Standard Configuration Files

In some cases an OEM-specific preconfiguration is necessary, therefore add the Standard Configuration File provided by your OEM. Add fragments of ECUC modules which will be used as project

1

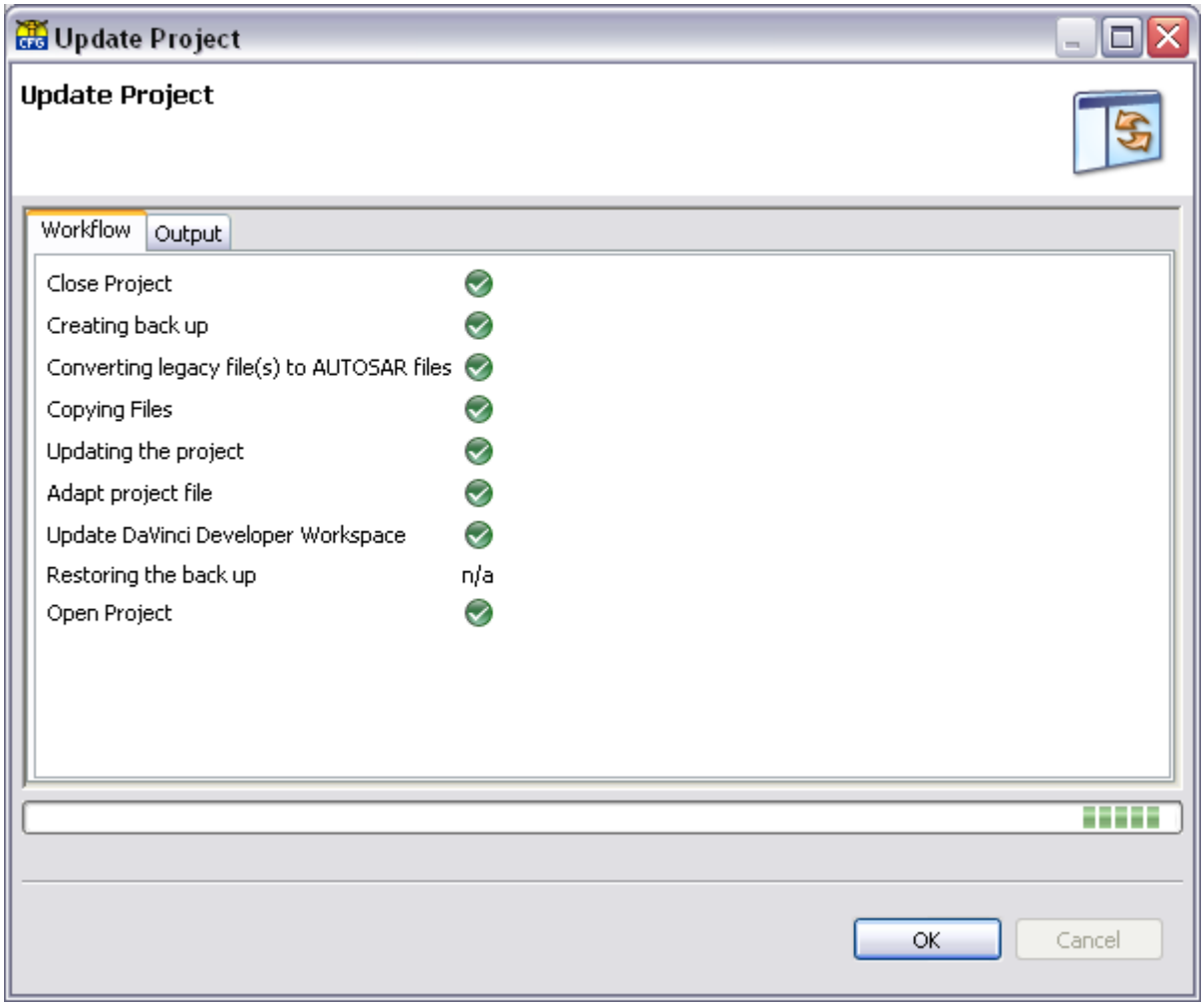
specific mandatory configuration.

2.1.4 Update Configuration

Any change of the input files lead to Pending Update state of the project, this requires an update of the configuration.

Update the configuration via .

What happens during the configuration update process?



The project file will be updated based on loaded input/diagnostic files.

The update was **successful?**

Note

After the first update/import of data bases all required modules are automatically enabled. This information is derived from the data base. If further modules are required or some shall be removed please refer to section Activate Your BSW Modules on page 34.

2.2 Define Custom Workflow Steps and External Generation Steps

To define custom workflow steps/external generation steps, open **Project Settings Editor** via **Project | Project Settings** or use the project settings icon [1]ofDaVinci Configurator Pro toolbar.

2.2.1 Custom Workflow Steps

In some cases it is necessary to define custom workflow steps, e.g. to Generate SWC Templates.

Then perform the following steps.

1. Select **Custom Workflow** | **Custom Workflow Steps**
2. Define **Name** and **Parameters** e.g. to Generate SWC Templates Rte[3]: -g i.

2.2.2 External Generation Steps

You have external software that needs to be started during the code generation of DaVinci Configurator? Or just before? Or afterwards?

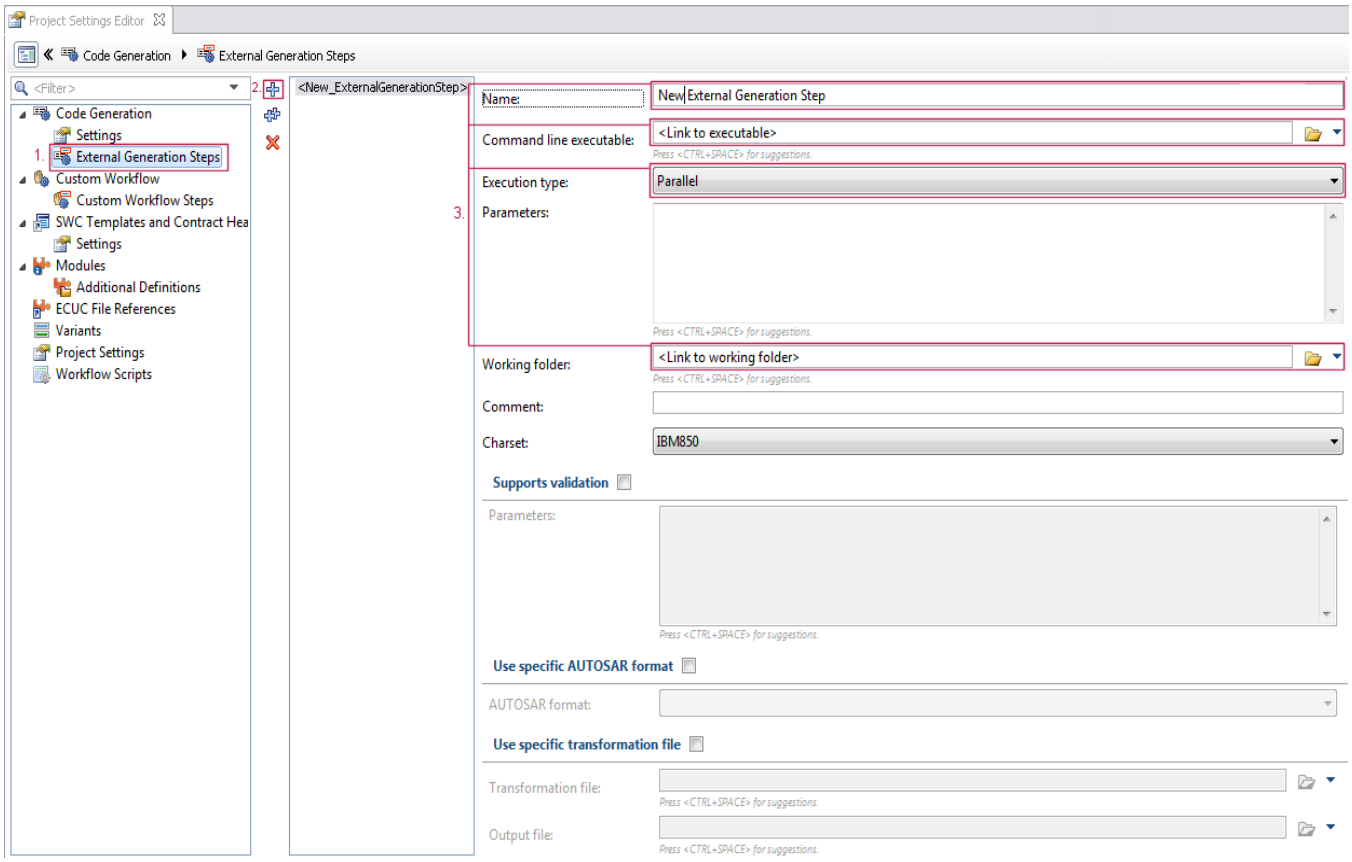
Then link your software to the **External Generation Steps** of DaVinci Configurator Pro.

How? Follow these steps.

12
2

[3](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

1. Select **Code Generation** | **External Generation Steps**
2. Add **[+]** step
3. Define
   Name,
   Command line executable,
   Execution Type and
   Working folder.



2.3 Activate Your BSW Modules

Before you start configuring your Basic Software, you have to activate all modules you need for your project. Therefore open **Project Settings Editor** via **Project** | **Project Settings** or use project settings icon[1] at DaVinci Configurator Pro toolbar.

Select **Modules** to see all modules currently activated for your project.

Info

It depends on your Input File which modules are loaded initially.

1

You need additional modules?

1. Click **[+]** and select the source of the module definition.
2. Select the module you would like to add to your configuration and confirm with **[OK]**.

Info

Multiple selection is possible for activating or deactivating modules.

Not all modules are needed?

To delete not used modules, select the module and delete it via **[x]**.

Caution!

It is not recommended to delete initially activated modules from project!

You have a completely configured module and would use this for the current configuration?

1. Make sure, that the module you would like to import, is deactivated in your current project.
2. Select **File** | **Import**
3. Add **[+]** ECUC file including the configured module
4. Click **[Next]**
5. Select the module configurations to be imported

Note

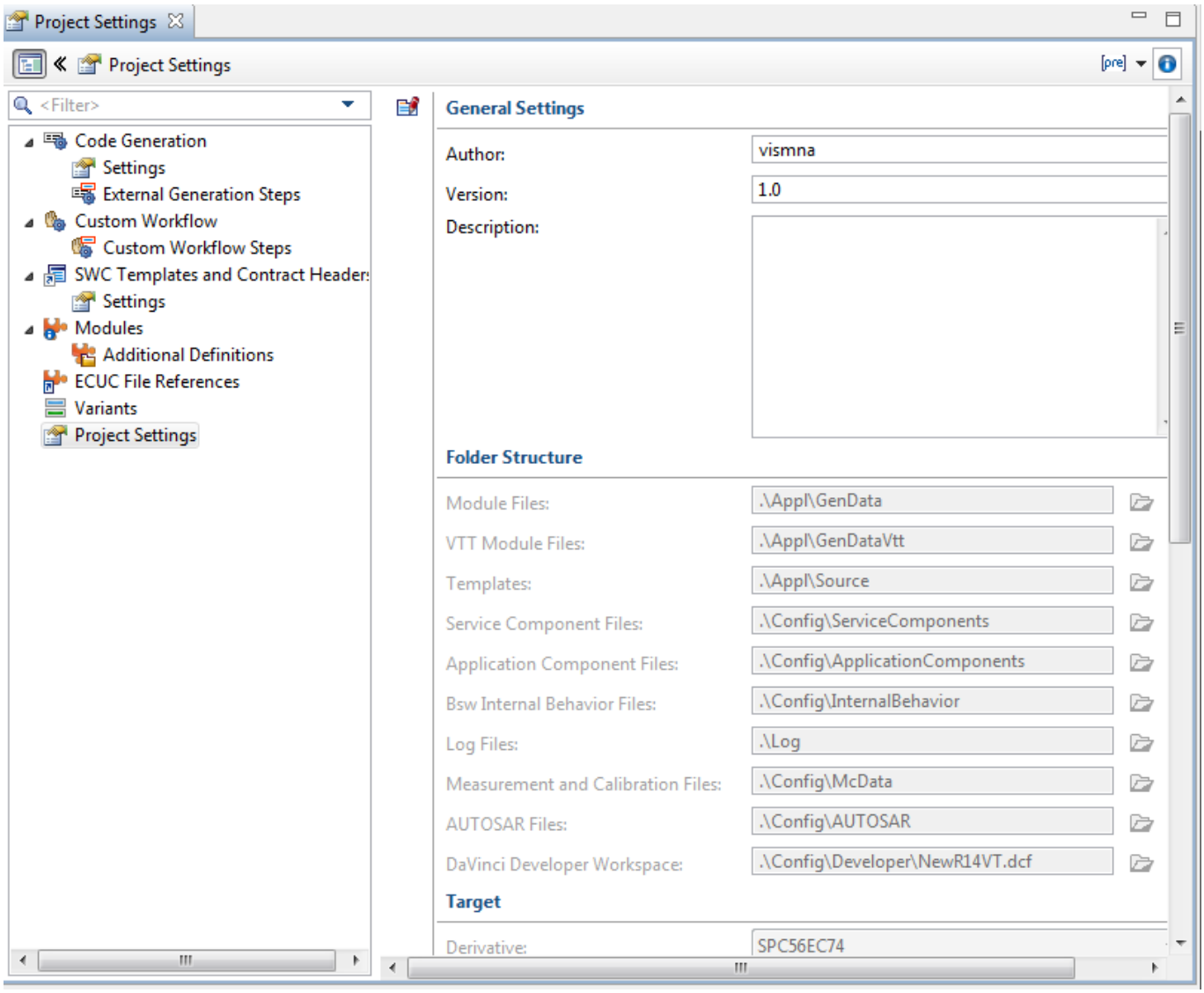
It is not allowed to import BSW modules which are currently activated.

6. Click **[Finish]**

2.4 Change Project Settings

In some cases it is necessary to change project settings, which was defined while project setup. Therefore activate editing via Edit Project settings icon[1].

1 



2.4.1 Postbuild Support

Caution!

These parameters should be changed for compelling reasons only. Parameter changes, require a lot of manual adjustment after project update!

Note

In case of project migration, the parameters will be set automatically.

3 STEP3 Validation

Start initial validation process to solve the first warnings and errors.

The DaVinci Configurator Pro provides validation routines that run in the background and are called live validation. At the beginning of the configuration, the project is incomplete and much information in the system is still missing. Therefore the DaVinci Configurator Pro offers a powerful solving algorithm.

Expert Knowledge

You need to know more? See section Validation on page 94.

3.1 Start Solve All Mechanism

First of all start solving mechanism via the Solve All button[1] at the validation view of the DaVinci Configurator.

Note

Not all errors can be fixed with **Solve All** mechanism.

Some errors might have two or more solutions, these errors cannot be fixed automatically and need your decision.

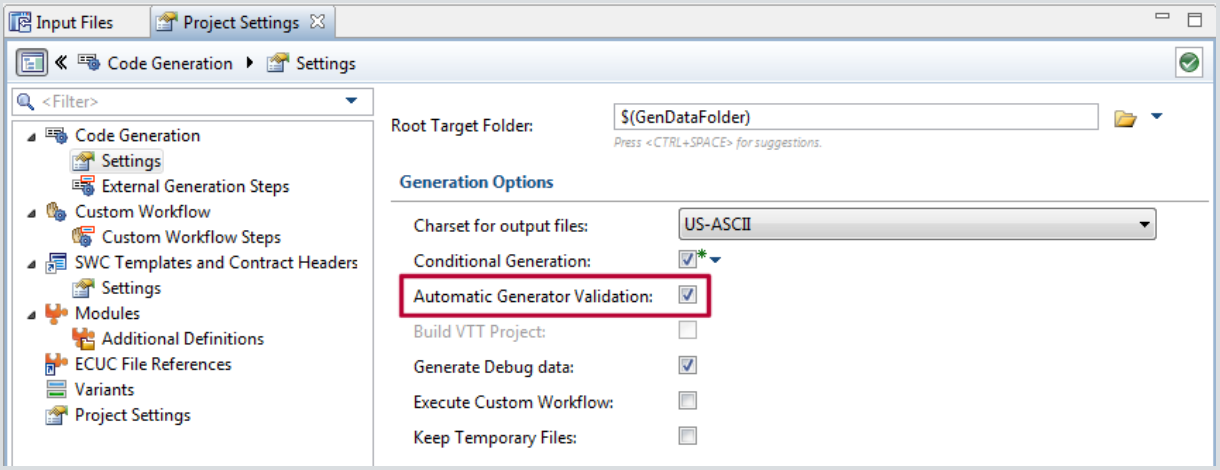
3.2 Live Validation - Solving Actions

The DaVinci Configurator Pro provides a live validation during project configuration. For some parameter errors the live validation process offers possible solution(s) in the validation view of the DaVinci Configurator Pro. These errors are marked with a bulb.

1 

Note 3rd Party Module Live Validation

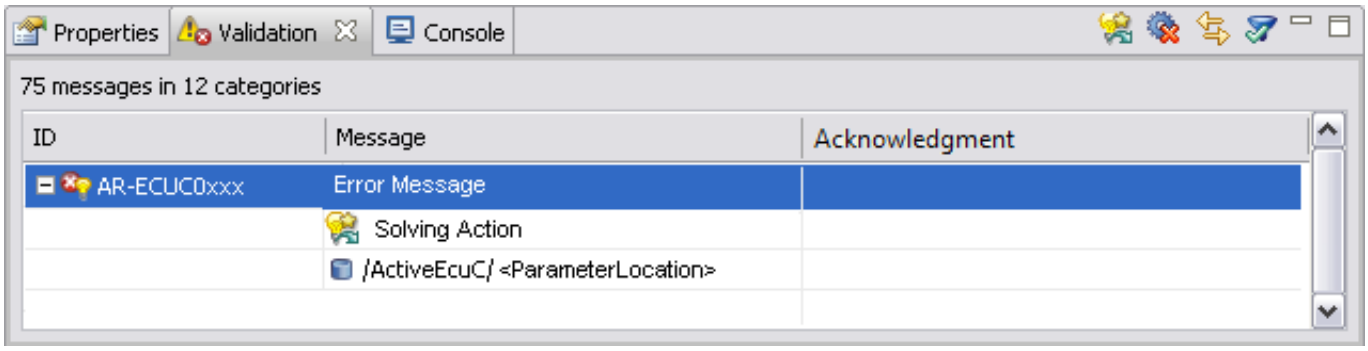
If **Automatic Generation Validation** is activated in **Project Setting Editor**, the configuration of 3rd party modules is also checked while live validation.



Check if the suggested solving action is correct. Double click the error message to open the configuration window of the parameter.

Solving Action is correct?

Start the solving action via double click on the solving action entry in the validation window.



Note

If the solving action is not correct, you can additionally use the link in the solving action to navigate to the parameter and configure the changes manually.

Cross Reference

You find more information about the feature Acknowledgment in the online help of the DaVinci Configurator Pro.

4 STEP4 Start BSW Configuration

Start the configuration of the BSW modules now. The target of this first description is a basically working configuration, using a minimal necessary configuration.

Expert Knowledge

You need to know more? See section Configuration with Configurator Editors on page 95.

Note

Changes in Configurator Editors could lead to errors during live validation!

If these errors include solving actions, check and start solving action in the validation view

4.1 Start Configuration with Configuration Editors

The configuration editors are use case-oriented and optimized for displaying or configuring those parts of the ECU configuration, which are related to the use case.

4.2 Base Services

The Base Services domain contains configuration editors for the configuration of the development error reporting, general purpose timer channels, microcontroller units and RAM test algorithms.

4.2.1 Development Errors

Open **Development Errors** | **Enable or Disable Error Tracing** and select necessary modules or domains to enable the development error tracing.

4.2.2 General Purpose Timer (GPT)

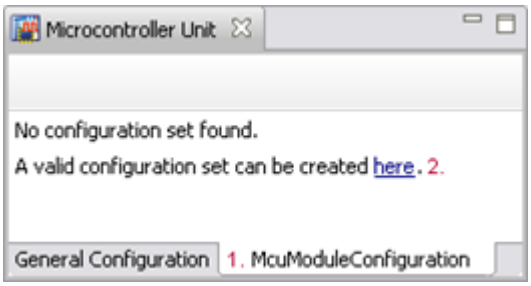
Use default settings.

Note

If the editor is deactivated (greyed out), you can activate the General Purpose Timer (GPT) module via click. The Modules Assistant opens to specify the short name of the module to be added. Click finish to add the module.

4.2.3 Microcontroller Unit

1. Open tab McuModuleConfiguration
2. Create new configuration via **[here]**.



3. Open **Clock Config Sets** | **McuClockSettingConfig** | **McuClockReferencePoint**, add [+] an **McuClockReferencePoint** and define **Clock Reference Point Frequency**

4.2.4 RAM Test

Some hardware-specific settings are necessary.

Note

If the editor is deactivated (greyed out), you can activate the RAM Test module via click. The Modules Assistant opens to specify the short name of the module to be added. Click finish to add the module.

4.3 Communication

The Communication domain contains configuration editors for the configuration of general parameters of the communication related modules, ECU Bus Controllers, protocol data units, data signals and the transport protocol.

Note

The following setting description are meant for demonstration purposes only and can differ from the settings, that are necessary for your project.

If there are modules describe, you do not have in your project, skip this chapter.

4.3.1 Communication General

Define central settings of the communication based modules.

CAN

1. Open **CAN** | **Miscellaneous**
2. Set **[...]** parameter **Counter Ref** to **SystemTimer**.
3. Set **Interrupt Category** to **CATEGORY 2**
4. Set **Interrupt Lock** to **DRIVER**

4.3.2 Bus Controller

1. Open the **CAN Controllers** | **<CANController>**

2. Define **[...]** Controller Default Baudrate
3. Define **[...]** CPU Clock Ref

Note

If no CPU Clock Ref is available, add a new reference via **[+]** in the Select Reference Target view.

4.3.3 PDUs

Configure the PDUs of the modules (depends on your project)

> CAN/LIN/FR
> CANIF/LINIF/FRIF
> CANTP/LINTP/FRISOTP
> PDUR
> IPDUM
> COM

4.3.4 Signals

Configure the communication signals of the ECU.

4.3.5 Socket Adapter Users

Use default settings.

Note

If the editor is deactivated (greyed out), you can activate the SOAD module via click. The Modules Assistant opens to specify the short name of the module to be added. Click finish to add the module.

4.3.6 Transport Protocol

1. Open **FlexRay** | **Tx Pdu Pools** and **Rx Pdu Pools** to add **[+]** a new **Tx Pdu Pool** and **Rx Pdu Pool**.
2. Open **FlexRay** | **Connections** | **FrTpConnection** and set **[...]**

> Connection Control,
> TxPduPool and
> RxPduPool.

4.4 Diagnostics

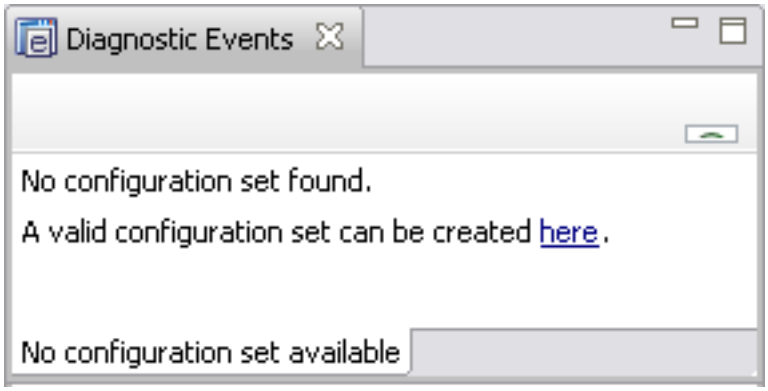
The Diagnostic domain contains comfort editors for the configuration of extended data records and snapshot records, diagnostic events and production error handling of the individual BSW modules and a automatic setup of the NV memory blocks.

4.4.1 Diagnostic Event Data

Use default settings as these have been derived from input files.

4.4.2 Diagnostic Events

If no configuration set is found, use the hyperlink **here** to create a valid configuration set.



4.4.3 Production Error Handling

Use default settings.

4.4.4 Setup Event Memory Blocks

The event memory block configuration needs to be updated. Start the Editor and confirm with **[finish]**.

Note

The automatic setup of the NV memory blocks is required for diagnostic purpose.

4.5 Memory

The Memory domain contains comfort editors for the configuration of the general parameters of the memory related modules, the memory block in the non volatile memory blocks.

4.5.1 Memory General

If FEE is used, define

- **BSS Thresholder Reserved**
  BSS - Background Sector Switch
- **FSS Thresholder Reserved**
  FSS - Foreground Sector Switch

4.5.2 Memory Blocks

Open **Memory Partitions** | **PartitionConfiguration** and define

- **Partition Device**
  Add the reference to their device of the partition

Note

If the editor is deactivated (greyed out), you can activate the EA/FEE module via click. The Modules Assistant opens to specify the short name of the module to be added. Click **[finish]** to add the module.

4.5.3 Optimize Fee

If Fee[1] is used, start FEE Optimization. The number of chunk instances (parameter FeeNumberOfChunkInstances) of the Fee blocks will be optimized. Please review the new value or select "skip" to leave the parameter.

Note

If the editor is deactivated (greyed out), you can activate the FEE module via click. The Modules Assistant opens to specify the short name of the module to be added. Click [finish] to add the module.

4.6 Mode Management Editors

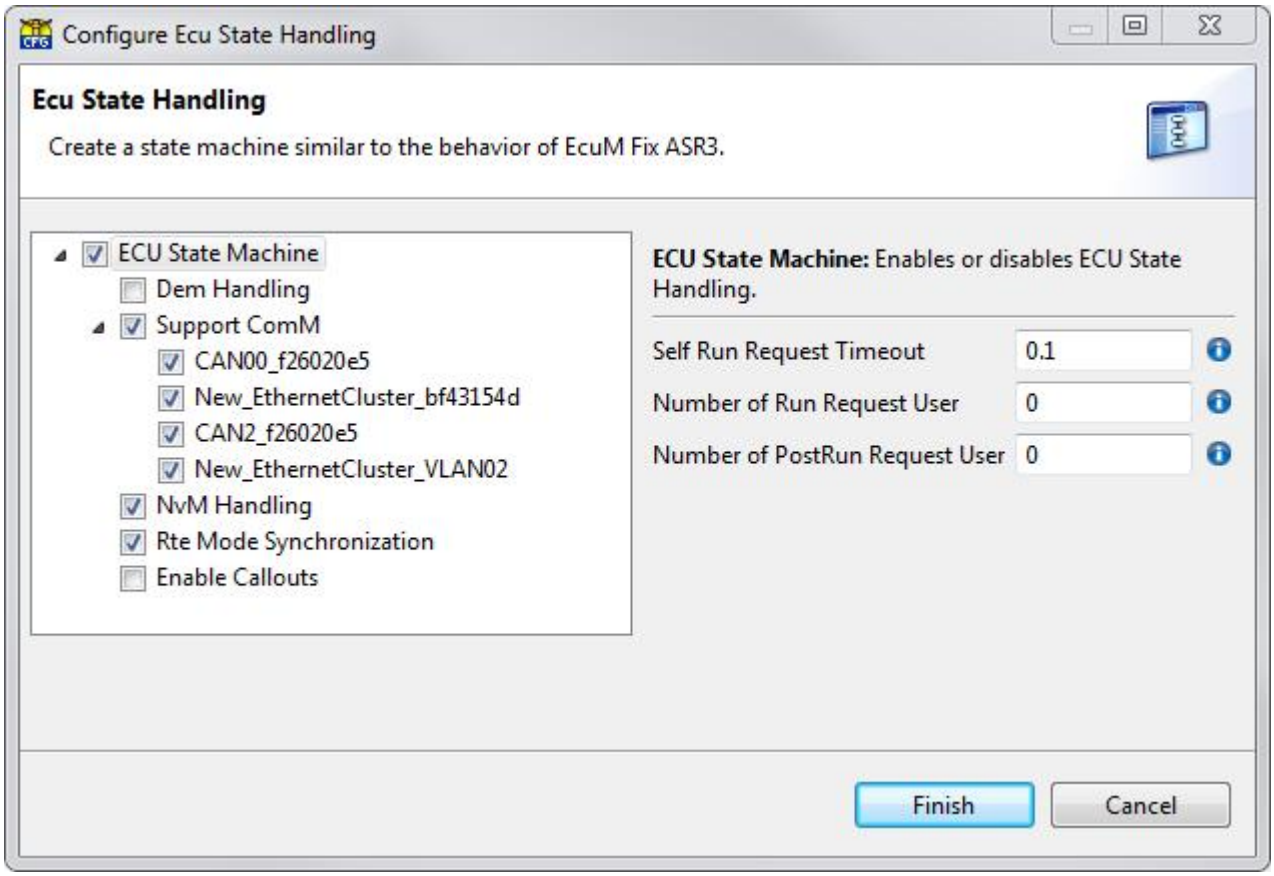
The Mode Management domain contains comfort editors for the configuration of BSW Management, ECU Management, initialization and restart sequences of the BSW modules, sleep modes and wakeup sources, supervised entities and watchdog modes.

4.6.1 BSW Management

Auto Configuration: Ecu State Handling
Open Auto Configuration: Ecu State Handling and configure Ecu State Handling settings via Configure Ecu State Handling hyperlink[2]. The Configure Ecu State Handling dialog opens.

[1](Flash EEPROM Emulation) The Fee module offers a hardware-independent interface for accessing flash data and uses a flash driver ((Fls) for this. In addition to reading, writing and clearing data, the Fee module also distributes write accesses to different areas of flash memory, so that all flash cells are uniformly stressed, which increases their life-time.

2 Configure Ecu State Handling



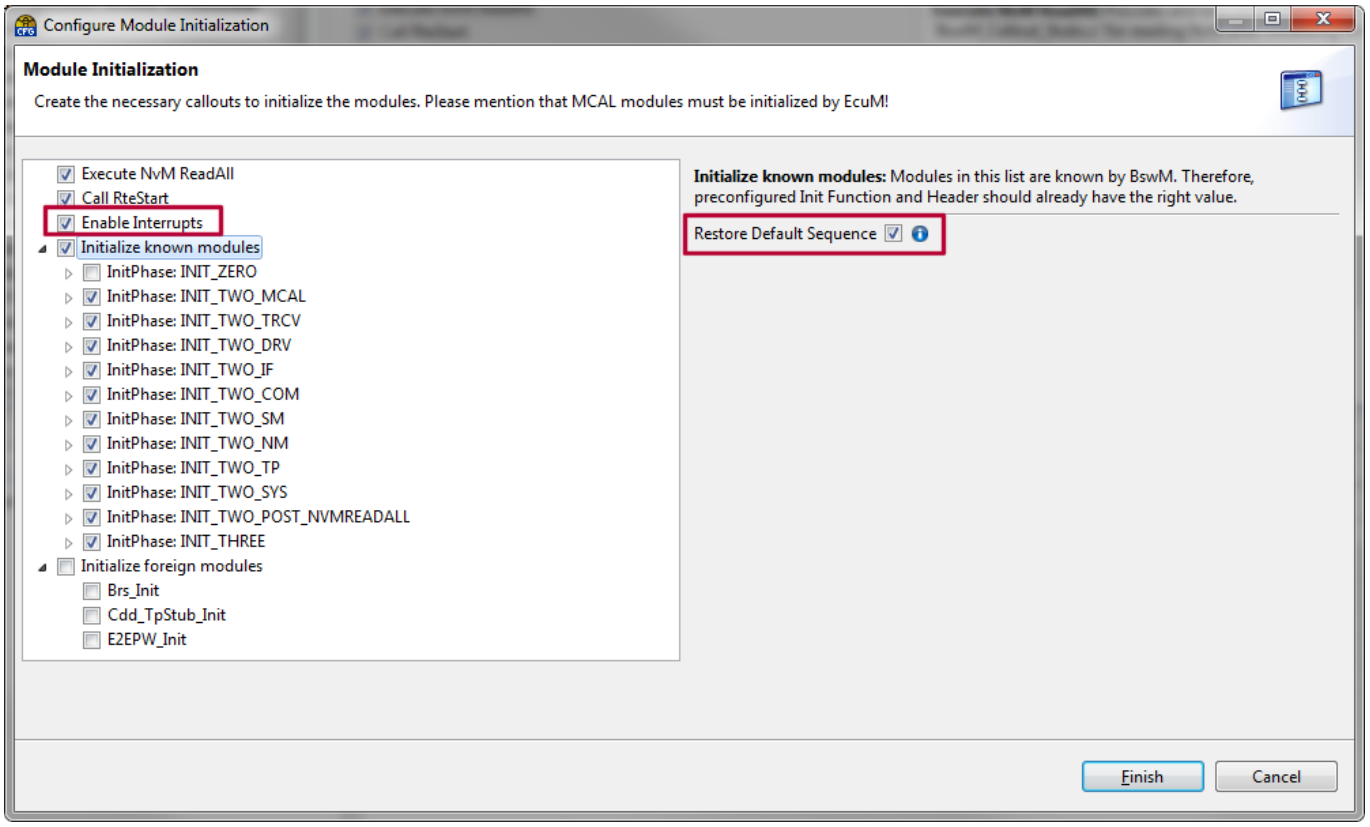
Select the necessary use case definitions to create a state machine similar to the behavior of EcuM Fix in AUTOSAR3.

Note

If you select Ecu State Machine, you have to define the project-specific settings **Self Run Request Timeout** and **Number of Run Request User**.

Auto Configuration: Module Initialization
Open Auto Configuration: Module Initialization and configure Module Initialization via Configure Module Initialization hyperlink[1]. The Configure Module Initialization dialog opens.

1 Configure Module Initialization



Select module to create the necessary callouts to initialize the modules.

4.6.2 Activate Interrupts of Peripherals Devices

Activate **Enable Interrupts** to get the callout `BswM`[1]`_AL_SetProgrammableInterrupts`. Use this callout to activate interrupts of the peripherial devices. Exceptions form this rule are explained in the technical reference of the drivers.

Note

Configure module initialization at initialization view.

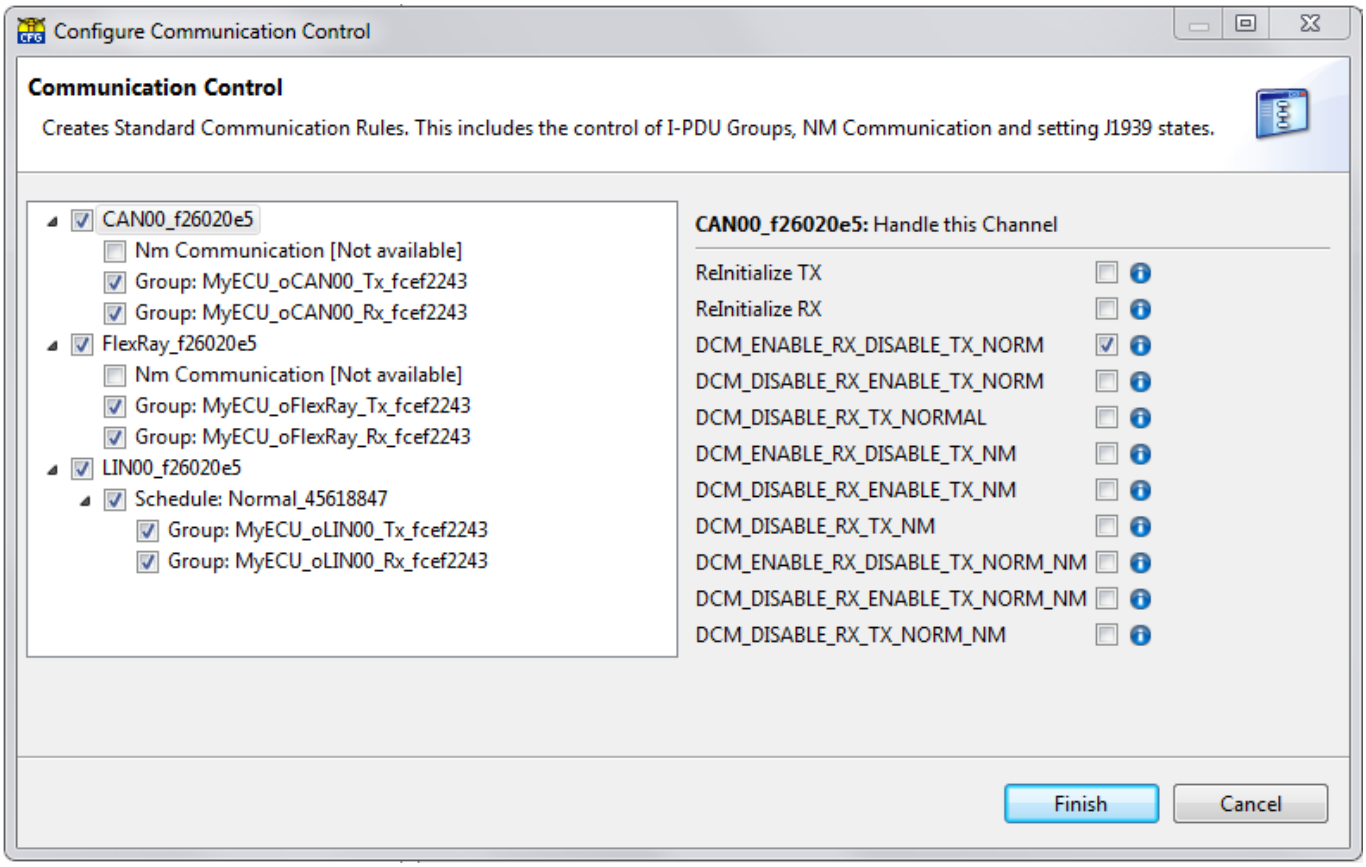
Note

If **Restore Default Sequence** is activated, the lists will be sorted automatically, all new entries will be located at the end of the list.

Auto Configuration: Communication Control
Open Auto Configuration: Communication Control and configure Communication Control via Configure

[1](BSW Mode Manager) The BswM module contains vehicle mode management and application mode management. It processes mode requests from SWCs or other BSW modules, and it performs actions based on the arbitration, such as control of deadline monitoring, switching schedule tables, and handling of IPDU groups. In conjunction with the EcuM, the BswM module is responsible for starting up and shutting down the ECU. The BswM also coordinates the multi-core partitions.

Communication Control hyperlink[1]. The Configure Communication Control dialog opens.



Select the bus systems for which you would like to create standard communication rules.

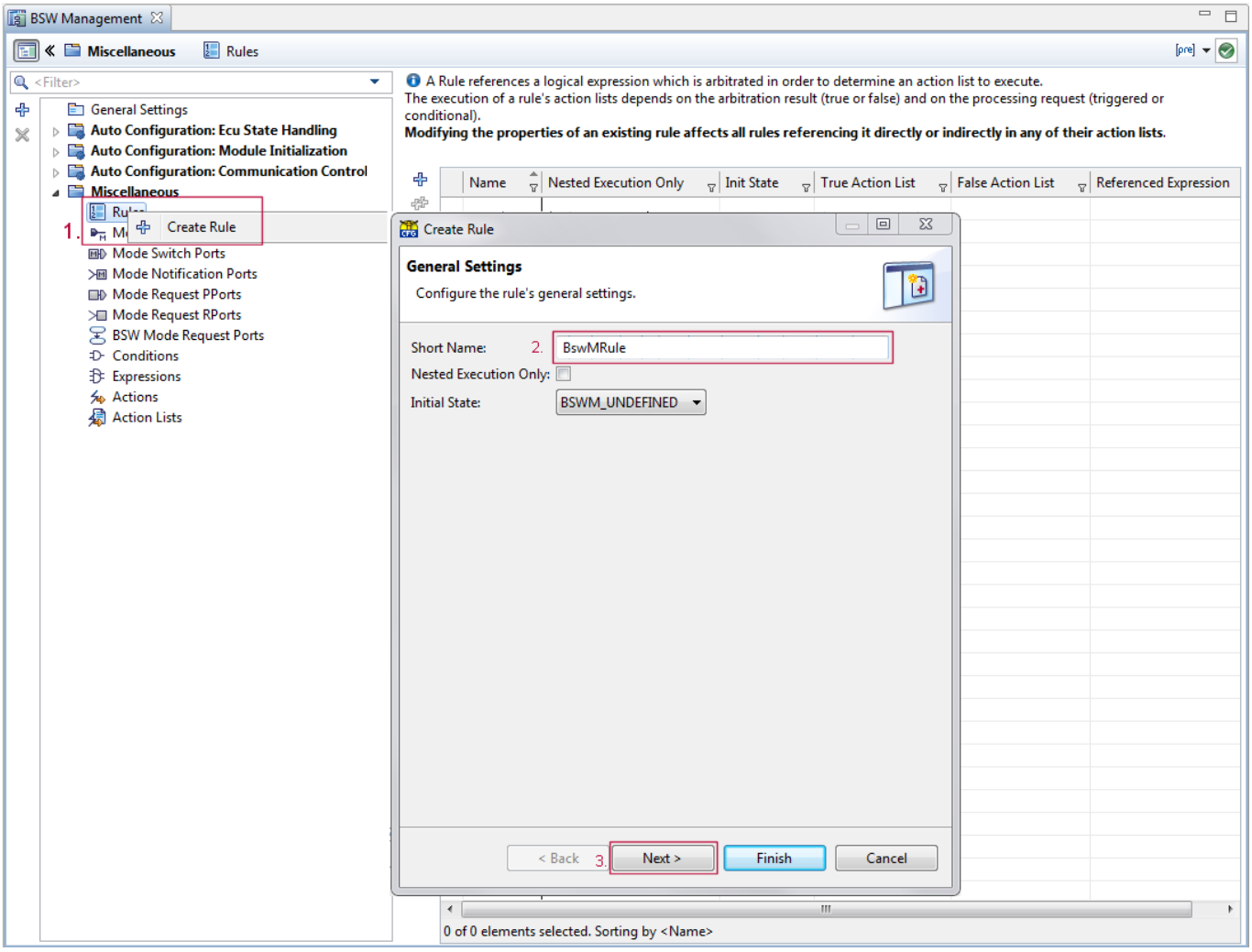
Custom Configuration
If necessary you can define additional Custom BswM Configuration information.

Note

To create a rule, at least one **logical expression** and one **action list** is necessary. If they are not available, you can additionally configure them via **Create Rule** dialog.

1 Configure Communication Control

1. Open **Custom Configuration**, right click **Rules** and select **Create Rule**.
2. Define **Short Name**
3. Go on with **[Next]**



4. Define a non-empty expression for the resulting condition
5. Configure Action List True, therefore define actions to be executed if the rule's condition evaluates to true.
6. Configure Action List False, therefore define actions to be executed if the rule's condition evaluates to false.

Note

If no actions are configured, no action list will be created.

Note

It is possible to create groups **[+]** subdivide the BSWM configuration into logical sections.

The elements can be organized in groups using drag and drop in the editor tree.

The groups are only for organization reason, functional usage is not possible. You can't move elements from auto configuration to the created BSWM Group.

4.6.3 ECU Management

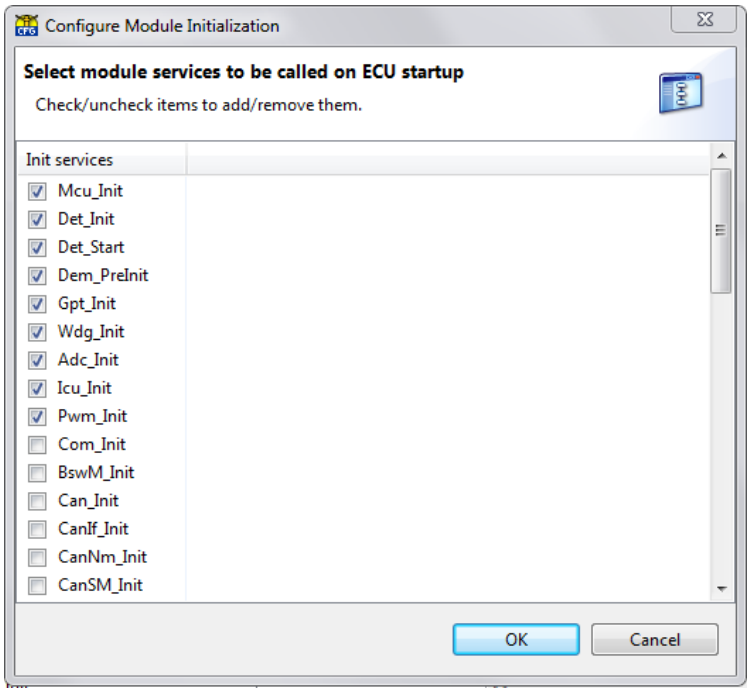
1. Open **EcuMCommonConfiguration** and define the **OS Resource** which is used to bring the ECU into sleep mode.
2. Open **EcuMFlexConfiguration** and define Mcu[1] **Mode Ref** which is being restored after a sleep.

4.6.4 Initialization

Open Initialization editor to configure initialization and restart sequence of the basic software.

Pre-OS Initialization Sequence

Open **Configure Module Initialization** via Configure Sequence button[2].



Select all MCAL modules and the DET to call them on ECU startup. Confirm with **[OK]** and order the elements according to your needs with the **Move up** and **Move Down** buttons.

---

[1](Micro Controller Unit Driver) The MCU driver provides the services for: - A microcontroller reset triggered by software - Selecting microcontroller states (STOP, SLEEP, HALT, etc.) - Configuring wake up behavior - Managing the internal PLL clock unit - Initializing RAM areas with predefined values.

2

4.6.5 Watchdogs

Use default settings.

Note

If the Editor is deactivated (greyed out), you can activate the Wdg[1] module via click. The Modules Assistant opens to specify the short name of the module to be added. Click **[finish]** to add the module.

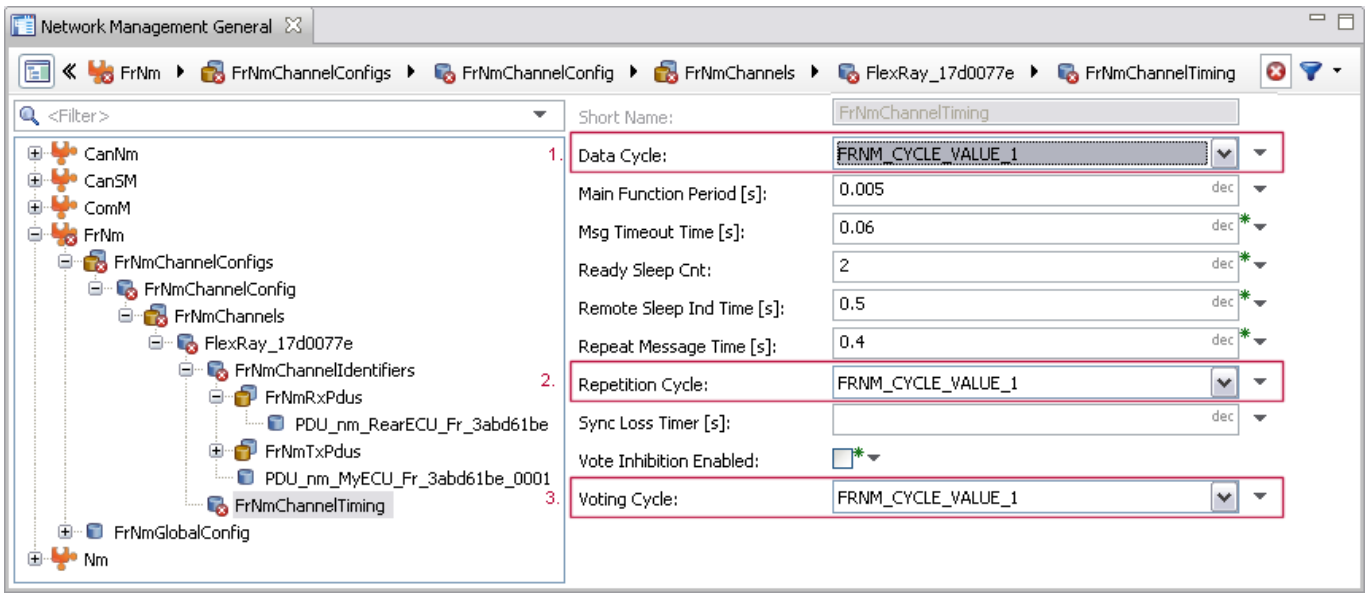
4.7 Network Management

The Network Management domain contains editors for the configuration of the general network management parameters and users which are relevant for the ECU communication management.

4.7.1 Network Management General

Open FrNm[2] | **FrNmChannelConfigs** | **FrNmChannels** | **<FRNMChannelIdentifiers>** | **FrNmChannelTiming** and define

1. Data Cycle,
2. Repetition Cycle and
3. Voting Cycle.



[1](Watchdog Driver) This module provides services for controlling and triggering the watchdog hardware. The trigger routine is called by the Watchdog Manager ((WdgM). For safety-related ECUs, the Wdg module must be developed according to ISO 26262.
[2](FlexRay Network Management) This module is responsible for network management in FlexRay. It synchronizes the transition to the bus sleep state.

Note

For all other NM modules, use default settings.

4.7.2 Communication Users

Use default settings.

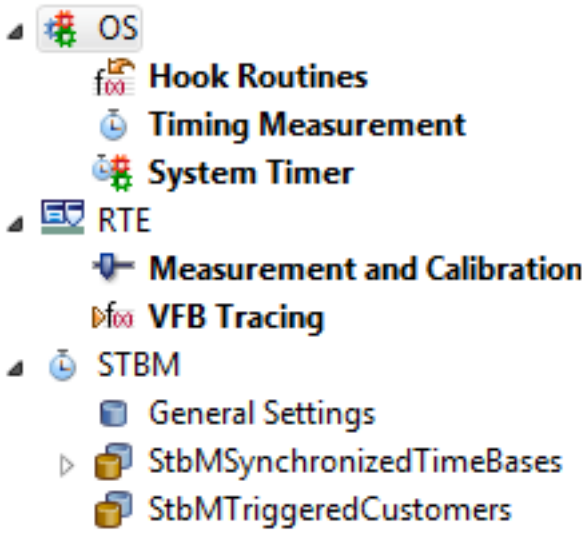
4.7.3 Partial Networking

Use default settings.

4.8 Runtime System

The Runtime System domain contains comfort editor for the configuration of general RTE and OS parameters, operating system and mapping assistants.

4.8.1 Runtime System General



OS

1. Open **Runtime System General** | **OS** and activate/deactivate the following settings if necessary.
   > Stack Monitoring
   > Use GetServiceId
   > Use Parameter Access
2. Define **Scalability Class**, **CC** and **Schedule**.
3. Open **OS** | **Hook Routines** and activate/deactivate the following settings if necessary.
   > Startup Hook
   > Error Hook
   > Shutdown Hook

- Pre-Task Hook
- Post-Task Hook
- Protection Hook

Note

The **Shutdown Hook** is required for shutdown of the ECU to power off.

The **Error Hook** is recommended to inform the application on any error events in the OS.

4.8.2 ECU Software Components

Service Components
Open **ECU Software Components** | **Service Components** and check if all Service Components are added to your project. If not create new component prototypes, based on component types created by the DaVinci Configurator Pro.

Note

The component types are stored in **<ProjectFolder>\Config\ServiceComponents**.

4.8.3 OS Configuration

Create Tasks

Open **OS Configuration** | **Task**. Add the following tasks, define Schedule, Priority and Type.

- Init_Task (Autostart has to be enabled!)
- Rte[1]_Task
- Main_Task

Note

All events and alarms which are required are created by the RTE automatically.

Derived from the settings for the runnables and their activation (triggers) some OS events an OS alarms will be created automatically for RTE by the DaVinci Developer. The names start with **Rte_**.

[1](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

4.9 Go on with Basic Editor

Open the Basic Editor and configure additional settings, currently not supported by specific configuration editors.

Note: Bottom Up Approach - from hardware-dependent to hardware-independent modules

Start configuration with hardware dependent modules and continue with the hardware independent modules.

4.10 Start Solving Actions

Changes in Configurator Editors/Basic Editor could lead to errors during live validation!

If these errors include solving actions, check and start solving action in the validation view.

4.11 Start On-demand Validation

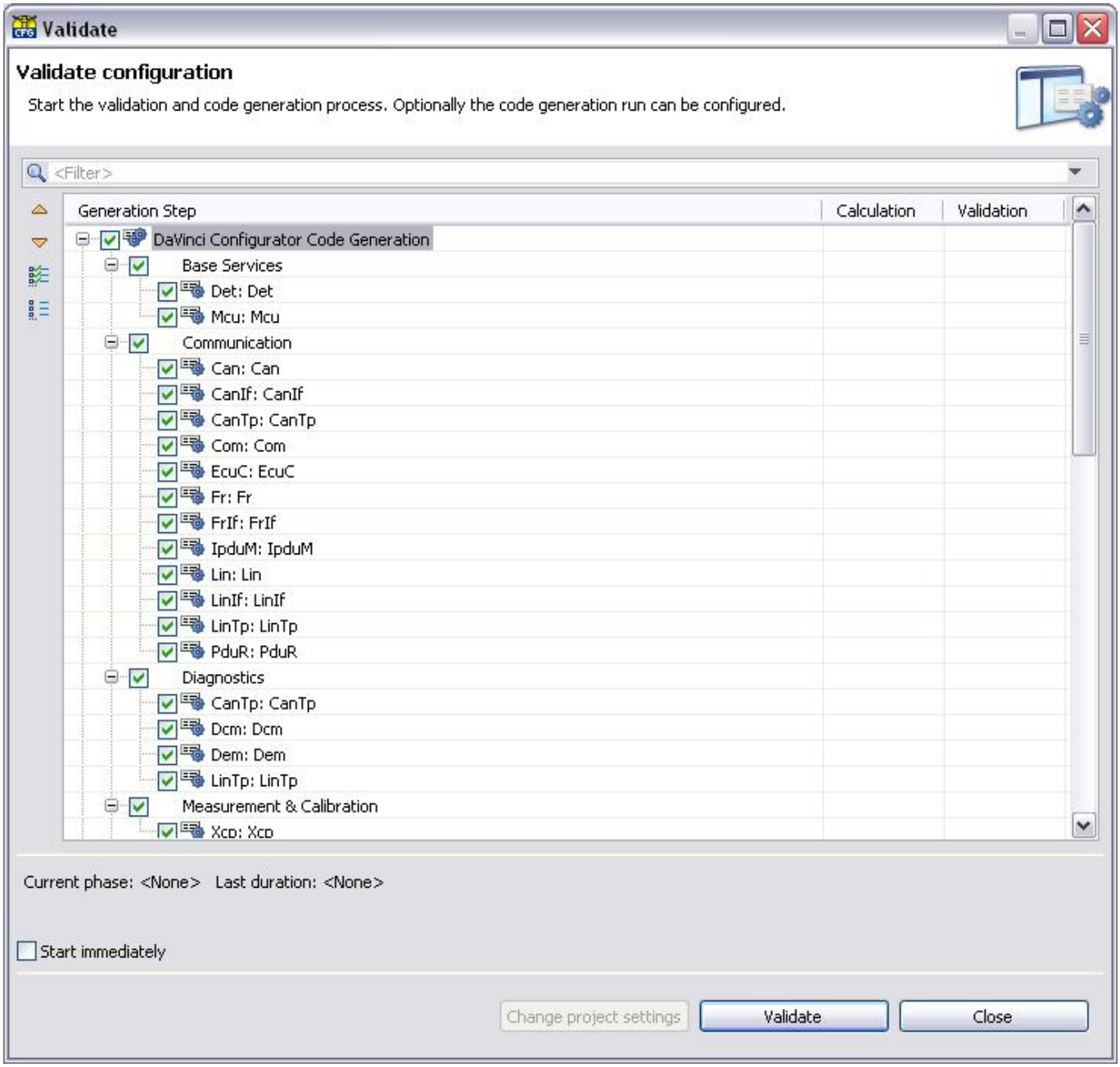
The on-demand validation is performed in addition to the automatic validation before starting the code generation or on explicit user request. The on-demand validation calls the specific validation function of the code generators.

1. Open **On-demand Validation** dialog via on-demand Validation button[1]
2. Deactivate **RTE**

Note

Currently the RTE is not configured completely, this causes errors while validation. The RTE configuration will be finished after SWC Design in the DaVinci Developer.

3. Start the Validation via **[Validate]**



1

Validation finished successfully?

Validation finished NOT successfully?

See error message in validation view and fix the configuration.

4.12 BSW Configuration finished

You have finished your BSW Configuration?

Now it depends on your project, how to go on. You have created a Developer workspace while setting up the project? Then you have to go on with the STEP5 Design Software Components on page 55 using the DaVinci Developer. If you have no Developer workspace, you can skip the following chapter an go on with STEP6 Mappings on page 56.

5 STEP5 Design Software Components

Switch to the DaVinci Developer to create, configure or modify software components, create or add ports and data elements. Connect software components and define runnables and their activation and interfaces (data access).

Expert Knowledge

You need to know more? See section Software Component Design on page 96.

5.1 Switch to DaVinci Developer

You can start DaVinci Developer with already loaded project in two different ways:

- **Start** | **Programs** | **Vector AUTOSAR Projects** | **<your project name>** | **Configure SWCs** (Only if Create entries in the start menu is selected)
- Right-click on your **<projectname>.dpa** and select **DaVinci Project Assistant** | **Configure SWCs**

5.2 Design Software Components

If the software components are not delivered by the OEM, use the DaVinci Developer to design your software components. How to do this is described more detailed in the expert knowledge part of this manual. Use the link above.

6 STEP6 Mappings

Start project mappings via assistants offered by the DaVinci Configurator Pro. It is also possible to perform data mapping within the DaVinci Developer.

Expert Knowledge

You need to know more? See section Mappings on page 117.

6.1 Perform Data Mapping within DaVinci Developer or DaVinci Configurator?

Via the data mapping, you connect data elements with network signals. Network signals have been loaded via import of ECU Extract of System Description.

Data Mapping can be done in DaVinci Developer or DaVinci Configurator Pro. Decide which tool should be used. In the following step, both variants are described.

Note

Data Mapping in DaVinci Developer has a higher priority than Data Mapping in the DaVinci Configurator Pro.

Data Mapping performed in DaVinci Developer cannot be overwritten in DaVinci Configurator Pro.

Cross Reference

To perform data mapping within the DaVinci Configurator Pro, skip this section and go on with section Switch (back) to DaVinci Configurator on page 59.

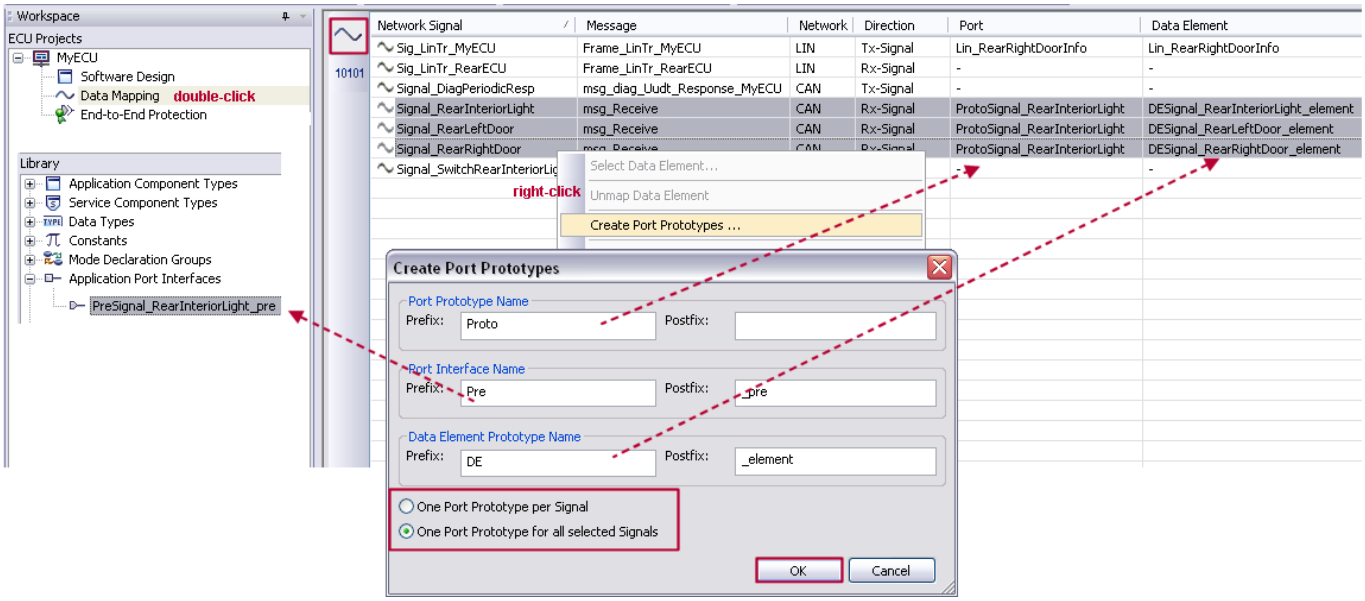
6.2 Data Mapping within the DaVinci Developer

The DaVinci Developer offers two ways of data mapping - **automatically** OR **manually**.

6.2.1 Data Mapping Automatically - DaVinci Developer

Network signal oriented approach
Select bus signals and let DaVinci Developer create all necessary data elements, ports and the mapping of the signals to the data elements.

Finally, you only have to drag'n'drop the port interfaces to your software components and create the connections. Here is how this works.



1. Double-click Data Mapping from the ECU Projects view
2. Select the Signal view mode [1]
3. Select signals to create Data Elements and Port(s)
4. Right-Click selected signals and select Create Port Prototypes … to open the Create Port Prototypes window.
5. Define prefixes and postfixes for the Port Prototype Name, Port Interface Name and Data Element Prototype Name.
6. Select whether you want One Port Prototype per Signal or One Port Prototype for all selected Signals.

Note

If you select One Port Prototype for all selected Signals, of course the signal must be of equal direction, only Tx signals or only Rx signals.

7. Confirm your settings with **[OK]**.
8. You see that one Port Prototype, three Data Elements and one Application Port Interface are generated. The pre- and postfixes help to easily find them in the list.
9. Select the Software Design view and see the delegation port that has been also created.
10. Now drag'n'drop the Application Port Interface form the library to your software component (same direction as the delegation port) and connect them.

6.2.2 Data Mapping Manually - DaVinci Developer

You can also connect the data elements with real bus signals manually.

---

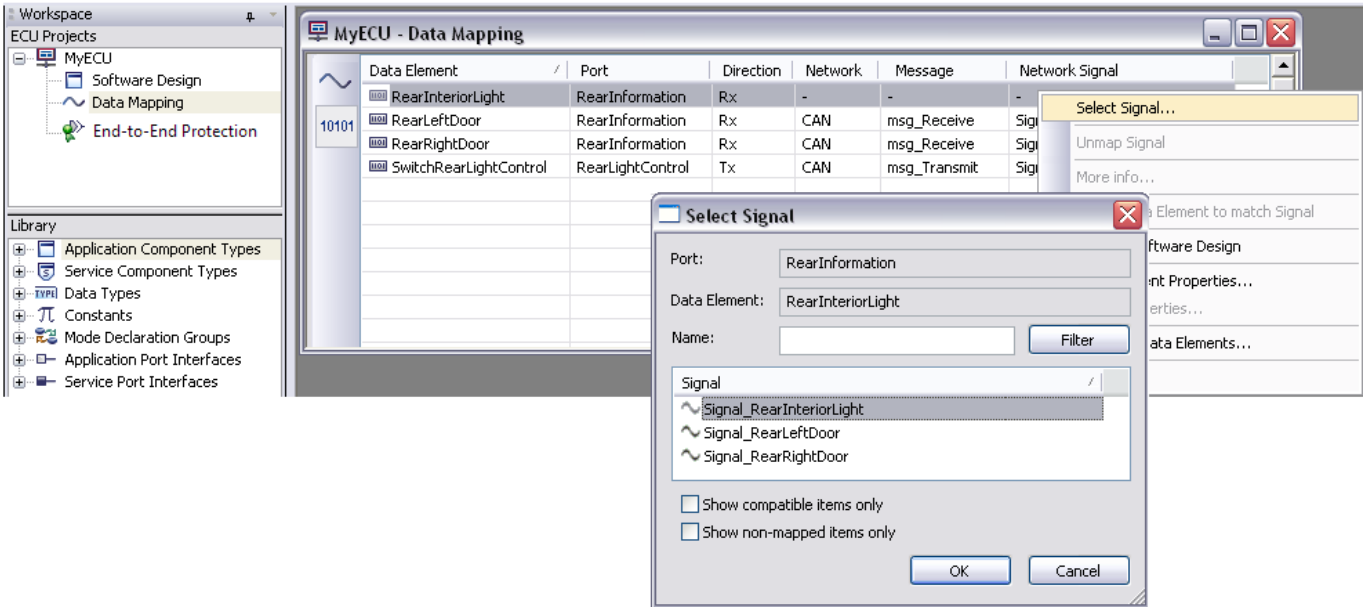
1 

Note

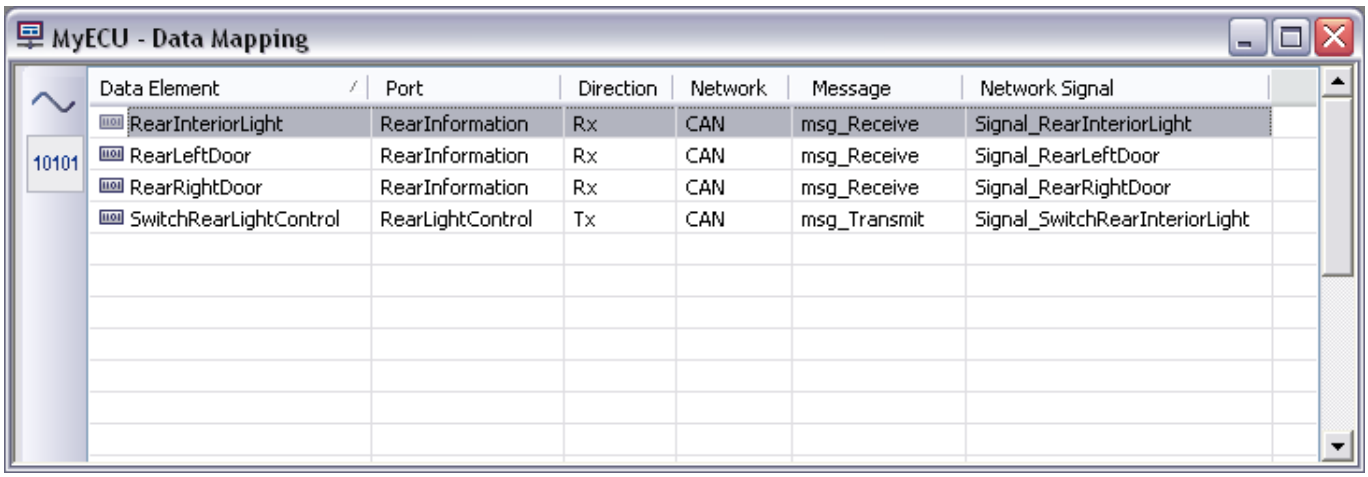
A prerequisite is that you have to create the necessary delegation ports before!

Select Data Mapping in the ECU Projects view and select e.g. the Data Element View Mode [1] and see a list of data elements. The dash (-) on the right side below Network, Messages and Network Signals shows that there is no assignment between data elements and messages/signals from the ECU Extract of System Description.

Right-click the **Data Element column** and click **Select Signal...**. Select the suitable signal in the Select Signal window and confirm with [OK]. Repeat this until every signal is mapped.



The result of the data mapping should look like this.



1 10101

6.2.3 DaVinci Developer - Save and close

You have imported your Service Components, configured your Software Components and finished Data Mapping?

Check [1] your workspace. If there are no messages in the **Output** view, save your project and close the DaVinci Developer.

6.3 Switch (back) to DaVinci Configurator

Start the DaVinci Configurator Pro (again) with already loaded configuration using one of the two possibilities.

> **Start** | **Programs** | **Vector AUTOSAR Projects** | **<your project name>** | **Configure BSW**
(Only if Create entries in the start menu is selected)

> Right-click on your **<projectname>.dpa** and select **DaVinci Project Assistant** | **Configure BSW**

6.4 Synchronize System Description

Changes by the DaVinci Developer require a synchronization of the System Description in the DaVinci Configurator Pro.

The necessity of a synchronization is detected by the validation process of DaVinci Configurator Pro and displayed in the **Validation view**. Start System Description synchronization via the validation message **RTE59000** | **Synchronize the System Description**.

6.5 Add Component Connection

Data elements can only be transported from one application component to another if their ports are connected via connectors. Create these connections using the Component Connection Assistant[2] (below **Runtime System**).

1. Start Assistant, select connector type and go on with **[Next]**
Select **Application Connector Prototypes** to connect software components or **Service Connector Prototypes** to connect service components
2. Select a **Component Prototype** and go on with **[Next]**
3. Select **Port Prototypes** and go on with **[Next]**
4. If necessary select additional search option and go on with **[Next]**
Initially the Component Connection Assistant will search matching port prototypes based on the port prototype name automatically
5. Check mapping. All Connector Prototypes are mapped with the right ports? Confirm with **[Finish]**

---

1 

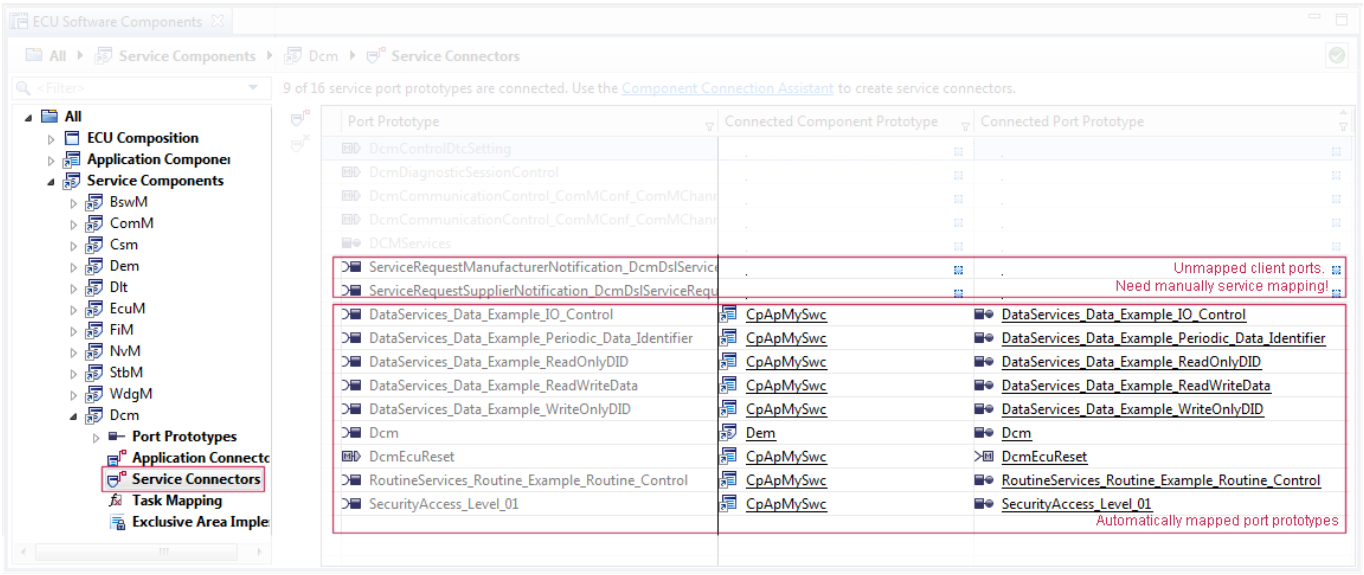
2  Add Component Connection

6.6 Service Mapping

6.6.1 Service Mapping via Service Component

If you have connected your service components via **Add Component Connection** in the previous step, the service mapping is basically done.

To define additional mappings manually you have to configure it in the service component directly. Therefore open **ECU Software Components** | **Service Components** and select the service component you want to map. Open service component and select **Service Connectors**.



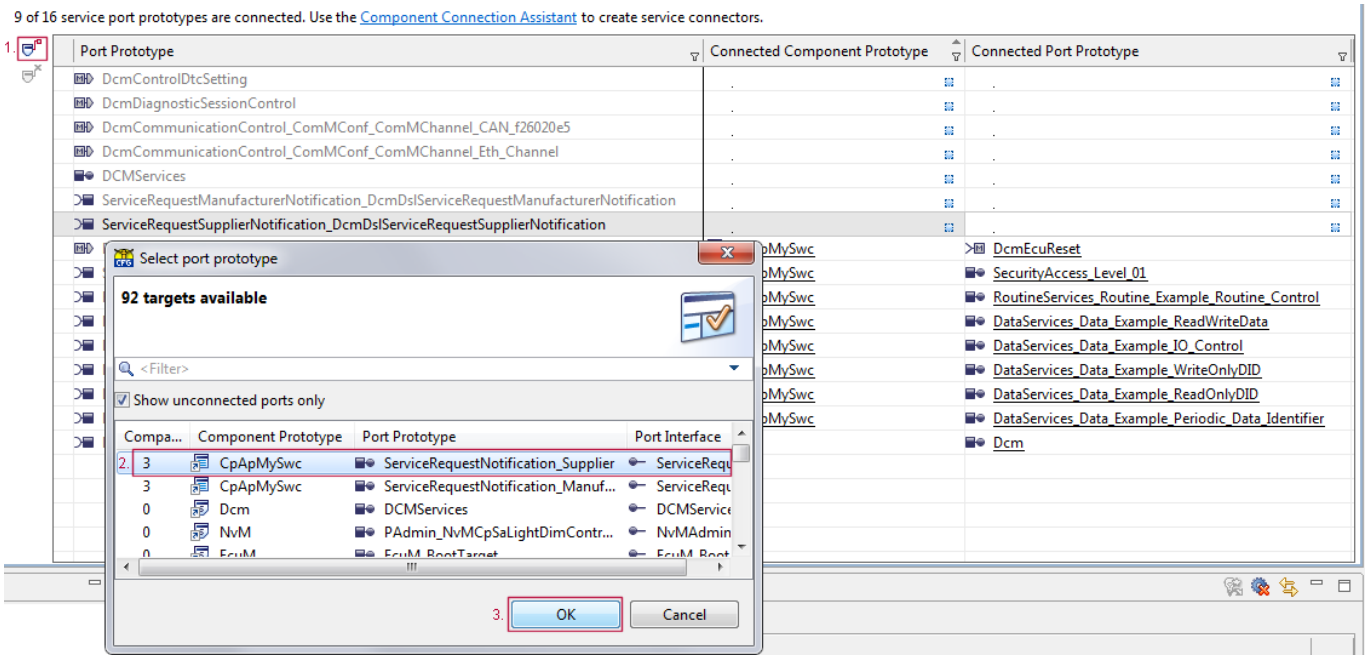
Now you can see all ports, the automatically matched connections and unmapped ports. **Is manually service mapping necessary?**

Note

Each client port[1] should be mapped to a server port[2]! For all unconnected client ports the RTE will generate a **dummy** function returning RTE_E_UNCONNECTED.

1
2

1. Select the service component port and use the connect button[1]
2. Assign the appropriate port prototype to the selected port of the service component
3. Finish the mapping with **OK**



The appropriate port prototype not exists? You like to connect the port to a new port prototype?

Therefore select the service component port and use the button **[connect to new port...]** to open the Connect To New Port Prototype Assistant.

1. Select the **Component** to be connected with the selected port prototypes.
   Only those components are displayed which component type is part of the DaVinci Developer workspace.
2. Define the name of the **Port Prototype to be created**. By default the same name is used. Therefore click in the **Port Prototype to be created** column.
   If a server port is selected, you can additionally select, if server runnables should created.
3. Click **Finish**

What happens?

The DaVinci Developer will open in background, a new port prototype (Application and Service port prototypes are possible) will be created within selected application component. The workspace will be saved and the DaVinci Configurator project will be reloaded. The new created port is now available within the DaVinci Configurator project and will be mapped automatically to the service component.

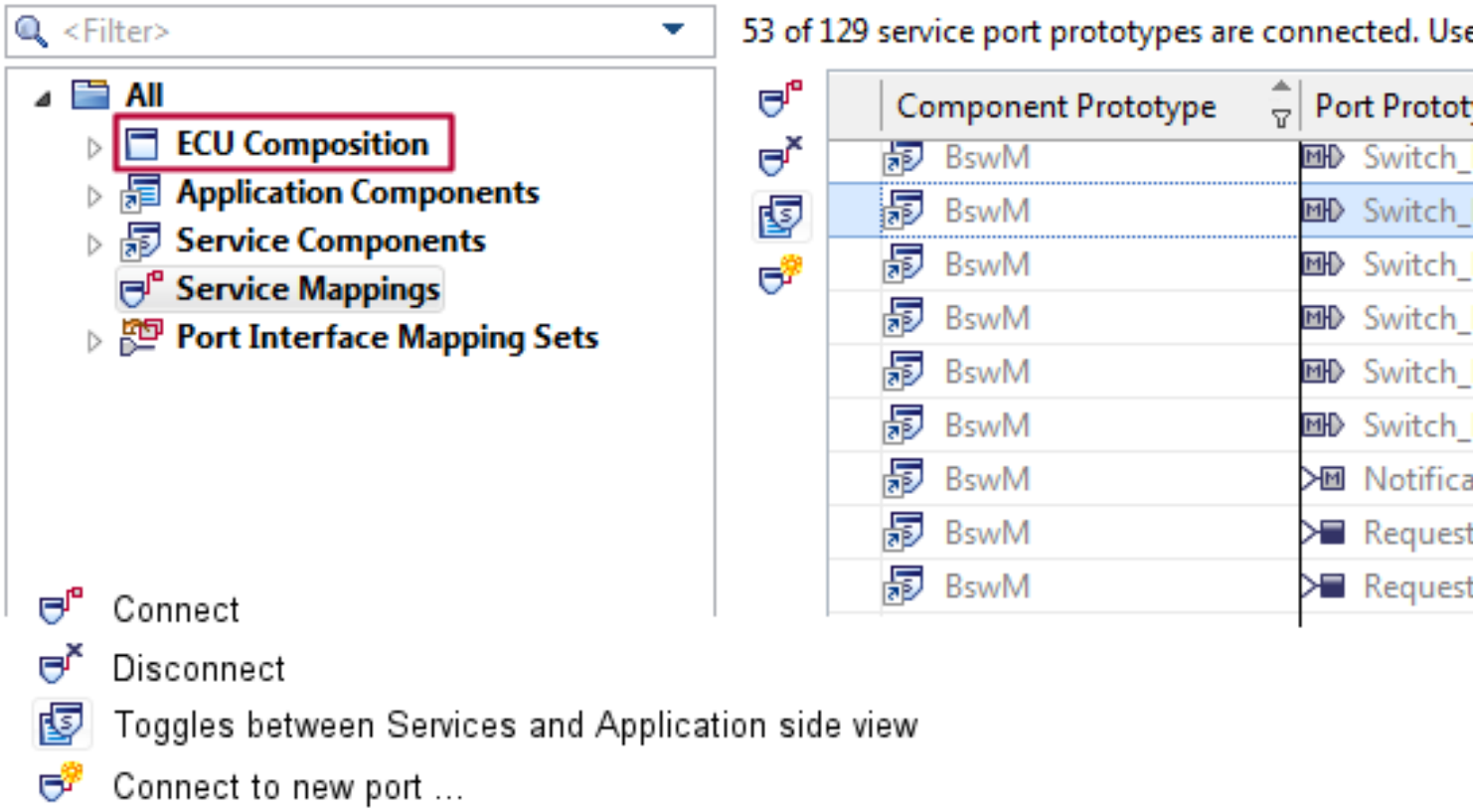
1

6.6.2 Service Mapping (overview)

Here you can do the complete service mapping. Connect or disconnect services to applications, toggle between services view and application view or connect to new port.

Here you also get a complete overview of all connections.

Show illustration



6.7 Add Data Mapping

You have already performed Data Mapping within the DaVinci Developer?

Yes? Skip this section and go on with Add Memory Mapping on page 64.

Cross Reference

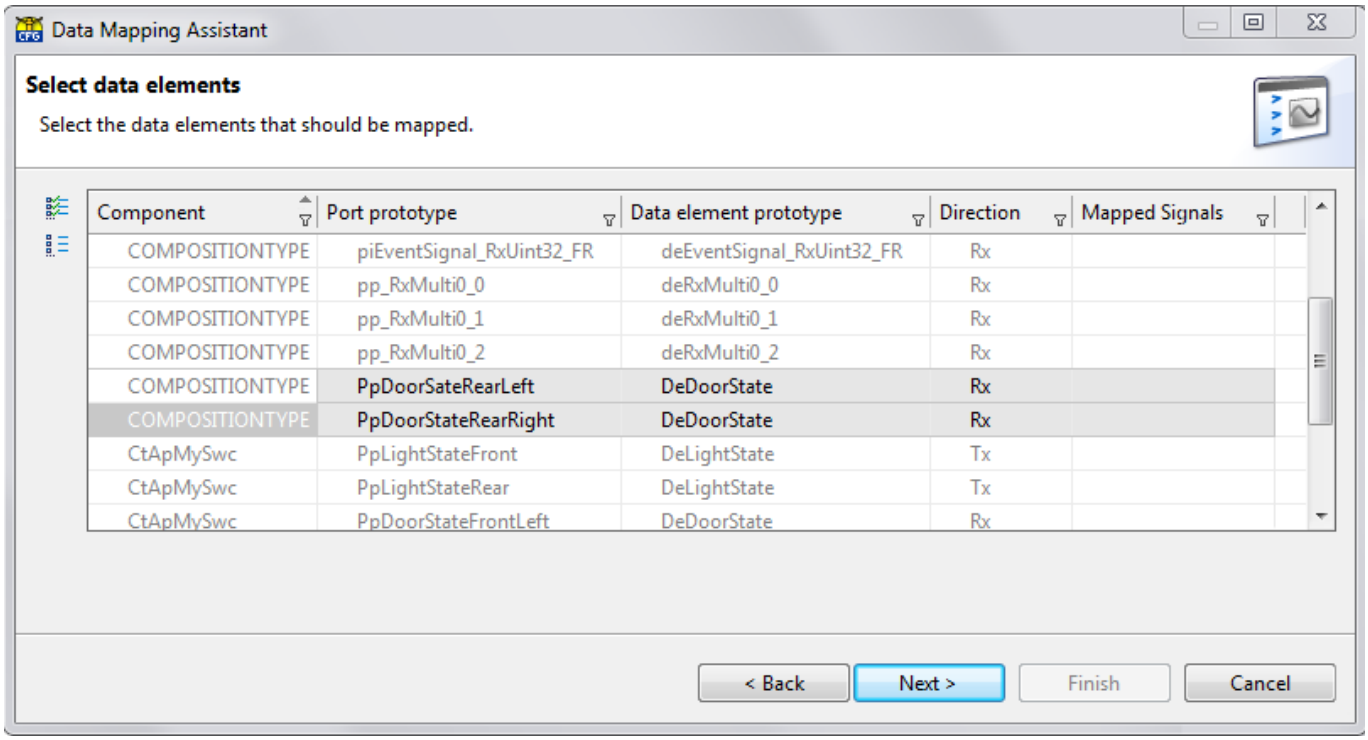
See section Perform Data Mapping within DaVinci Developer or DaVinci Configurator? on page 56

No? Start Assistant via Add Data Mapping[1] and select mapping direction, i.e. choose whether **Signals** or **Data Elements** should be mapped.

1 Add Data Mapping

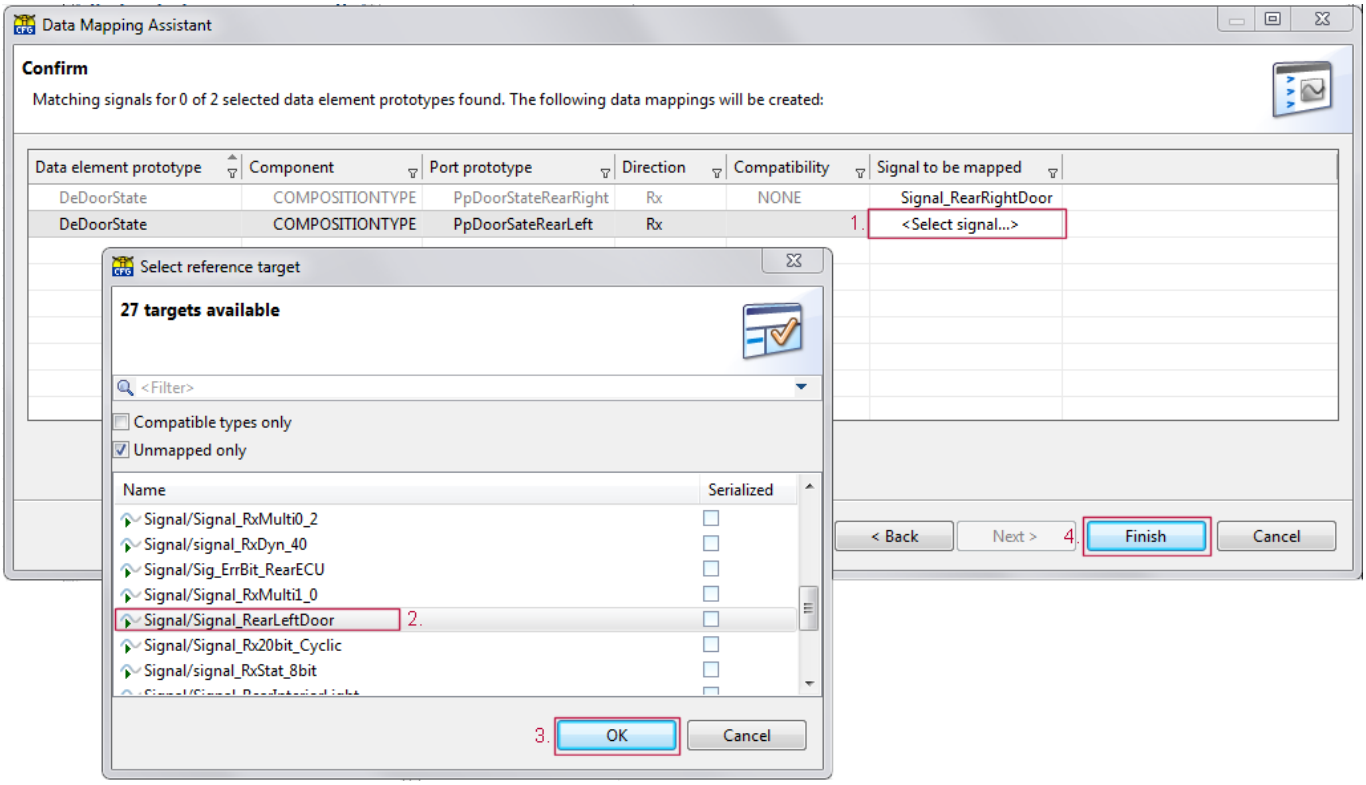
Choose **Find matching signals for the data elements** to assign your signals based on a selected data element and go on with **[Next]**

Now you can see a list of data elements. The empty row **Mapped Signals** shows that there is no assignment between data elements and messages/signals from the ECU Extract of System Description.



Select the Data element prototypes you want to map and go on with **[Next]**.

Now you can see if matched signals are found automatically. If there are no entry in row **Signal to be mapped**, you have to define the signal manually.



1. Click **<Select signal...>** to open the **Select reference target** window.
2. Select the signal you want to map.
3. Confirm with **[OK]**.

Repeat these steps until all data elements you have selected for mapping are mapped to a signal.

4. Close Data Mapping with **[Finish]**.

6.8 Add Memory Mapping

You have to map all per-instance memory objects of the application components to NV memory blocks. Therefore use the Memory Mapping Assistant[1].

Note

A per-instance memory object is mappable if it is referenced by a service need object. The definition of per-instance memory objects or service needs is part of the component type definition and not yet editable by DaVinci Configurator Pro.

1. Start Assistant, choose **Use Case** and go on with **[Next]**
2. Select component prototype and go on with **[Next]**
3. Select per-instance memory objects
   Choosing Use Case Find exisiting NV memory blocks?

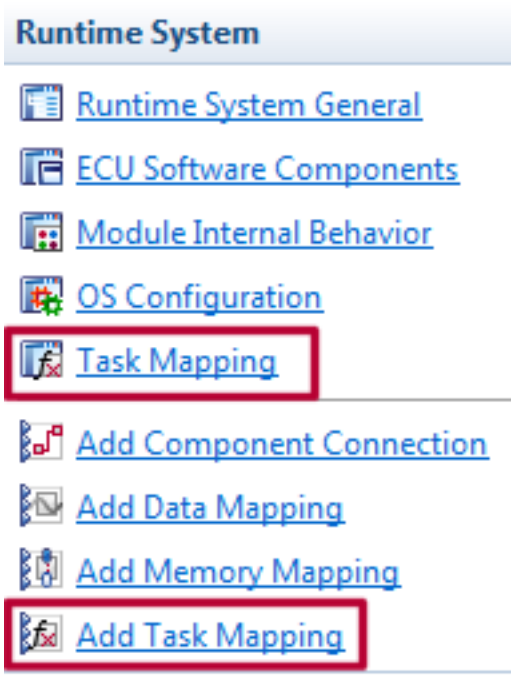
1 Add Memory Mapping

Go on with **[Next]**, Check the memory mappings and confirm with [Finish]
Choosing Use Case Create new NV memory blocks?
Confirm with **[Finish]**

6.9 Add Task Mapping

You have to map all runnable entities and schedulable entities (e.g. Main Functions of the BSW modules) to a task. There are two ways to perform the task mapping.

Where to activate the two ways of task mapping



Either the Task Mapping Assistant[1] or the Task Mapping view  .

Our advice is to use the assistant for the first mapping and then switch over to the Task Mapping view  to do the fine tuning.

Note

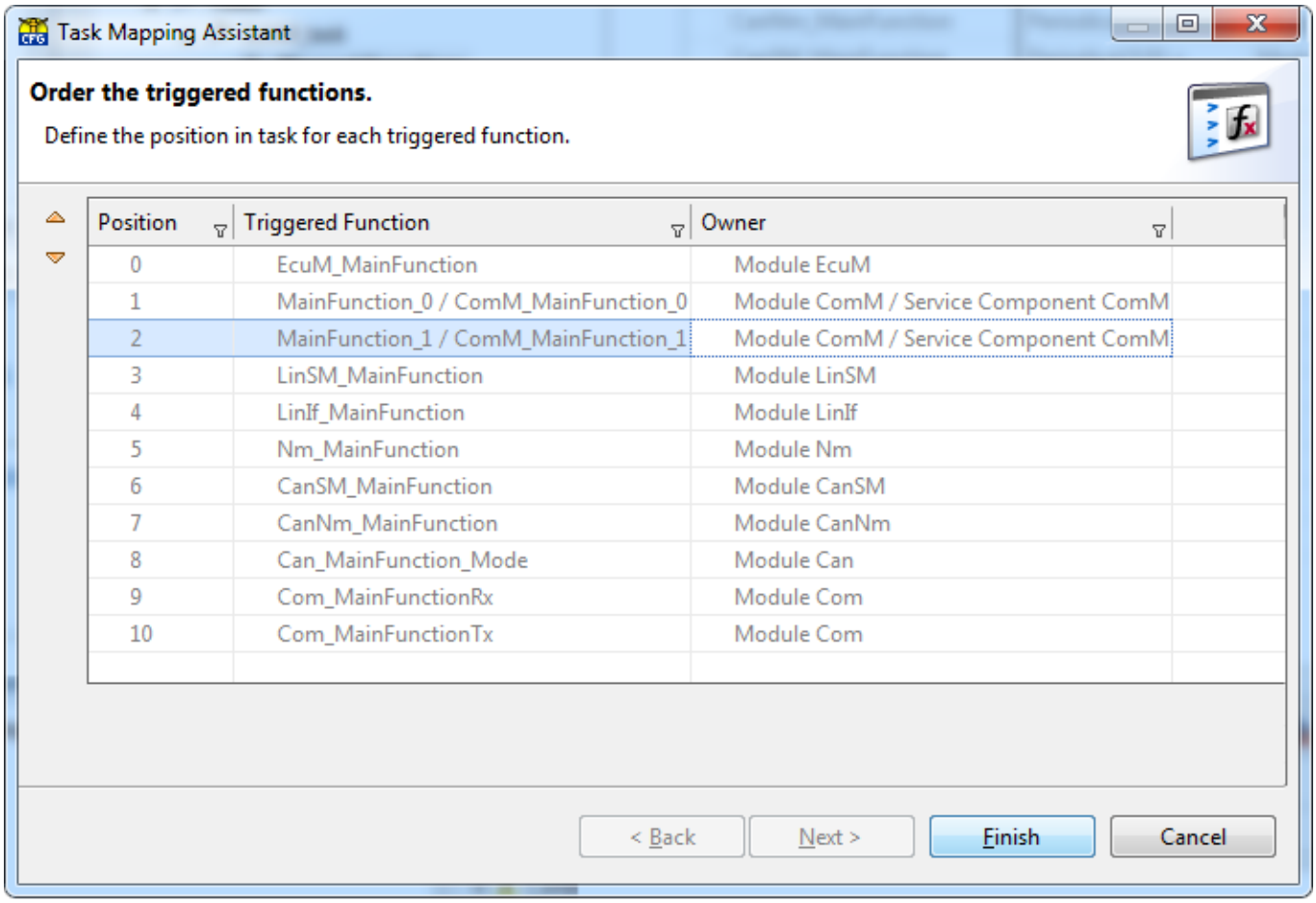
A runnable has to be mapped to a task if it is not re-entrant but could be called re-entrantly during system operation.

6.9.1 Task Mapping Assistant

1. Start Assistant, select all Triggered Functions with Function Trigger On Init and go on with **[Next]**
2. Select Init_Task and go on with **[Next]**

1  Add Task Mapping

3. Define the position in task for each triggered function and confirm with **[Finish]**



4. Repeat the steps and select all **Main Functions** of **Service Components** and **Modules** and map them to **Main_Task**.
5. Repeat the steps and select all **Runnables** of your **Application Components** and map them to Rte[1]**_Task**.

Note

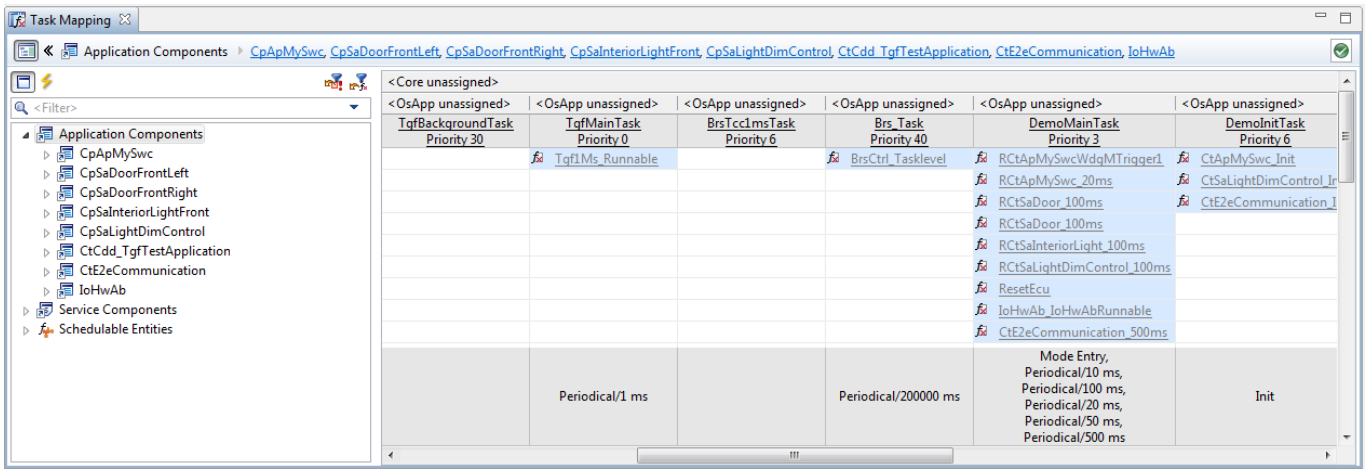
Runnables with Function Trigger **On Operation Invocation** are normally not needed to be mapped to a task, as they result in direct function calls.

6.9.2 Task Mapping

The task mapping is one view where you can

[1](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

map runnable entities per drag and drop, move mapped runnable entities from one task to another, filter the list on the left side for mapped mandatory or unmapped runnable entities, you get information of the mapped runnable entities concerning their triggers, etc.



More details about the features of this dialog you find in the **Online Help** of the DaVinci Configurator Pro.

Mappings finished? Validation successfully? Then go on with project generation!

7 STEP7 Code Generation

Now you can go on with code generation.

Expert Knowledge

You need to know more? See section Generation on page 124.

7.1 Start Custom Workflow Steps

Open Project settings via Project Setting icon [1] from DaVinci Configurator Pro toolbar.

Available?
Start Custom Workflow Step via Execute Custom Workflow icon[2].

Not available?
Create a Custom Workflow Step with parameter **RTE: -g i**.

Cross Reference

How to create Custom Workflow Step? See section STEP2 Define Project Settings on page 33.

7.2 Start Code Generation

1. Open the Generation dialog via Generate button [3]

Note

The activated generation steps depend on the project settings. If you want to take over your selection to the project settings press **[Change project settings]**.

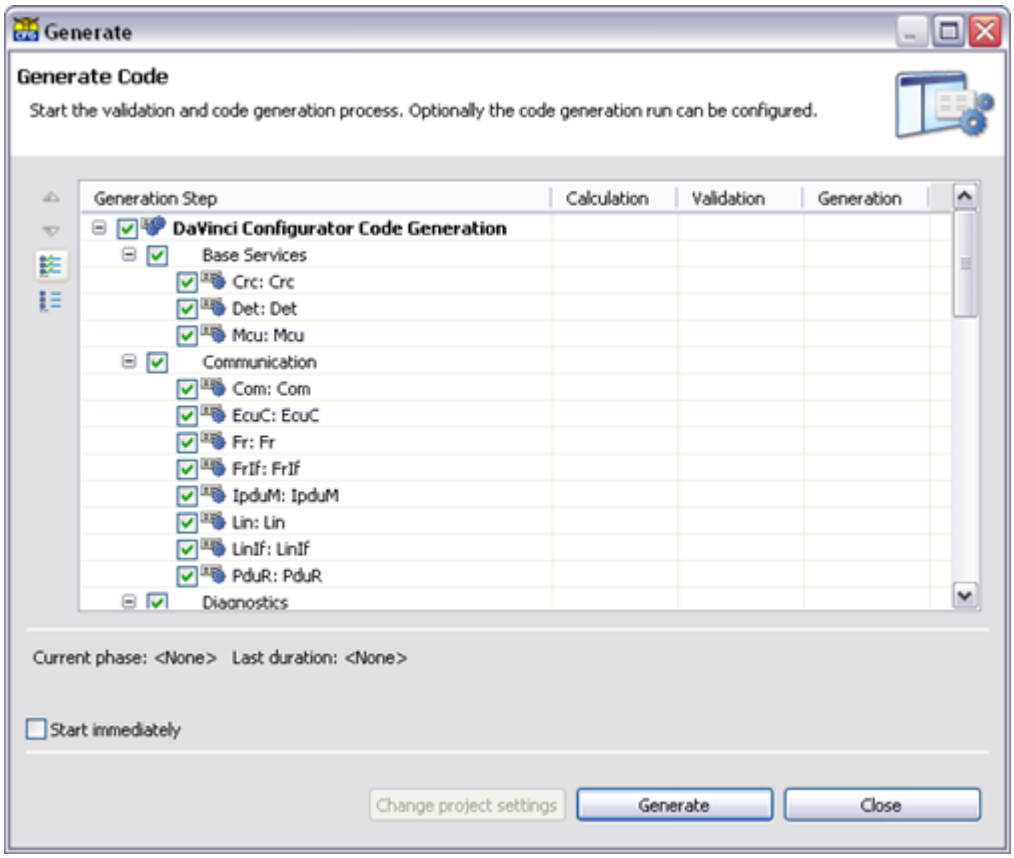
---

1 

2 
3 4

4

2. Start Generation Process via **[Generate]**
   If you select **Start immediately** the generation starts immediately after opening this dialog.



Note

Before the **Generation** process starts, **Calculation** and **Validation** has to be finished successfully.
If code generation canceled because of calculation and validation errors, start solving actions at the validation window of the DaVinci Configurator Pro (see section STEP3 Validation on page 37).

7.3 Generation Process finished!

Generation Process finished and all generation steps are marked with Phase finished successfully ?

Note

The DaVinci Configurator Pro stores all generated *.c and *.h file in the modules files folder, given during project setup (the default is **<ProjectFolder>\Appl\GenData**).

Go on with implementing your runnables using the runnable skeletons created during the generation process.

8 STEP8 Add Runnable Code

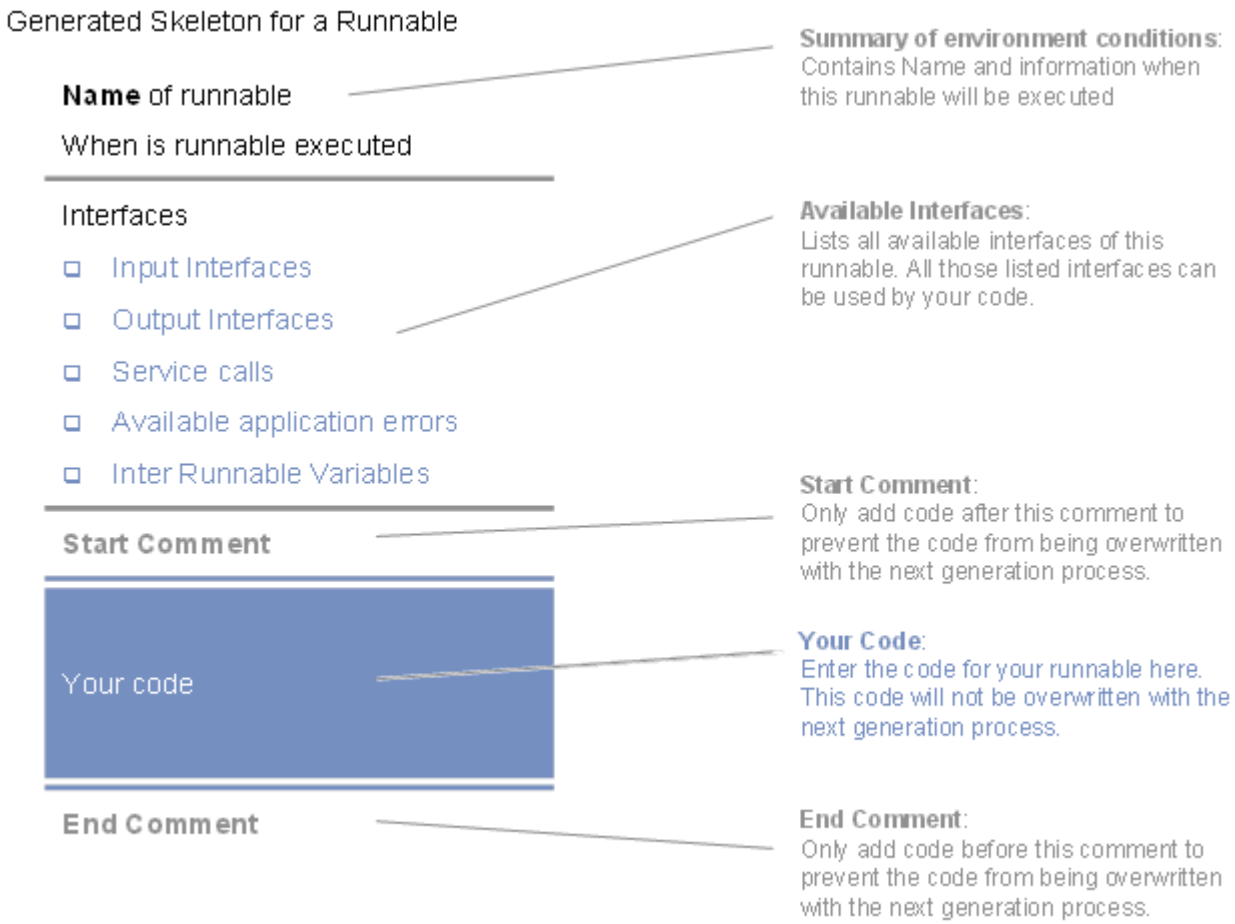
If configured, the DaVinci Configurator Pro code generator generates the skeletons for your runnables into the software component template files. Now you have to implement your code to the runnable skeletons.

Expert Knowledge

You need to know more? See section Runnable Code on page 125

8.1 Component Template

The software component templates contain all runnables of the according software components. The RTE Template Generator generates skeletons for the runnables. The structure of those skeletons looks as shown in the illustration below.



8.2 Implement Code

Implement your code using the runnable skeletons. Make sure to enter your code between the start and end comment. Only this section is protected against modification by a further generation process.

```
FUNC(void, RTE_AP_MYSWC_APPL_CODE) MySCW_Code(void)
{
/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!          << Start of runnable implementation >>          DO NOT CHANGE THIS COMMENT!
 * Symbol: MySCW_Code
 *********************************************************************************************************************/
Boolean dataLeft, dataRight, dataLeftRear, dataRightRear;

Std_ReturnType value;


Rte_Read_FrontLeftDoorState_OpenClose(&dataLeft);
Rte_Read_FrontRightDoorState_OpenClose(&dataRight);
Rte_Read_RearInformation_RearLeftDoor(&dataLeftRear);
Rte_Read_RearInformation_RearRightDoor(&dataRightRear);


if (dataLeft || dataRight || dataLeftRear || dataRightRear)
{
    Rte_Call_UR_ComMUser_CAN_GetRequestedComMode(&value);
    if (value != COMM_FULL_COMMUNICATION)
    {
        Rte_Call_UR_ComMUser_CAN_RequestComMode(COMM_FULL_COMMUNICATION);
    }

    Rte_Write_FrontLightState_OnOff(TRUE);
    Rte_Write_RearLightControl_SwitchRearInteriorLight(TRUE);
}
else
{
    Rte_Write_FrontLightState_OnOff(FALSE);
    Rte_Write_RearLightControl_SwitchRearInteriorLight(FALSE);

    Rte_Call_UR_ComMUser_CAN_GetRequestedComMode(&value);
    if ((value != COMM_NO_COMMUNICATION) && (lightDimControl == 0))
    {
        Rte_Call_UR_ComMUser_CAN_RequestComMode(COMM_NO_COMMUNICATION);
    }
}
/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!          << End of runnable implementation >>          DO NOT CHANGE THIS COMMENT!
 *********************************************************************************************************************/
}
```

All other sections like execution information or interfaces can be changed depending on your settings in the DaVinci Developer. If access to e.g. a port interface is missing, go back to the DaVinci Developer and adapt your configuration, regenerate and then you can use this port interface. There is no other way to do it properly.

Info

A backup (*.bak) file will be generated before a component template C file is modified to make sure that your previous version is not deleted in case of any generation problem caused by non-predictable reasons.

And there is a backup section at the end of the template file to rescue the code of runnables that have been deleted by the DaVinci Developer.

9 STEP9 Compile, Link And Test Your Project

Now you can compile, link and test to get your executable for the download to your hardware.

Expert Knowledge

You need to know more? See section Compile and Link on page 127.

9.1 Finish your project with compiling and linking

1. Compile and Link your code.
   You can start compilation of your code from the vVIRTUALtarget tool. Enable the checkbox Build Software (DLL) in the VTT tool and select  **[Build]**. You can also open the Visual Studio solution and start the build process by **Build Solution (F7)**.
2. Test your code
   A proper compile and Link process is just the beginning. The final step is to test what you have done until now. The ideal tester would be CANoe from Vector Informatik.

To test your application with CANoe, the following steps are necessary:

> Create a new CANoe configuration
> Import the communication matrix (e.g. *.dbc file)
> Right-click on the ECU to be simulated, and click on **Configuration**
> Open the **Components** tab, add another node layer DLL and select the *.dll file, you have created with the Microsoft Visual Studio solution in the previous steps.
  This completes the basic setup for testing your application. See CANoe references for further information.
> Start the simulation.
> Have a look at the trace window and observe bus communication on your virtual ECU.

3. Debugging the application
   When the simulation in CANoe is started, switch to Microsoft Visual Studio and select **Debug | Attach to process ...**. Select **RuntimeKernel.exe** and make sure that the code type (Attach to) is set to Native code. Now you can debug your code. For example, try to set a breakpoint in a cyclic runnable!

9.2 No error frames? Congratulations, that's it!

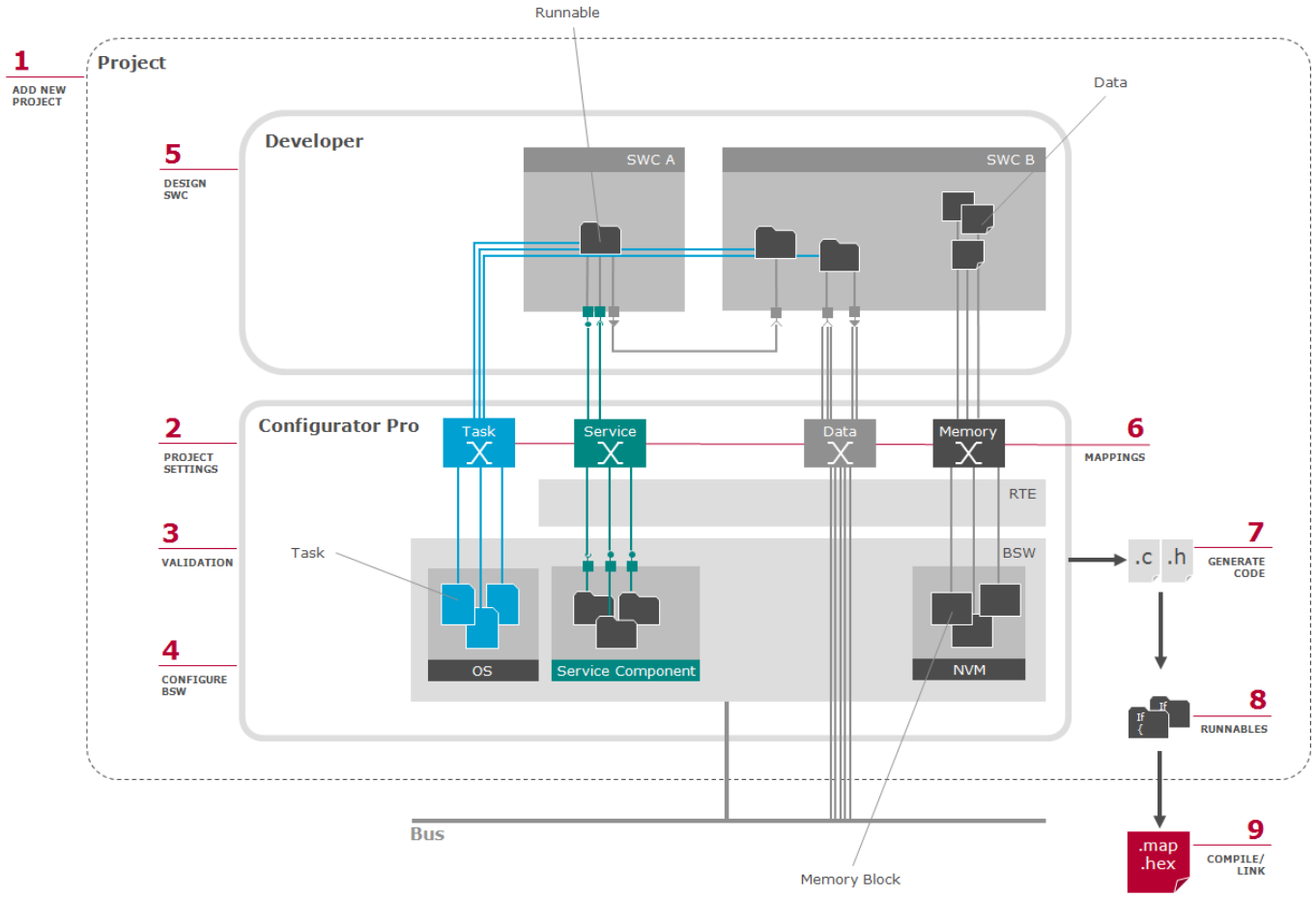
The basic step is done, the GM SLP2 software is basically working together with your application.

That is the base you can start from to optimize your system. Refer to Technical References of the BSW modules for more detailed information.

Cross Reference

You need additional information? See section Additional Information on page 128.

II Concept


Runnable
1
ADD NEW PROJECT
Project
Data
5
DESIGN SWC
Developer
SWC A
SWC B
2
PROJECT SETTINGS
Configurator Pro
Task
Service
Data
Memory
6
MAPPINGS
RTE
3
VALIDATION
Task
BSW
.c
.h
7
GENERATE CODE
4
CONFIGURE BSW
OS
Service Component
NVM
8
RUNNABLES
Bus
.map
.hex
9
COMPILE/ LINK
Memory Block

1 General Overview

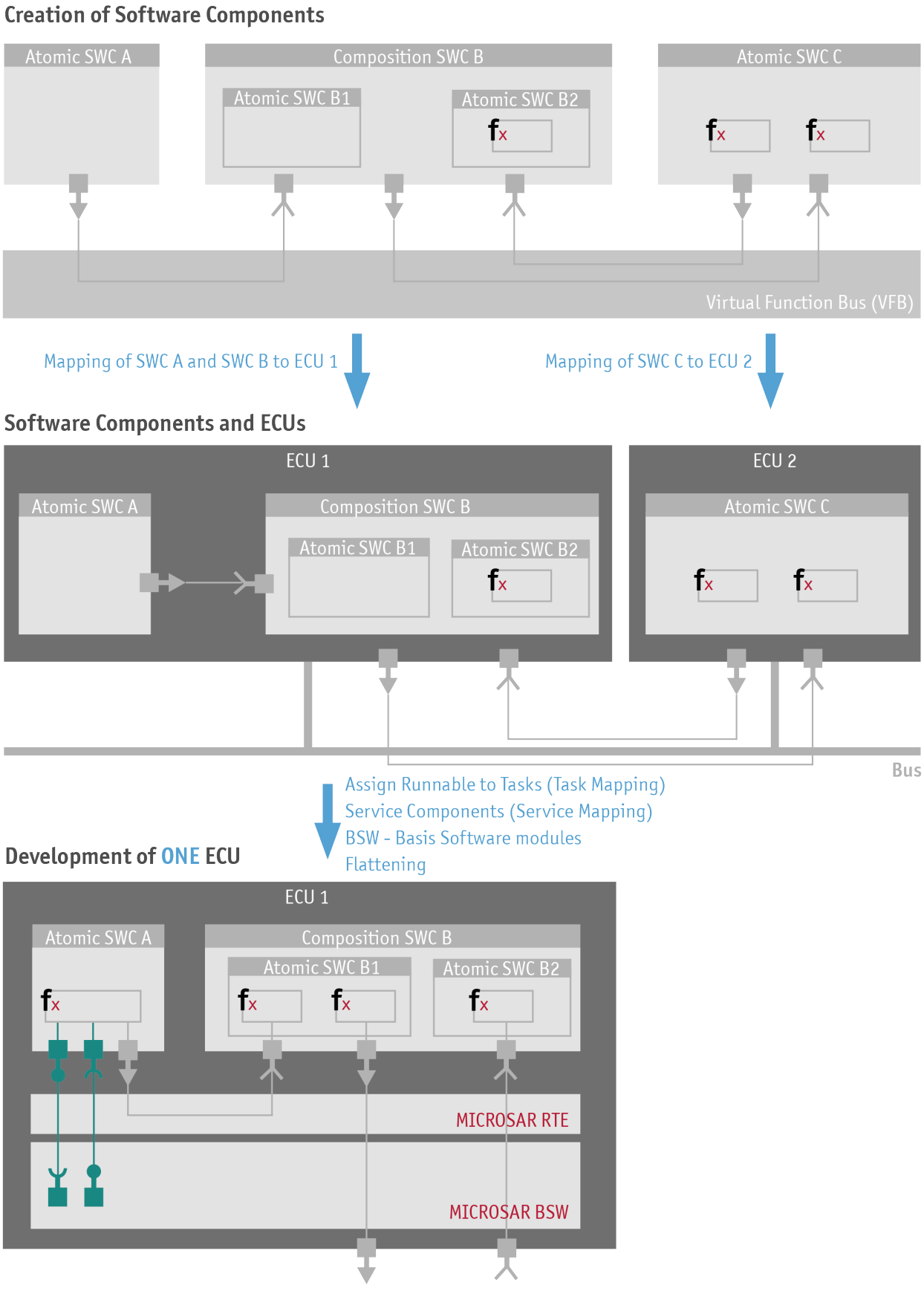
The following illustration shows the basic idea behind AUTOSAR and can be divided up into 3 parts, upper, middle and lower part, named as

- Creation of Software Components
- Software Components and ECUs
- Development of ONE ECU

The upper part deals with software components, runnables, ports and connections. It is the design level of functions and applications.

During the transition to the middle part, the software components are assigned to ECUs. Software components that are assigned to the same ECU can communicate internally, via so-called internal connections. Software components, that communicate with each other and that are mapped to different ECUs communicate via so-called external connection, i.e. via a real bus system.

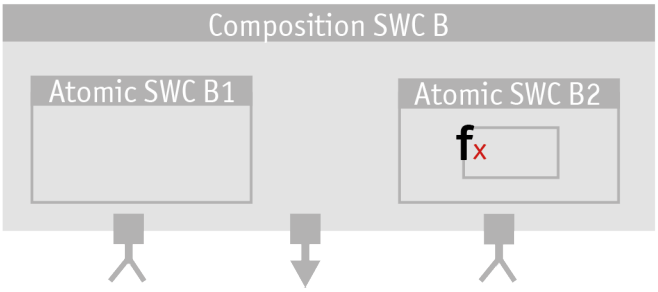
In the last transition towards the ECU sight, the BSW is put in-between the software components and the hardware. And this is the starting point for the ECU development.


Creation of Software Components
Atomic SWC A
Composition SWC B
Atomic SWC B1
Atomic SWC B2
Atomic SWC C
Virtual Function Bus (VFB)
Mapping of SWC A and SWC B to ECU 1
Mapping of SWC C to ECU 2
Software Components and ECUs
ECU 1
ECU 2
Bus
Assign Runnable to Tasks (Task Mapping)
Service Components (Service Mapping)
BSW - Basis Software modules
Flattening
Development of ONE ECU
MICROSAR RTE
MICROSAR BSW

The following sub chapters explain all the means that belong to AUTOSAR and that make it work.

1.1 Software Component

Software Components are described by their SWC file - a file in XML format. By definition SWC files are independent from any ECU except for the sensor and actuator software components. Those software components are dependent on the sensors and actuators attached to a specific ECU.



There are several kinds of software components

1.1.1 Atomic components

- Application
- Sensor Actuator
- Calibration
- I/O Hardware Abstraction
- Complex Driver
- Application (End-to-end Protection)
- Non-Volatile Memory Block
- Service components

1.1.2 Compositions

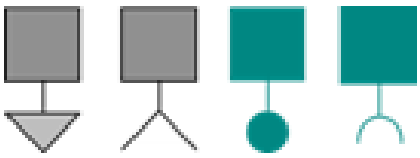
Use compositions to group software components.

1.2 Runnables

A runnable entity, runnable for short, is a C function that carries your code. It depends on the configuration of the runnable when it is called and what data your code is allowed to access within the runnable. Runnables are triggered by the RTE.

1.3 Ports

Via ports, a software component can communicate with its outside world. This outside world can be another software component within the same ECU or a software component within another ECU. There are several kinds of ports:



1.3.1 Application Port Interfaces

To communicate between different SWCs

- Sender port
- Receiver port
- Client port
- Server port
- Calibration port
- Mode port
- Interface to non-volatile data

1.3.2 Service Port Interfaces

To communicate between BSW and SWCs

- Sender port
- Receiver port
- Client port
- Server port
- Calibration port
- Mode port
- Interface to non-volatile data
- NV Data Interface

1.4 Data Element Types

Data elements types determine the kind of data that can pass a port. There are predefined data element types and you can define data elements on your own.

1.5 Connections

To define which software components communicate with each other, their ports have to be connected via so-called connectors. The connectors realize the sender/receiver and client/server communication between the software components. A connector can only be drawn between two ports, if their port interfaces are compatible.

1.6 RTE

The RTE is the interface between the SWCs and the BSW. In essence the RTE

- controls the execution of the Runnables (your code),
- controls access of Runnables to the basic software modules,
- controls the access of Runnables to Services of the BSW,
- knows about external and internal transmission of data and
- provides data consistency

1.7 BSW – Basic Software Modules

The BSW contains configurable modules that offer services to the SWCs dealing with:

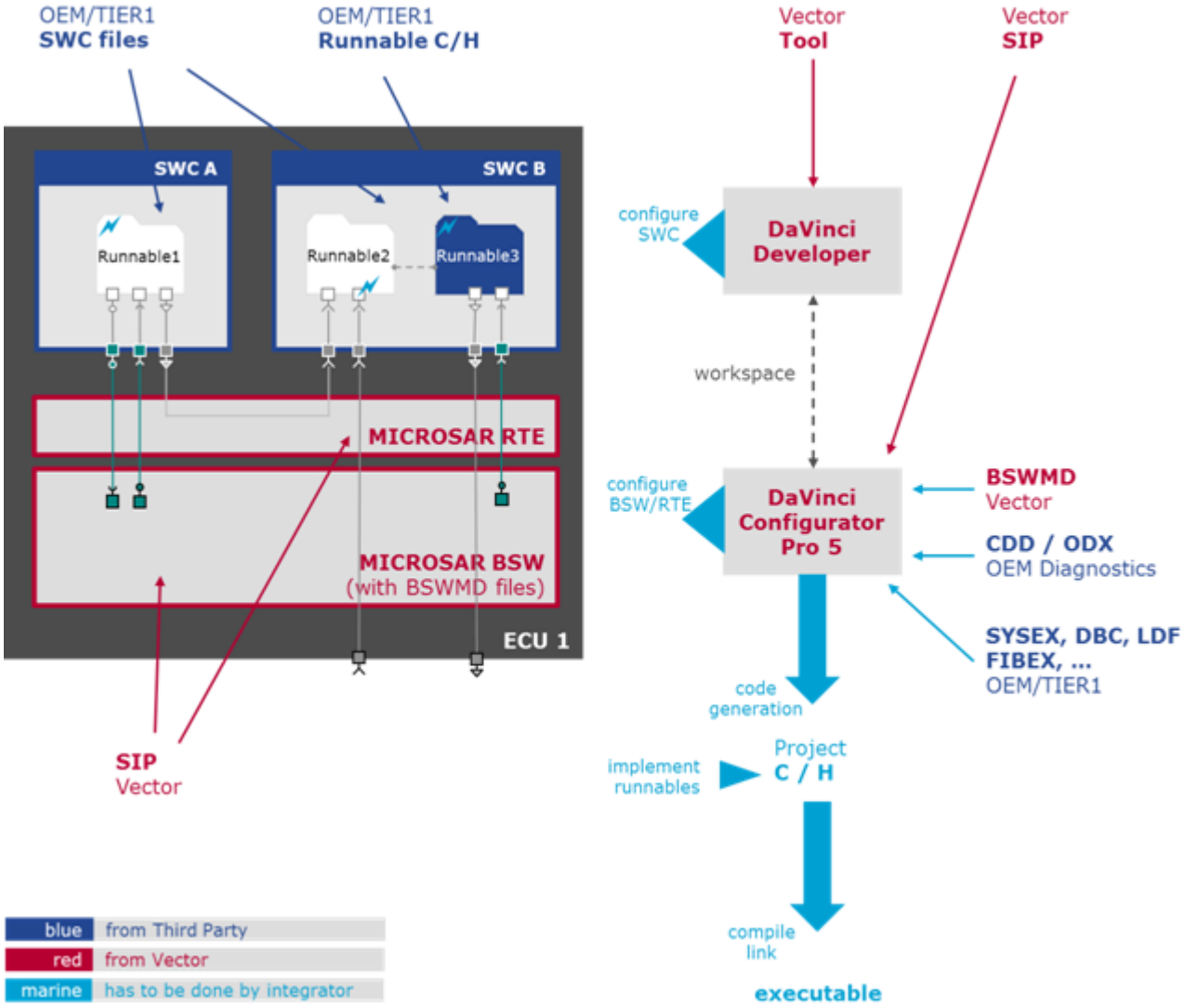
- Communication
- Diagnostics
- Mode management
- Memory management
- Network management

1.8 Software, Tools and Files

When setting up a new project using the DaVinci Configurator Pro, many things that are not necessary to know for your basic work with the tool happen in the background. But sometimes it might be necessary to know more. Then this part of the document is the place to get a deeper insight.

This chapter gives you an overview what kind of files you need to set up a new project, where the files normally come from and how to work with the DaVinci Configurator Pro and, if necessary for software component design, with DaVinci Developer. Get familiar with what is installed where and how to work with it and to know, how to update if necessary.

Starting point is the development of a single ECU. The focus is set on the MICROSAR Solution and where all the necessary software and files do come from.



To build up a project according AUTOSAR with MICROSAR from Vector you need the following tool(s), software and files:

> **Software Components (SWC) and Runnables**
Software components are created by the OEM or the TIER1. A software component can be an SWC file, where you have to perform the task mapping and decide about runnables and the code they should contain. Or the software component is delivered ready-made, together with C and H files. Then you only have to map it to a task and compile and link the files together with your project.

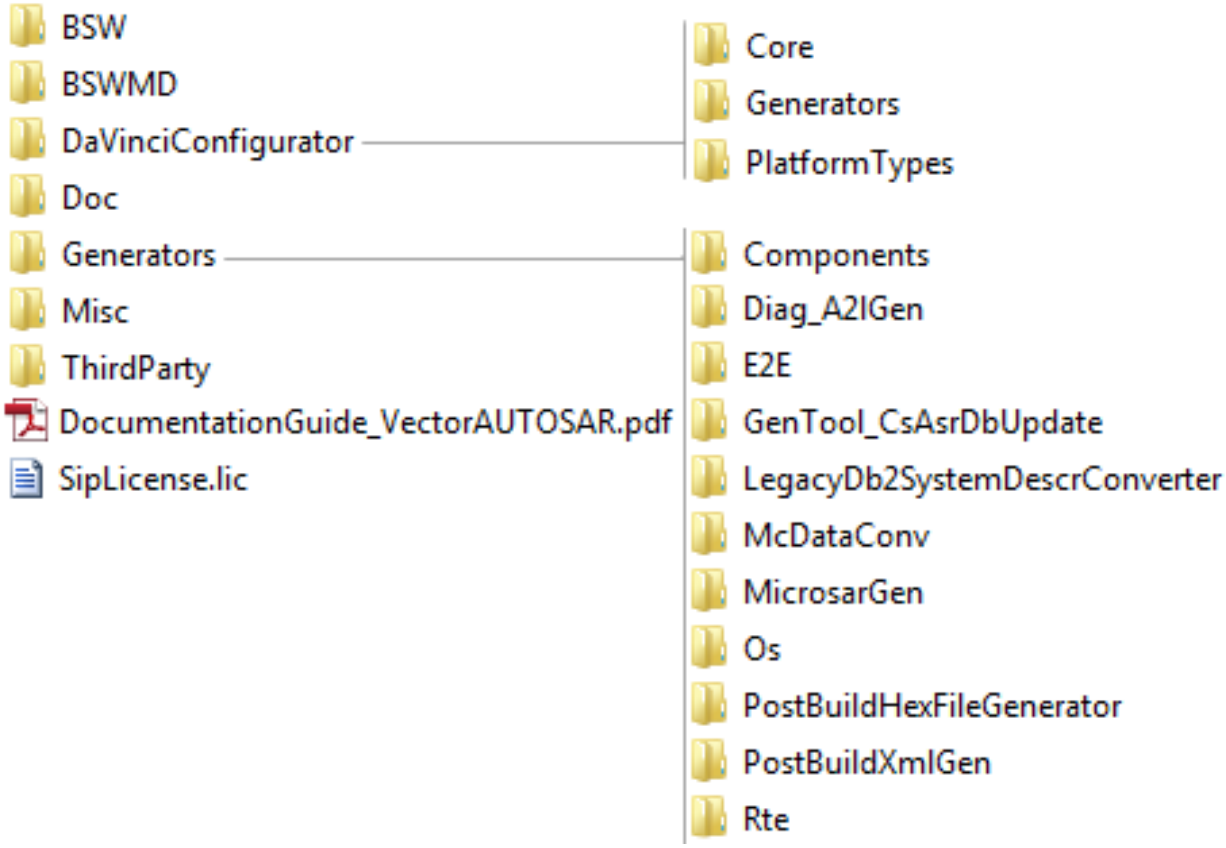
> **DaVinci Developer (Vector Tool, has to be ordered separately - NOT included within the SIP)**
The DaVinci Developer is the Vector Tool to create and configure Software Components. Via a graphical view you can draw software components, add ports, draw connections, etc. DaVinci Developer and DaVinci Configurator Pro exchange information via the DaVinci Developer Workspace.

> **BSW and RTE (included in SIP)**
The BSW modules together with their BSWMD files (dependent on your order) and the RTE are part of the Vector delivery that is called SIP. **SIP** stands for **Software Integration Package**. A SIP is a ready-made software from Vector, already tailored and tested for your hardware, derivative, compiler, compiler version (as you filled-out the questionnaire) and can be put to work within a few steps (see step by step description).

> **DaVinci Configurator Pro (included in SIP)**
The DaVinci Configurator Pro is the Vector Tool to configure the BSW and the RTE. It is part of the SIP delivery. It contains a comfort editor to configure the BSW cluster-oriented, a powerful validation concept to check your settings against constraints and it also offers to directly edit AUTOSAR parameters.

1.9 Structure of the SIP Folder

The screenshot below shows the file structure that is created by installation of the SIP.



> **BSW**
Contains the code files (*.C and *.H) files for each Basic Software Module (BSW).

- **BSWMD**
  Contains the BSWMD files for each Basic Software Module, which include necessary information for DaVinci Configurator Pro.
- **DaVinciConfigurator**
  Contains the DaVinci Configurator Pro tool itself. This is also the place where your tool is updated when an update is available.
- **Doc**
  Contains SIP relevant documentation e.g. Technical References, User Manuals, Startups (containing Release Notes) and Application Notes.
- **Generators**
  Contains Generators and additional tools
- **Misc**
  Contains SIP-specific additional data
- **ThirdParty**
  Contains MCAL-specific data

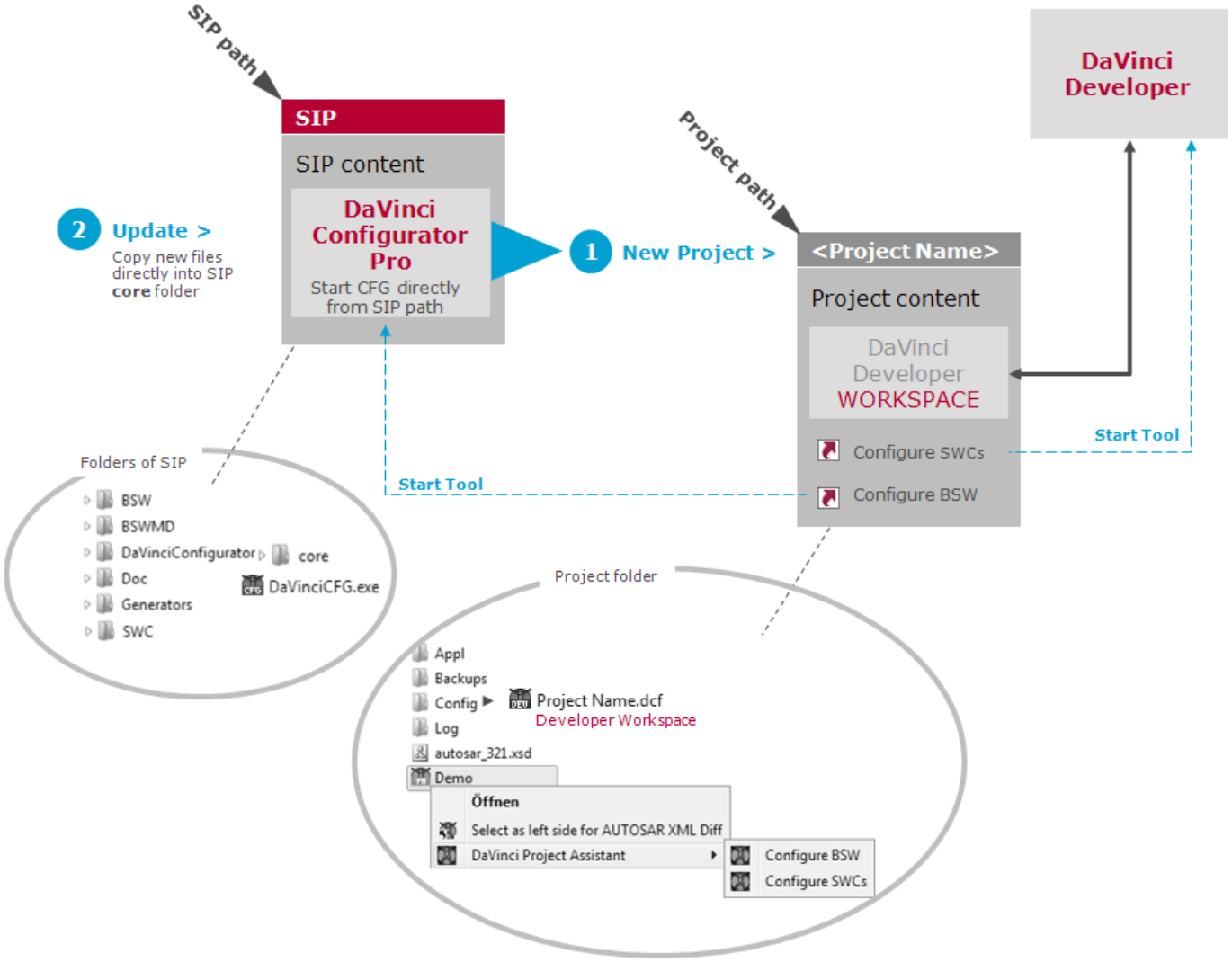
Note

Each MCAL is listed within a separate folder (<MCAL Short Name>).

This folder includes:

- **\Supply**
  Location of the 3rd party MCAL delivery in its original structure
- **\VectorIntegration** (optional)
  Location of scripts for the integration into the MICROSAR SIP
- Legal notes (optional)

See **TechnicalReference_3rdParty-MCAL-Integration.pdf** for detailed information.



Vector MICROSAR for AUTOSAR 4 now has one major tool – the DaVinci Configurator Pro. DaVinci Configurator Pro is the starting point, coming with the SIP. To start the tool, use the file **DaVinciCFG.exe** located in the **CORE** folder of the DaVinci Configurator folder.

2 Set-Up New Project

DaVinci Configurator Pro offers a Project Assistant to set up that helps you setting up a new project file in just a view steps. The assistant guides you through several windows to enter necessary project information.

The DaVinci Configurator Pro creates your new project to the defined folder. The project folder contains:

- **Appl** folder for generated files and source files.
- **Config** folder for software components, BSWMD files, ECUC files, System description and DaVinci Developer workspace
- **Backup** folder to store older DaVinci Developer workspaces
- **Log** folder
- Access to DaVinci Developer and DaVinci Configurator Pro via context menu on the *.**dpa file**.

Your development project consists of a folder structure on your disk, where all the configuration files, template files, etc. are generated to and where you implement your runnables using the empty runnable skeletons. Then you compile and link and get your executable that you can download to your hardware.

The DaVinci project file (DPA) is the central file which references all the related folders and files. It also references the SIP, which is used for the project.

2.1 DaVinci Configurator

As already mentioned the DaVinci Configurator Prois the major tool for Vector MICROSAR for AUTOSAR 4.

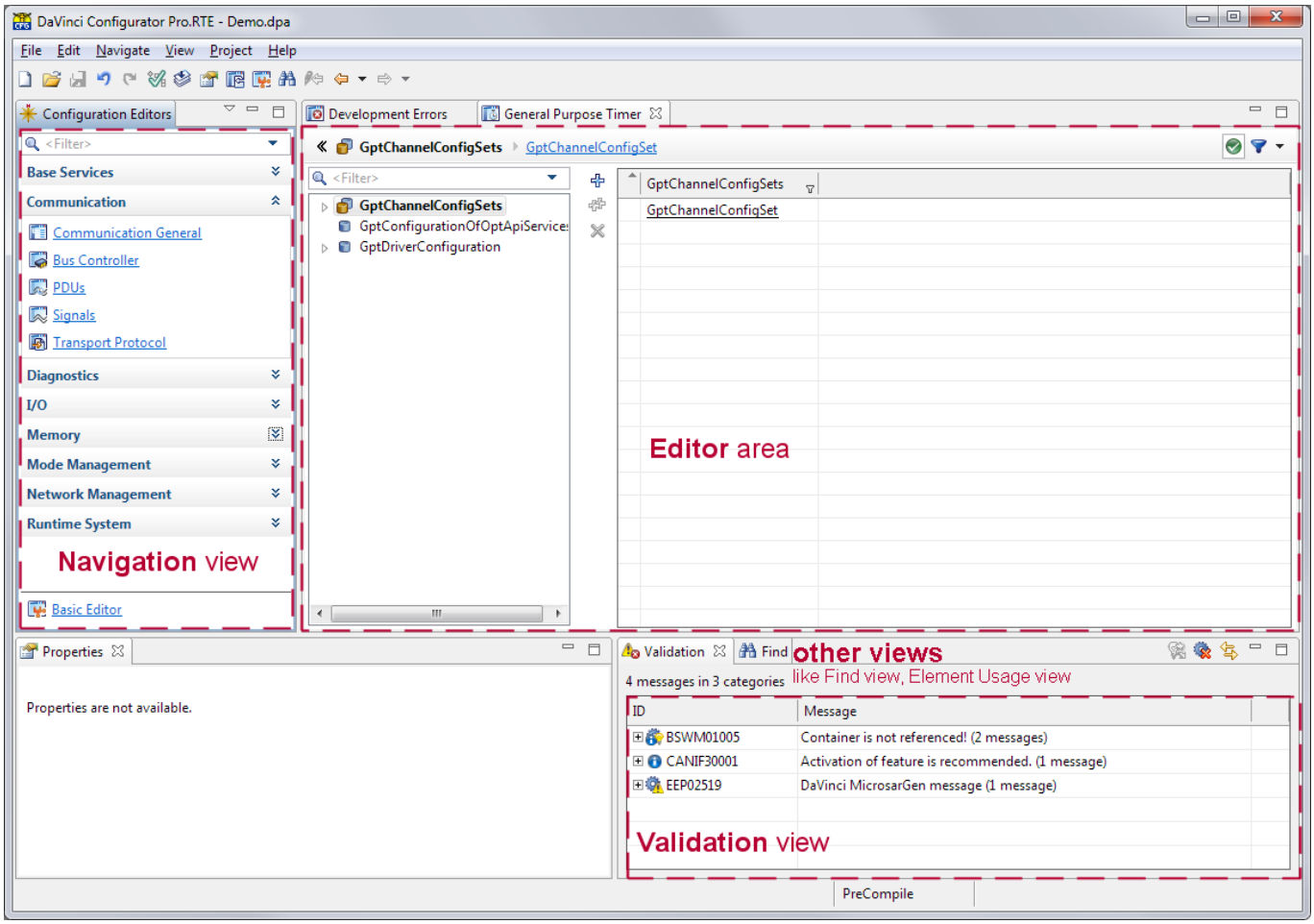
2.1.1 The Main Window of DaVinci Configurator Pro

The illustration below shows the main appearance of the DaVinci Configurator Pro. It consists of 3 main areas or views. The positions of the views can be defined manually and depend on your habits in handling with tools.

- Navigation view
- Editor Area
- Validation, Output and Properties view

Cross Reference

For more information about simple tool handling and icons refer to the help of the DaVinci Configurator Pro.



Navigation view
The Navigation view displays all available editors and assistants sorted by domain. What you select in the Navigation view will be displayed in the Editor view.

Editor area
The Editor area is the place to configure the modules or whatever you have selected in the Navigation view.

Validation view
The Validation view displays the overall list of validation messages.The messages are grouped by their ID. You may expand such a group to display all message of the same ID. By expanding an individual message, the related items are displayed as well as the proposed solutions, the so-called solving actions.

Using according commands in the shortcut menu you can navigate to the editors displaying the related items, or execute one of the proposed solutions for solving the problem.

You can use the Validation view with the following buttons:

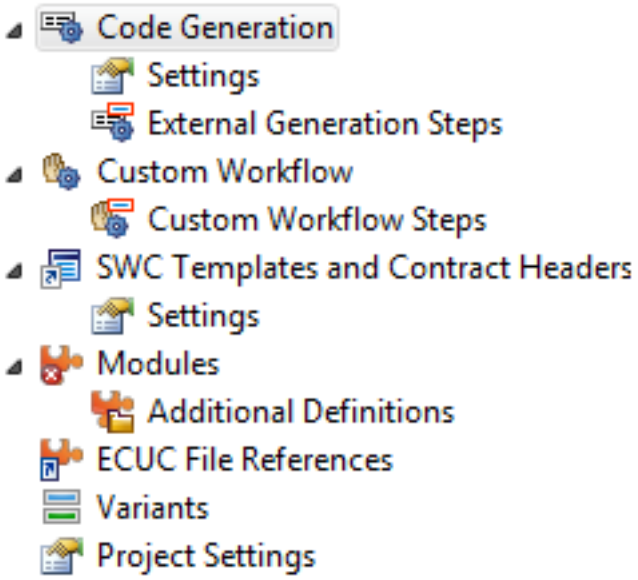
- Solve All [1]
  Calls the default solving action of all messages that provide a default solving action.
- Clear On-Demand Validation Messages [2]
  Deletes all validation messages issued by the external generators during the on-demand validation.
- Link with Editor [3]
  Filters the Validation View to display only messages related to the currently focused editor.

Properties view
The Properties view displays details of the object (parameter or container) selected in an editor. The view contains the following tabs:

- **Definition**: Displays the definition of the selected object
- **Status**: Displays information about the parameter state
- **Description**: Displays the description of the parameter definition

Project Settings

This Overview shows all necessary project data and enables the configuration of customer workflow steps, the definition of external configuration steps and the activation of additional BSW modules.



Code Generation: Configure the code generation sequence

Settings: The code generation settings can be changed within this section

External Generation Steps: Define and execute a sequence of custom workflow step

Customer Workflow: Configure and execute a sequence of **custom workflow steps**

SWC Template and Contract Headers:Definition of additional pathes to BSWMD files relevant for the project.

Modules: Activate or deactivate modules

Additional Definitions: Definition of additional pathes to BSWMD files relevant for the project.

Additional Definitions: Definition of additional pathes to BSWMD relevant for this project

---

1 
2 
3 

ECUC File Reference: Include mechanism for external ECU configuration files from other projects.

Variants: Handling of Variants when used.

Project Settings: The project settings can be changed within this section.

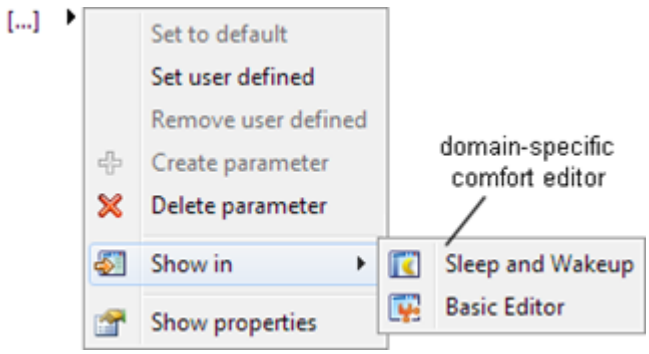
2.1.2 Editors and Assistants

Generally spoken there are two different configuration editor concepts in the DaVinci Configurator Pro.

- **Domain-specific** configuration editors, which provide an optimized view on the ECUC
- **Basic** Editor, which provides a native view on the ECUC

Assistants
An assistants guides you through special tasks like connections, data mapping, memory mapping or task mapping.

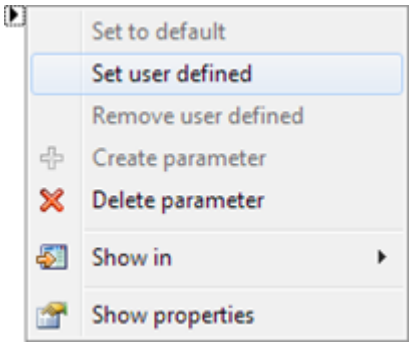
Switch Between the Two Editors
You can switch from the basic editor to the domain-specific comfort editor and back. Move to a setting or a value and click the little triangle to open the context menu. If there is a pendant in the other configuration editor, you can switch from one to the other via **Show in**.



Set User Defined

It is very important to understand that you are responsible for these modifications!!

Move to a setting or a value and click the little triangle to open the context menu.

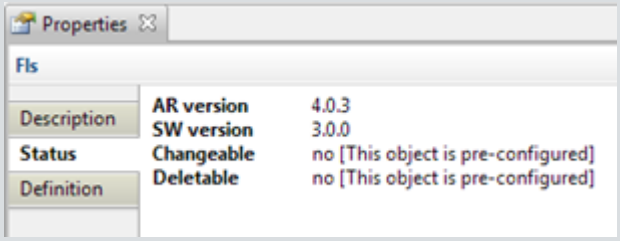


With **Set user defined** you can change values even when they are grayed because their values are derived from the input files or locked by a configuration dependency.

When you use **Set user defined**, the parameter will be marked with a little blue pin on the left side. In the example illustration below the upper check box is marked as user defined, the lower one is not.

Note

Set user defined will not work for pre-configured parameters. The setting in the context menu is grayed and cannot be used. The information about predefined or not can be seen in the **Properties** view. Select **Status** and look at **Changeable** or **Deletable**.



Remove user defined
To set back the user defined status, use the context menu and select **Remove user defined.**

Note

Every parameter set to user defined will provoke a warning during validation with the information, that this parameter was set to be user defined.

Caution!

The feature **set user defined** enables settings that might be terribly wrong. As soon as you use this feature, you have to be sure that you know what you do and you have the complete responsibility.

3 Define Project Settings

Your first steps after the new project is set up, are:

- Define Input files
- Define your specific workflow steps and external generation steps (if needed)
- Activate all necessary BSWs

3.1 Input files

The DaVinci Configurator Pro needs information about your system, the bus communication, etc. This information is stored in the SYSEX or when using legacy file formats dependent on the bus system, in the **DBC**,**LDF**or **FIBEX** files. One or more (e.g. for systems with multiple channels) of these files must be given to the DaVinci Configurator as input files.

The necessary diagnostic information is stored in CDD or **ODX** files and is also needed by the DaVinci Configurator Pro.

3.1.1 System Description Files

You need one of the following description files:

3.1.2 SYSEX

ECU-specific extract of the System Description. This file is typically provided to the TIER1 by the OEM. It contains

- ECU composition
- Atomic SWCs
- Compositions
- Communication
- Data mapping
- typically NO service SWCs

3.1.3 ECUEX

Base for the ECU development is the ECU Extract of System Description, ECUEX for short. It is created from the SYSEX by flattening the SWC architecture. This means that SWC compositions have been removed, only the atomic SWCs are left. And they now communicate directly with each other. Additionally to the SYSEX, the ECUEX contains:

- Service SWCs
- Service mapping, i.e. service connections between the ports of the SWCS and the ports of the service SWCs.

3.1.4 Legacy Data Base files (DBC, LDF, FIBEX, ...)

The legacy data base files are normally not in ARXML format. They also contain information about EUCs, messages, signals, attributes, etc. Good AUTOSAR tools can read them and convert their information into AUTOSAR-conform notation. The missing information when using legacy file formats must

be given to the configuration tool by the software developer.

3.1.5 Diagnostic Data Files

Your Project uses Diagnostic? Therefore you need one of the following diagnostic description files:

3.1.6 CDD / ODX

These files contain the diagnostic description for the ECU.

- ***.CDD**
  The CANdelaStudio Document (*.cdd) format is specified by Vector for describing the diagnostic feature set of an ECU.
- ***.ODX/*.PDX**
  ODX (Open Diagnostic Data Exchange) is an ASAM standard which defines a unique, open XML exchange format for diagnostic data. A packaged ODX file (*.PDX) comprises all files of an ODX project.

3.1.7 State Description

Diagnostic session and security level management is modeled as a state machine which describes the transitions between sessions and states triggered by the execution of diagnostic services.

If a CANdela Document (*.CDD) is used as diagnostic description file, all diagnostic session and security level related information will be imported from the input file.

If a Packaged ODX file (*.PDX) is used as diagnostic description file, it will depend on the used ODX version if an additional state description is required.

- **ODX 2.2.0**
  For ODX 2.2.0, the state machines are defined within the ODX file, using the data model intended for this purpose.
- **ODX 2.0.1**
  As ODX 2.0.1 does not provide means to explicitly model diagnostic sessions, security levels and their corresponding transitions, dedicated SDGs (special data groups) are used to annotate the ODX data with supplemental information about the diagnostic session behavior. This information is not sufficient in all cases and an additional state description will be mandatory to augment the ODX data with the required information during the import of the diagnostic description.

A state description is a *.CSV (comma separated values) file which defines the transition matrices of the diagnostic session and security level state machines.

3.1.8 Standard Configuration Files

In some cases a OEM-specific preconfiguration is necessary, therefore additional Standard Configuration Files provided by your OEM are necessary. Add fragments of ECUC modules which will be used as project specific mandatory configuration.

3.2 Custom Workflow Steps / External Generation Steps

In STEP7 Code Generation on page 68, after the configuration, DaVinci Configurator Pro generates code. You can additionally let DaVinci Configurator Pro call any external tool during this step in a free configurable order. If something is to be done before code generation, if you use some command line

tools or whatever, add these tools to the list and define the call order. With STEP7 Code Generation on page 68, all your mentioned tools will be also called.

3.3 Activate BSW

Which modules are activated and which are not depends on the information in the input files. Activate or deactivate the modules.

4 Validation

Nothing done but imported some files and already many mistakes in the configuration? Why is that?

The DaVinci Configurator Pro provides powerful validation routines that run in the background named **Live Validation**. At start-up of your project, there is only the information from the input files of STEP2 Define Project Settings on page 27. The content of these input files normally has leaks, sometimes errors or the information of the system is still missing and has to be accomplished during configuration. This all leads to a number of errors detected by the validator of DaVinci Configurator Pro. But with the powerful solving algorithms and assistants, it is very easy to solve off the first warnings and errors.

4.1 Validation Concept

The validation concept comprises:

- Detailed validation of the ECU configuration
- Comprehensive set of domain-specific validation rules
- Navigation from validation message to the editors
- Tool makes proposal for solving an error (solving actions)
- Automatic consistency (auto-solving action) after changing the configuration via Comfort Editors or via the Basic Editor

Live Validation [1]

The Live Validation is performed in the background after loading a project or after changing the configuration. The Live Validation directly gives feedback after entering wrong values.

On-Demand Validation [2]

This validation is performed when you start a generation process. You can start the validation via **Project | on-demand validation**. The on demand validation will also launch external generators or execute more complex validation tasks that cannot be executed live.

Note

The result of a validation is displayed in the **Validation View**.

1 

2 

5 BSW Configuration With Configuration Editors

5.1 DaVinci Configurator Pro Editors

To configure the BSW modules you can use the **Basic Editor** or the more comfortable and cluster-specific **Configuration Editors**.

Basic Editor 

The Basic Editor is a generic configuration editor (GCE) and includes the native view of the ECU configuration. It is based on the basic software module description (BSWMD) format and displays all modules of the ECU configuration.

Configuration Editors 

The Configuration Editors offer a use-case oriented view of the ECU configuration. They are optimized for displaying or configuring those parts of the ECU configuration, which are related to the use case.

Use the Configuration Editor first. To get the necessary settings for your fist step in dependency of your OEM and its use cases refer to Start Configuration with Configuration Editors of the step by step description.

6 Software Component (SWC) Design

6.1 Data Exchange between DaVinci Developer and DaVinci Configurator Pro

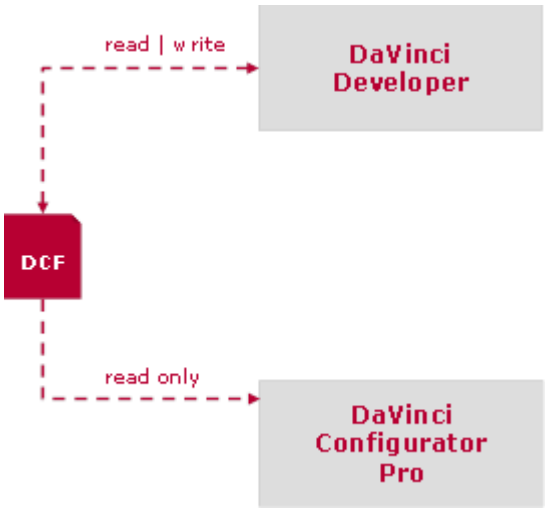
The data exchange between DaVinci Developer and DaVinci Configurator Pro is done via the DaVinci Developer workspace, file with **DCF** format.

DaVinci Developer uses this file to store the complete project information. It can read and write this file.

DaVinci Configurator Pro only reads this file and only at tool start-up.

Note

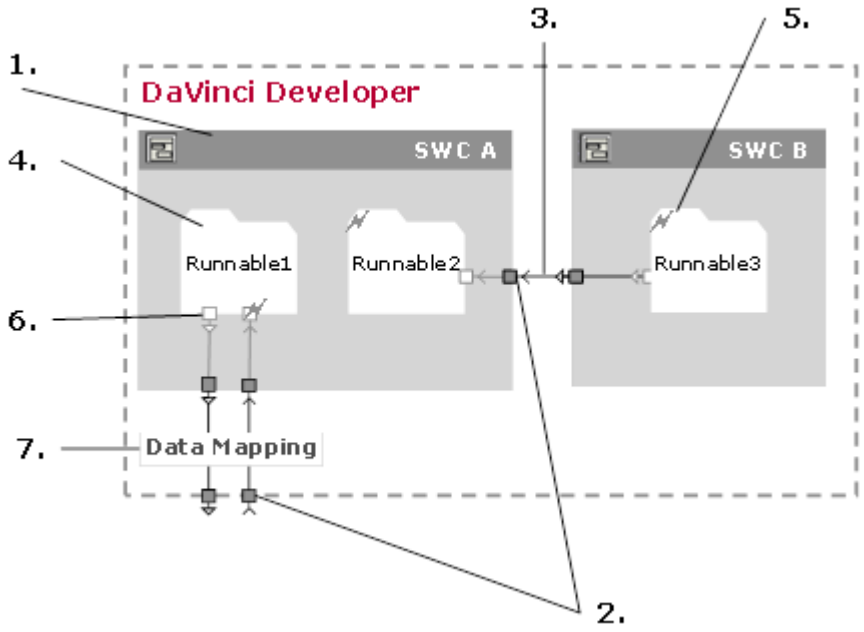
Please re-open the DaVinci Configurator Pro if you have changed the DCF file via the DaVinci Developer to get the current data from the DCF file.



Keep in mind that the DaVinci Configurator Pro is not able to write to DCF file.

6.2 About Application Components, Ports, Connections, Runnables and More...

Learn all about application components, ports, connection and runnables, how they work and how to exchange information between application components.



1. Application Components
2. Ports, Port Init Values and Data Elements
3. Connections
4. Runnables
5. Triggers
6. Port Access
7. Data Mapping

Cross Reference

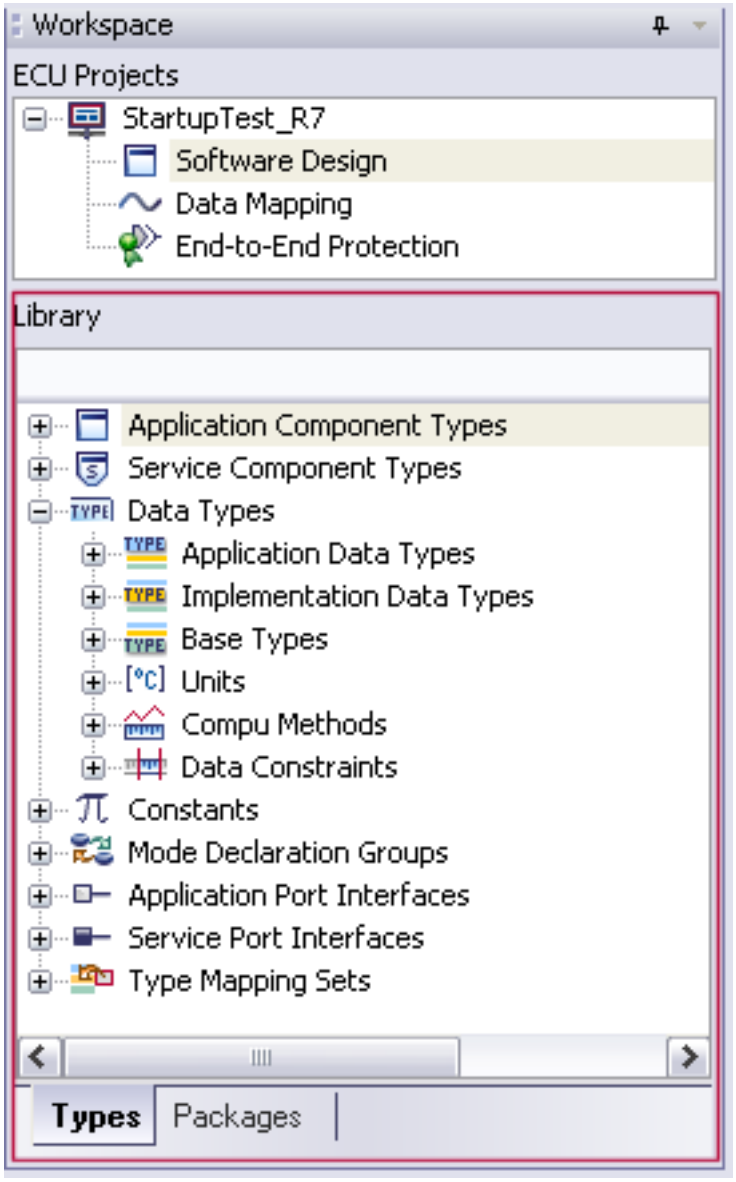
To get basic tool handling information, refer to the online help of the DaVinci Developer.

6.3 Application Components

To create application components use the DaVinci Developer.

6.3.1 The Library – Type and Package

You find the Library of the DaVinci Developer (in standard configuration) below the ECU Projects.



Types or Package?

The Library you use in the following steps can basically be used in two ways.

- Type-oriented
- Package-oriented

The type-oriented workflow is shown in the following steps. But you can also choose the package-oriented one. This concept will now be described briefly.

Packages

With Packages you can pack elements together like:

- Application Component Types
- Service Component Types

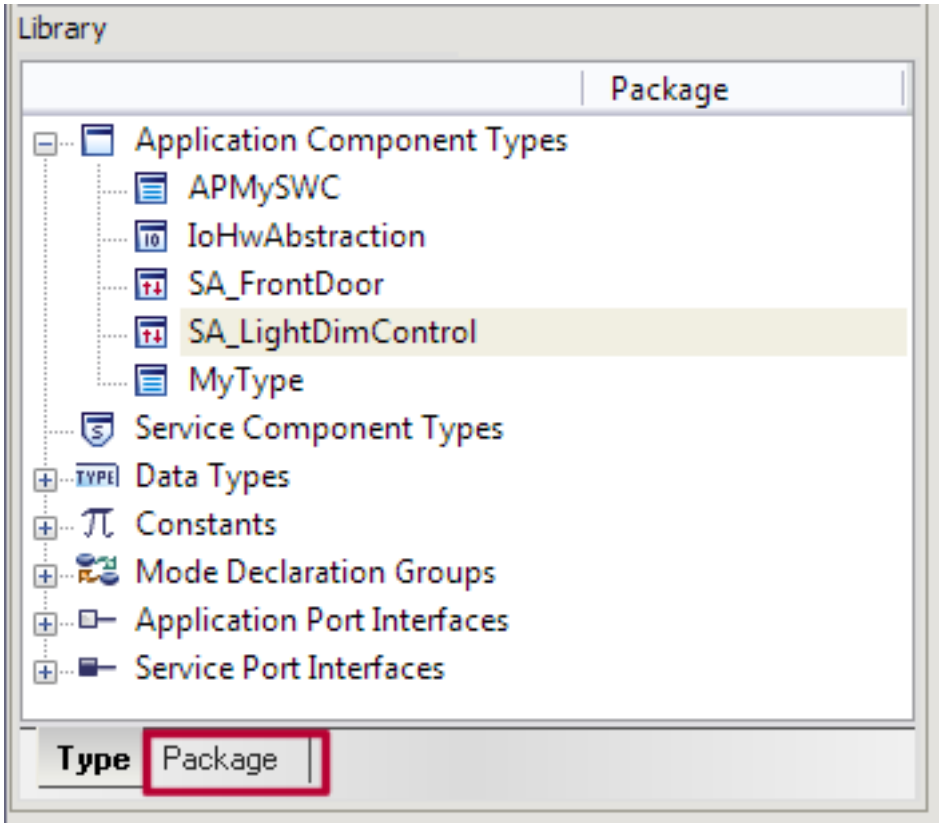
- Data Types
- Constants
- Mode Declaration Groups
- Application Port Interfaces
- Service Port Interfaces

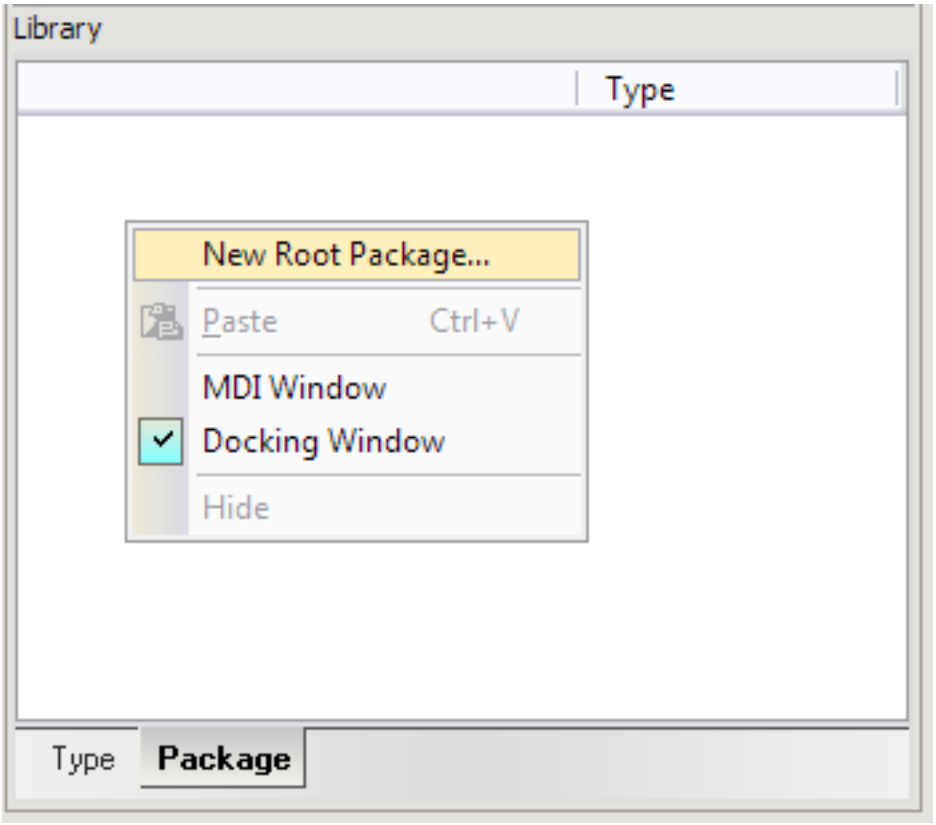
It is like a kind of grouping with its own name space. A name of an object has to be unique within a package but can be used more than once within the project.

These packages can be exported and imported for round-tripping with other tools also supporting the packages.

Switch to package view

At the bottom of the Library you can select Type view or Package view. Select the Package.
You need a Root Package first. Create it via right-click in the empty Package view.

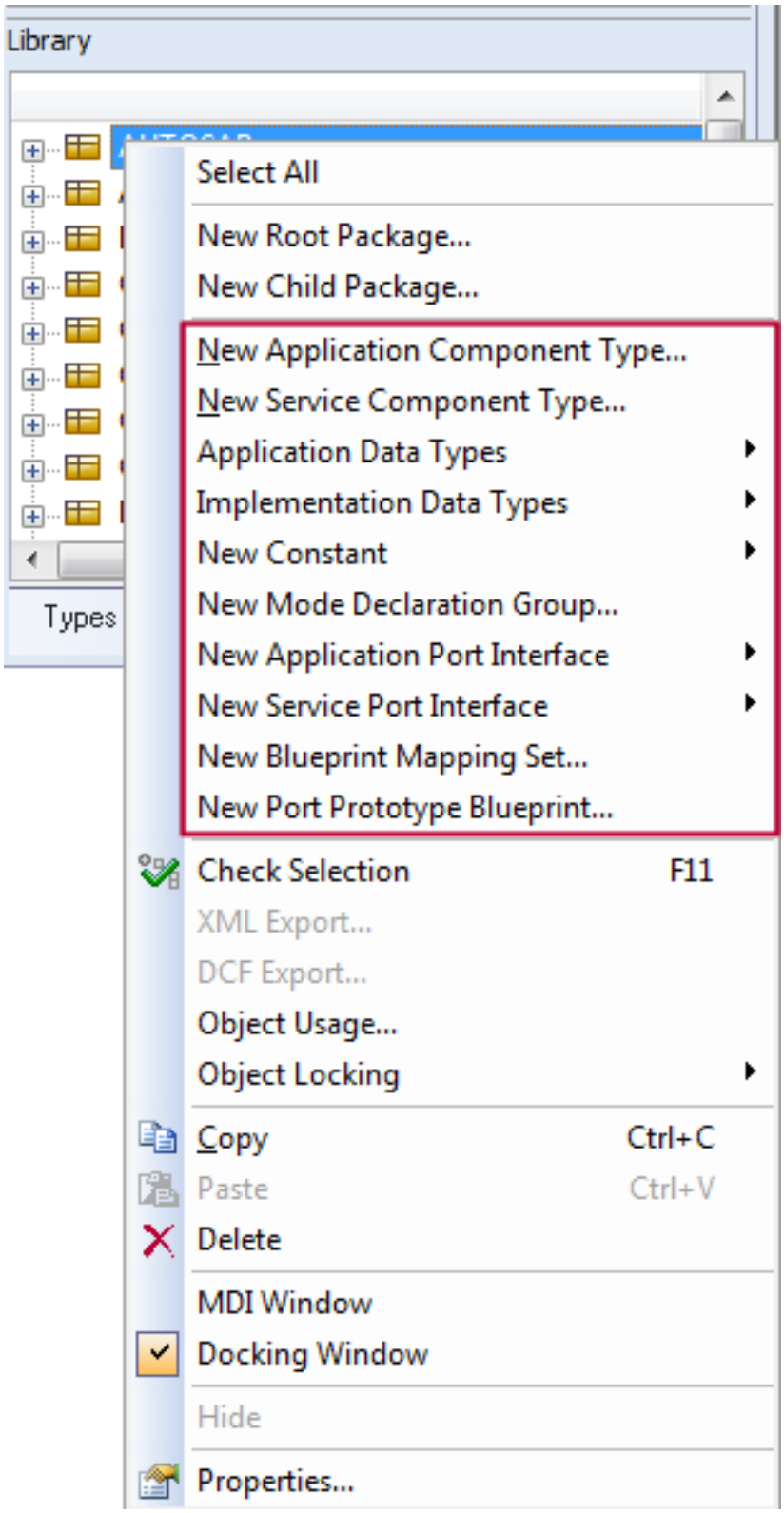




New Child Package...

Then right-click the root (e.g. MyECU_PartA) and you will get to the context menu. Use New Child Package to nest the packages or you put the objects plain below the root package.

The context menu for the packages



Add Objects

Via Add Objects you open a window to select and add already existing object to your package.

New ...

The section New… of the context menu comprises all available objects which you can also create via the Type view of the Library (explained in the following chapter).

XML Eport..., DCF Export...

Export your package as XML file or as DCF file.

Object Usage...

Shows dependencies between used objects in a tree-view.

Object Locking APMySWC

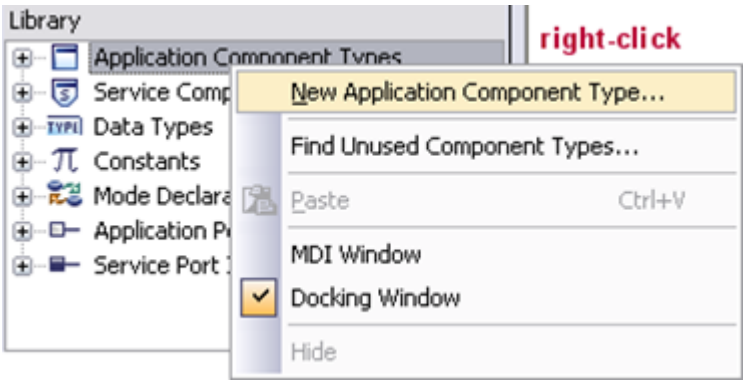
Objects can be locked and unlocked. A locked object is displayed with a little lock. Locked objects cannot be deleted without being unlocked before.

Define new Component Types in the Library

Right-click on Application Component Types in the Library and select New Application Component Type.

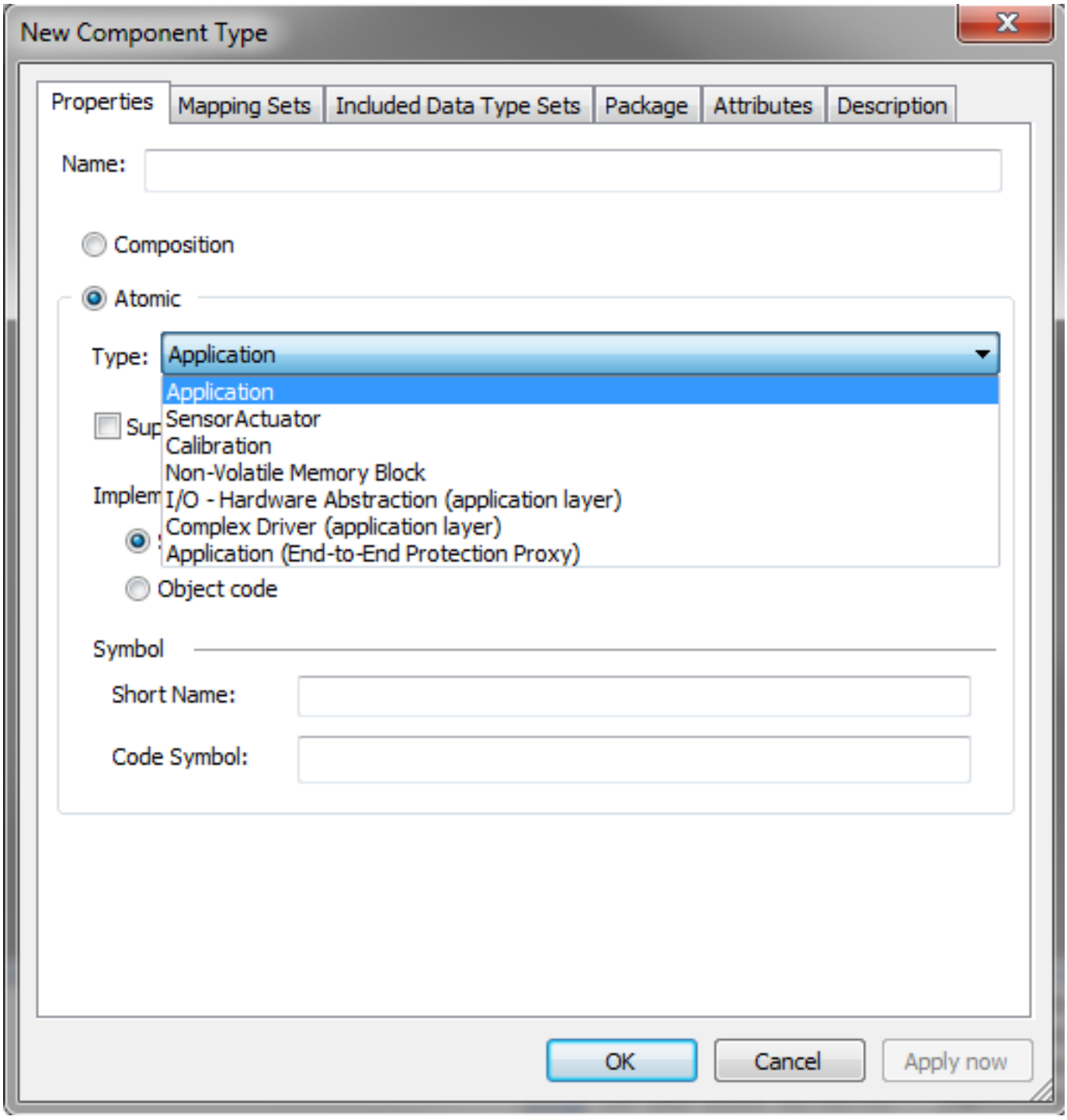
6.3.2 New Application Components

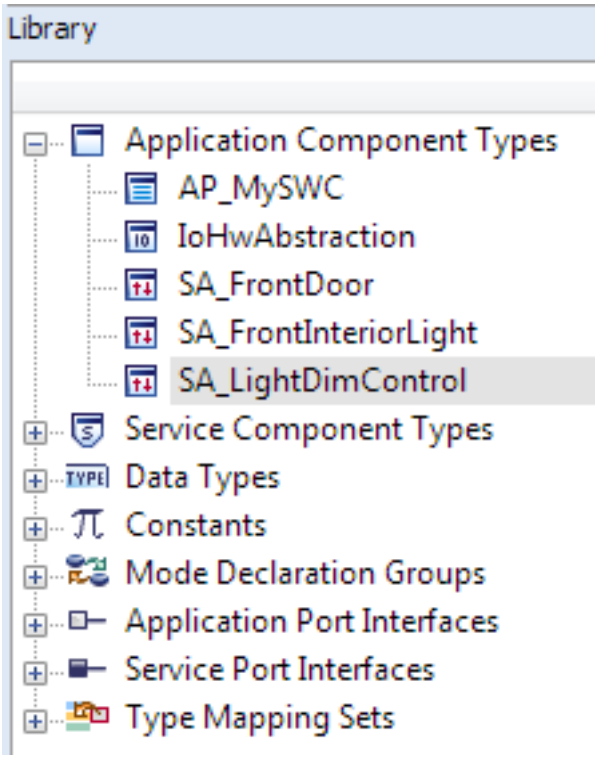
Right-click on Application Component Types in the Library and select New Application Component Type.



It is good to give a speaking name to see what kind of component it is. e.g. AP – Application, SA – Sensor / Actuator

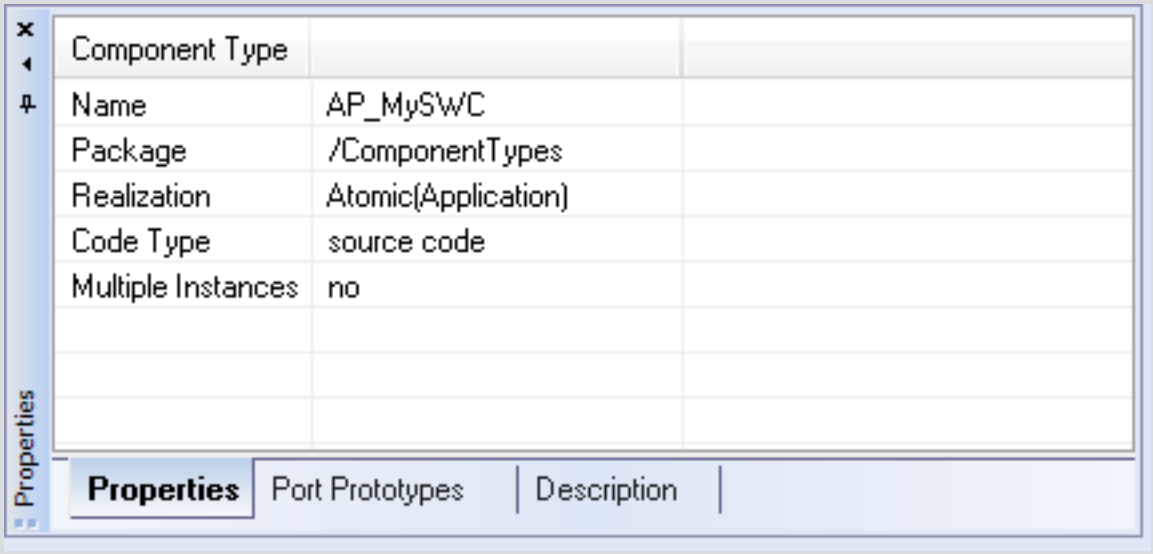
Enter a Name for the Application Component Type. Select Composition if the SWC should contain other SWC or select Atomic. Select its Type and confirm with [OK]. All application software component types are displayed in the library.


New Component Type
Properties
Mapping Sets
Included Data Type Sets
Package
Attributes
Description
Name:
Composition
Atomic
Type:
Application
Application
SensorActuator
Calibration
Non-Volatile Memory Block
I/O - Hardware Abstraction (application layer)
Complex Driver (application layer)
Application (End-to-End Protection Proxy)
Object code
Symbol
Short Name:
Code Symbol:
OK
Cancel
Apply now



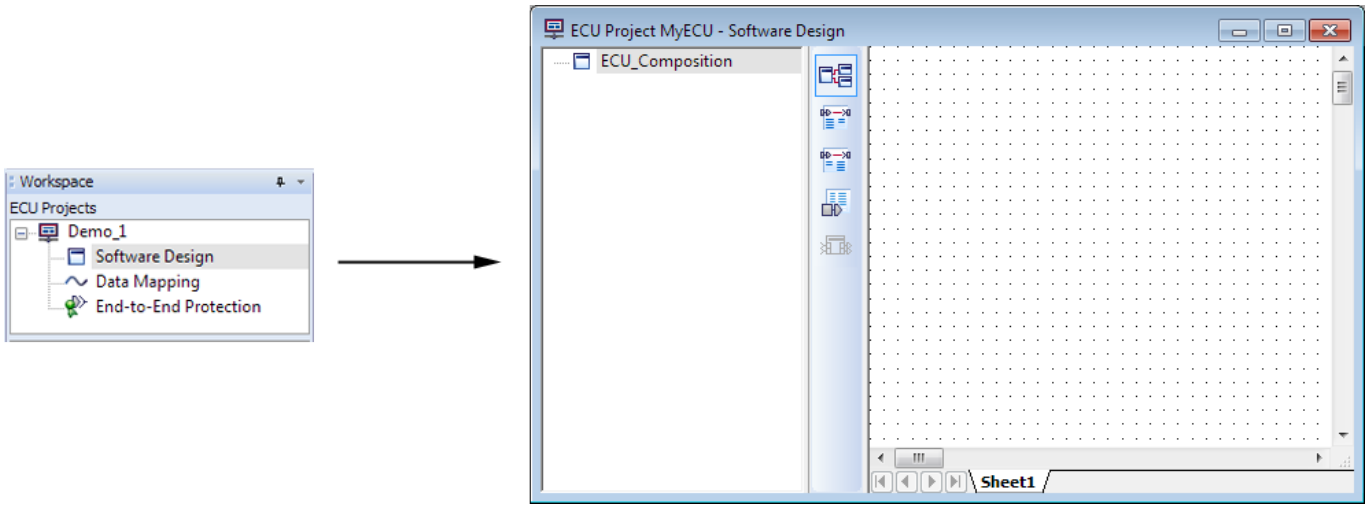
Note

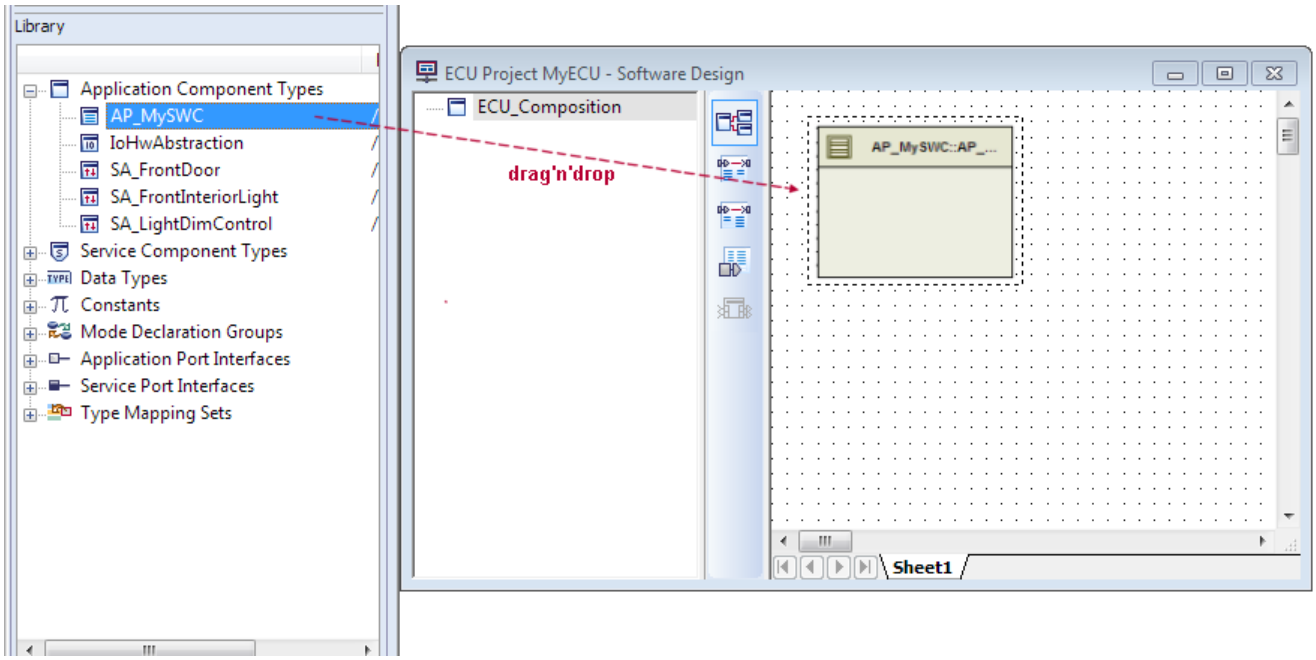
To get fast information about any element in the library just select one and see its Properties, Port Prototypes (if available) and Description in the Properties view (bottom left).



Component types become component prototypes by being used.

Double-click Software Design in the ECU Projects view to open the Software Design view for your ECU project (e.g. MyECU).





Note

In the graphical representation of an application component prototype you can change its size using the little squares shown at the angles and in the middle of the sides.

An application component type becomes an application component prototype when it is used. It also needs a name. Using drag'n'drop, both names are the same. Open the properties window from the context menu for a component prototype to change the prototype name.

Define and use as many application software components as you need for your ECU.

Note

The notation of the software component starts with the software component prototype followed by the software component type.

6.3.3 Understand Types, Prototypes and Interfaces

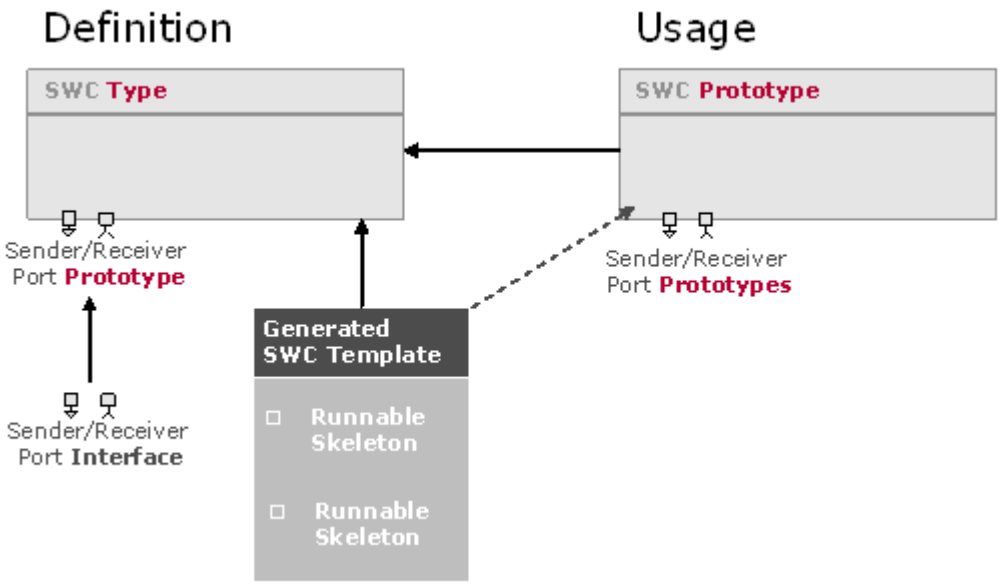
This illustration will help you to understand working with the DaVinci Developer. You have to deal with

- Prototype
- Type
- Interface

When is a software component a software component type, when a software component prototype? When is a port a port interface, when a port prototype?

In the library, the software components are types, the ports are interfaces. As soon as you use them, they become prototypes.

- Port Interface used by a component type ➔ Port Prototype
- Component Type in library used in software design view ➔Component Prototype



Runnables are always connected to the component type.

6.4 Ports, Port Init Values and Data Elements

To communicate and to exchange information the components need so-called ports (S/R ports).

For communication between software component ports have to be defined.

There are different kinds of application/service ports,

- Sender Ports [1] to provide information
- Receiver Ports [2] to receive information
- Sender/Receiver Ports[3] to provide and receive information within one port
- Server Ports [4] to provide services (operations)
- Client Ports [5] to use services (operations)

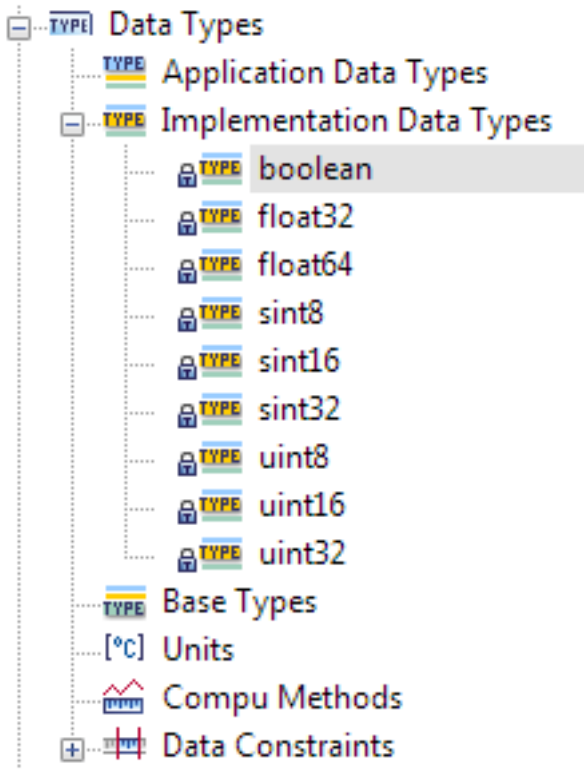
The following ports are listed here for completeness.

- Calibration Ports to hand over calibration parameters
- Mode Ports to e.g. trigger or not trigger runnables within certain modes

Before you can use application ports you have to define application port interfaces. To completely define the port interfaces you have to define data types first, if you don't want to use the predefined ones.

Predefined data types in library

Just right-click and select from the list.



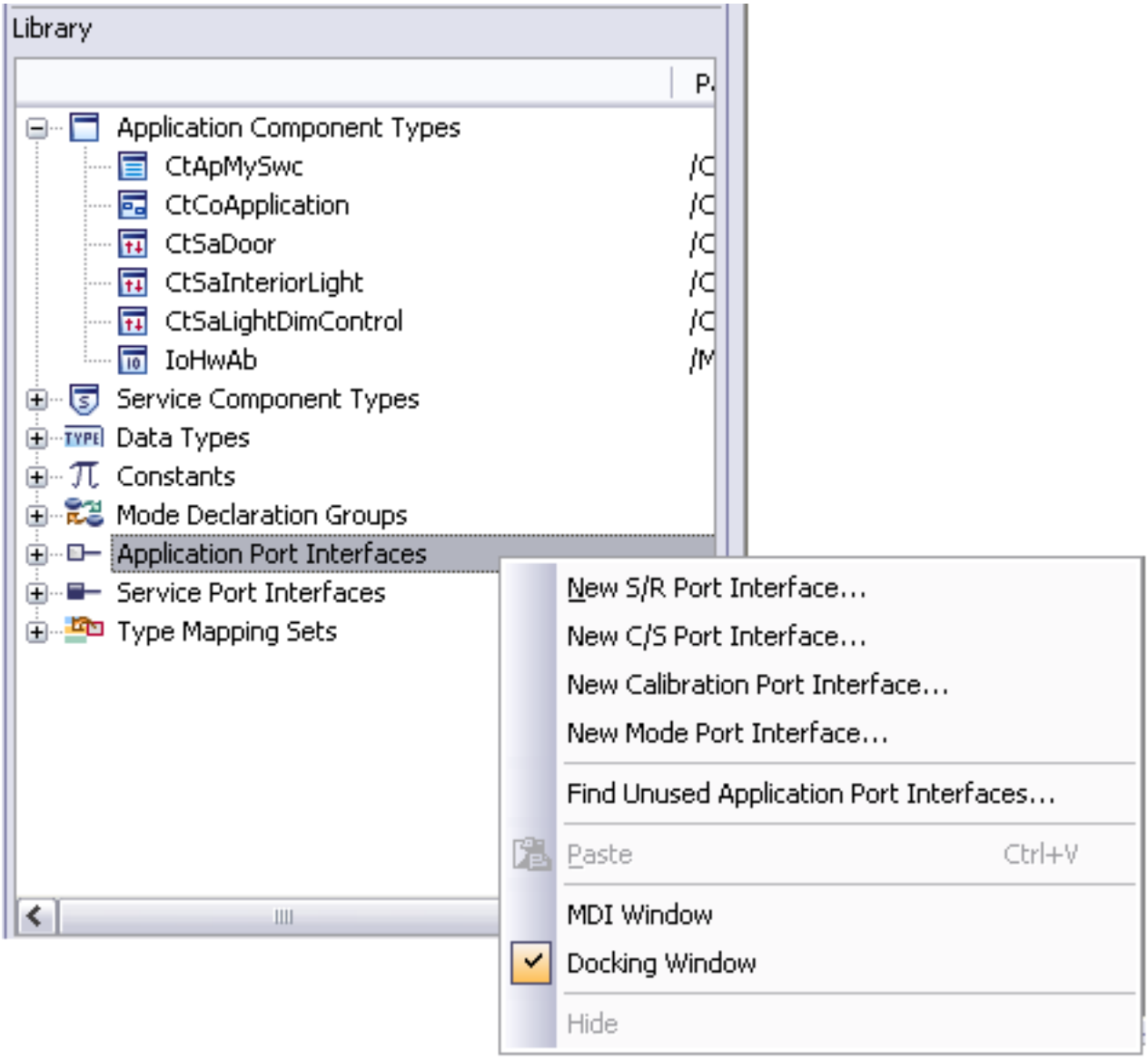
1
2
3
4
5

Caution!

You can handle application port interfaces like application component types. Define the application port interfaces in the library and then use them as application port prototypes for each application component. The same applies for data types and data elements.

Create Port Interface

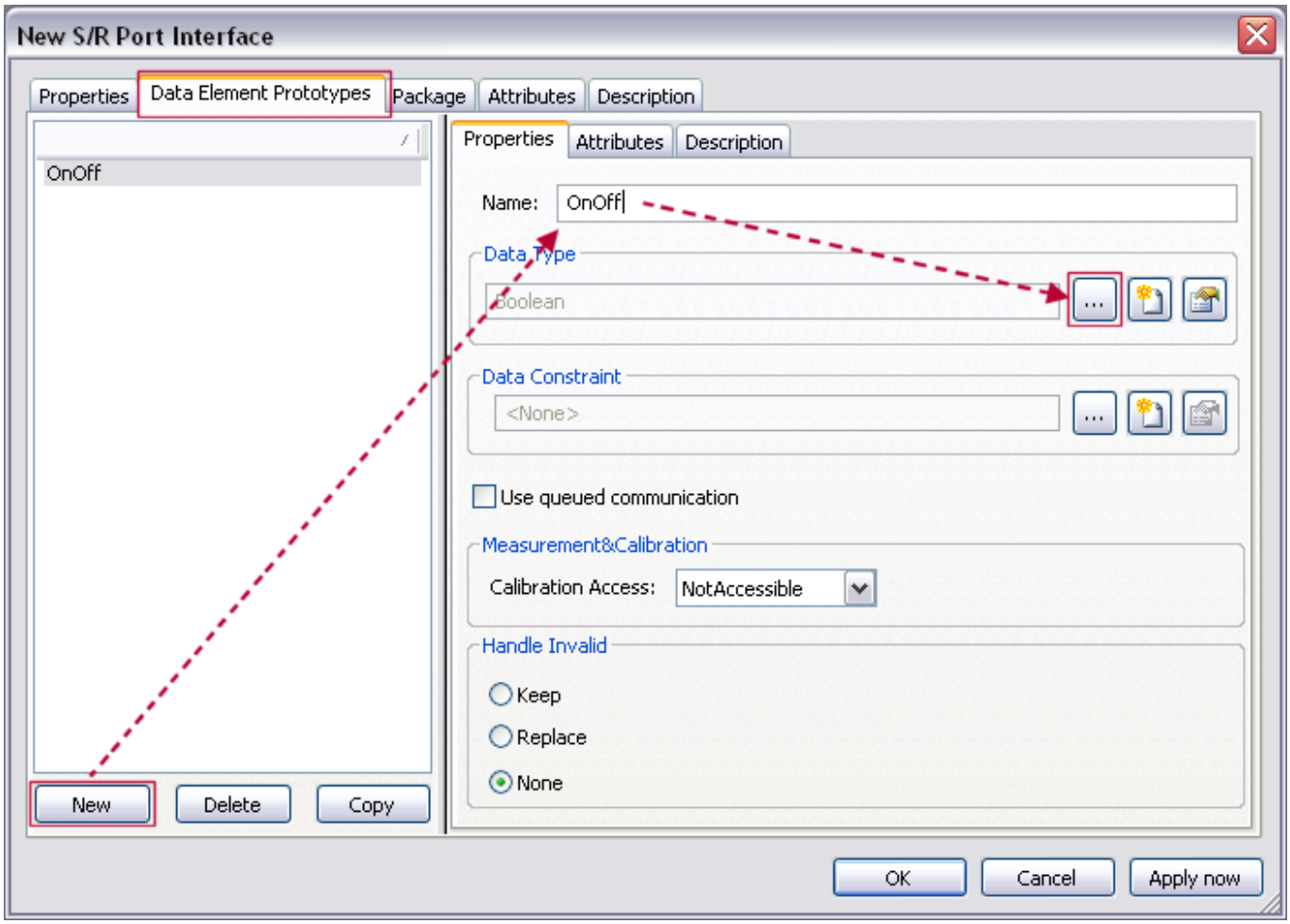
To define a new application port interface, right-click the Application Port Interface in the library and select New S/R Port Interface….



Then the window for the S/R Port Interface opens. Enter a name, select the tab **Data Element Prototypes** and define the content of the port. In this case it is just one data element called OnOff with the Data Type Boolean, use the button […] to select Data Type.

Note

Data elements are the contents of ports.



Note

A port interface can carry many data elements of different data types.

Provide software components with ports

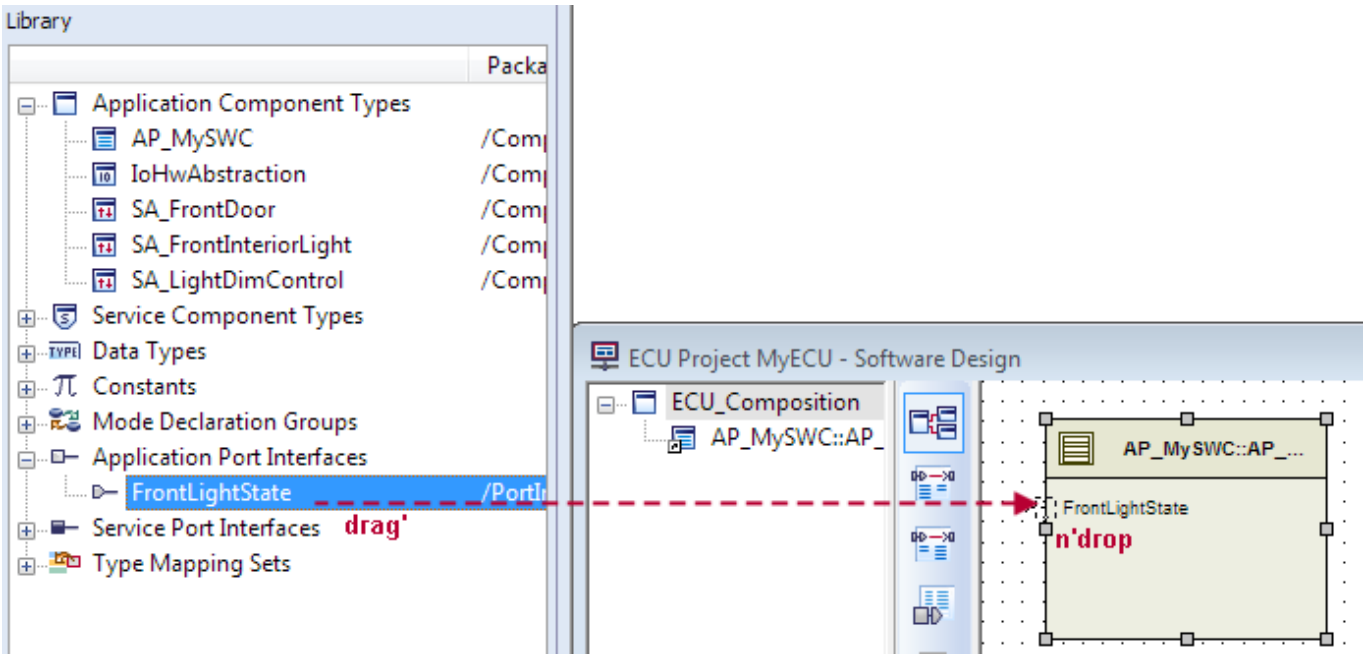
> You have created your application software components.

> You have created necessary application ports interfaces and their content (data element prototypes).

Now you have to define which application component needs which application ports.

Add ports to the application components via drag'n'drop

Select an application port interface from the library and place it onto the application component via drag'n'drop. This transfers a port interface into a port prototype.



Note

Press <Ctrl> while drag'n'drop to change between sender or receiver port.

Interfaces

The ports are added to the components as graphical element together with the name of the port.

- Sender port[1]
- Receiver port[2]

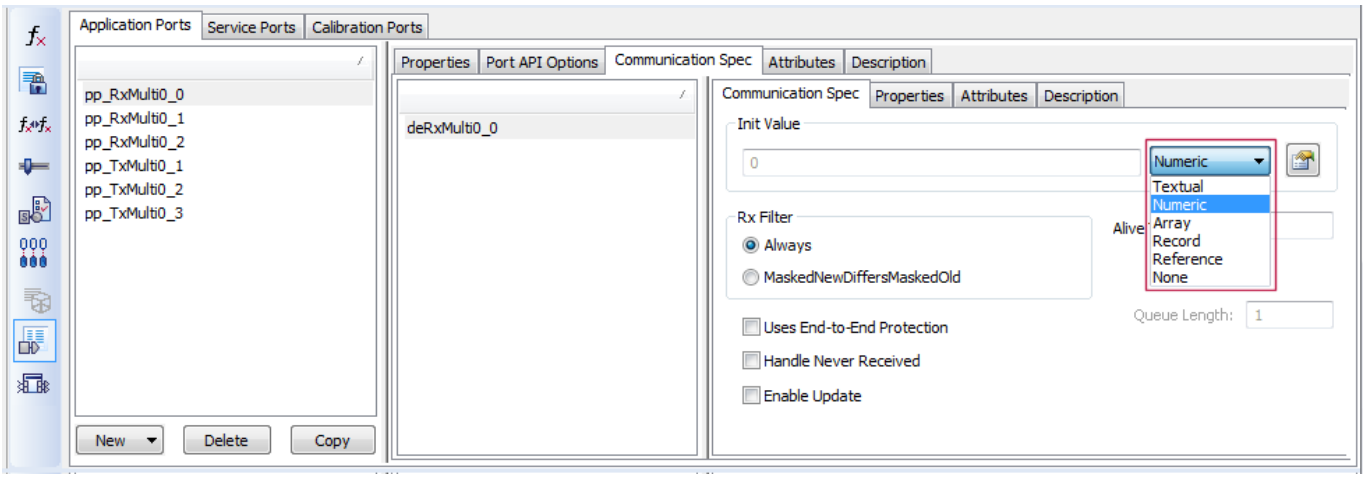
Note

Ports need Init Values.

1
2

Port Init Values

You have to assign an initial value to every used application port.



6.5 Configure Service Ports within your Application Components

Your Service Components will be loaded automatically to the DaVinci Developer workspace .

Note

The DaVinci Configurator stores the service components within the folder **<Project Folder->\Config\ServiceComponents**.

The content of the service components are not part of the DaVinci Developer workspace.

If you want to connect one of your Application Component to a Service Component, you have to define a Service Port Prototype based on a Service Port Interface.

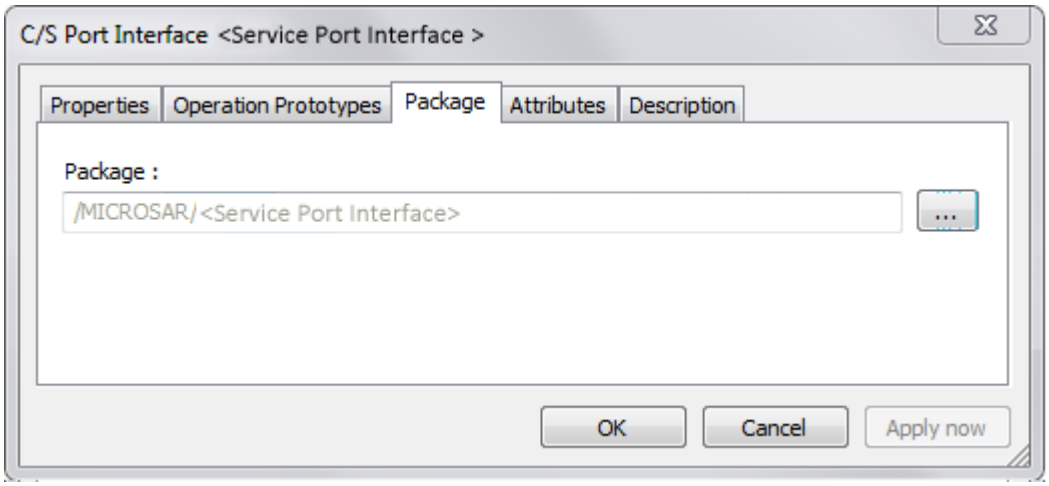
Service Port Interface name and content depend on the BSW configuration and version of the service component within the DaVinci Configurator, so it is not stable. An Application Component shall always define its own or a general Service Port Interface definition.

The Service Port Interface definition from the loaded Service Component is always in read-only status. These Service Port Interfaces are marked with a read-only icon[1] within the library.

You have currently not defined your Service Port Interfaces?

Then copy and paste the definition from the Service Component. Assign the definition to an own package, so it is possible to use the same name as before. Therefore open the Service Port Interface, select **Package** tab and choose a package via **[...]**.

1



Assign your copied Service Port Interface to your Application Component. Therefore open your Application Component Prototype, select Port Prototype List[1], open **Service Ports** tab and add your Service Port via **[New]** | **From Port Interface**. Now the Service Port used by your Application Component is independent from the Service Component description.

Note

If incompatible changes apply these will be reported by RTE checks.

6.6 Define your Runnables

Now configure runnables that will carry your code.

There are four important topics for a runnable:

- **SYMBOL** and **NAME** – what the runnable skeleton is called in the template file
- **TRIGGER** – when is the runnable executed
- **PORT ACCESS** – what data can the runnable access
- **MAPPING** – in which task context does the runnables work?

Caution!

Runnables can only be defined for atomic application components types.

Note

The runnable skeletons are generated by the DaVinci Configurator Pro/the RTE and can then be accomplished with your code. Find more details follow in the next chapters.

Check also later hints to template generation

1

1. Open a software component via the library or the software design view.
2. Click on the runnable icon [fx].
3. Click [New] and select Runnable.
4. Enter Name and Symbol for the new runnable on the Properties tab.

If you leave the Symbol field empty, the runnables (functions) will be named according to the entry in the **Name** field.

Minimum Start Interval
Define the minimum time interval between the successive triggering of this runnable.

Can[1] **Be Invoked Concurrently**

A runnable can be marked as **Can be invoked concurrently** if it can be executed while it is already running. In other words it can be safely executed concurrently (re-entrant). There are a few criteria for your implementation code:

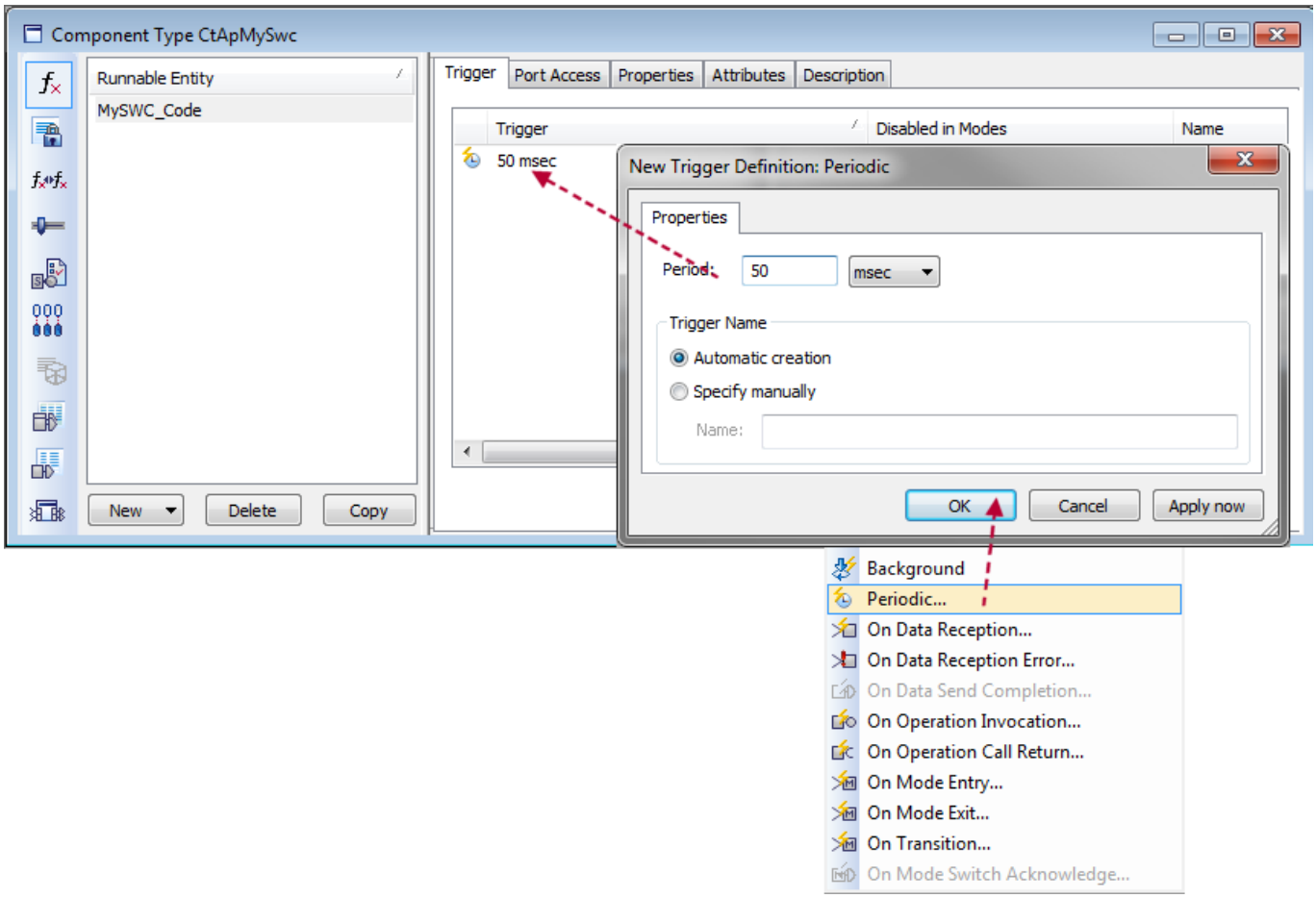
- No static (or global) non-constant data
- No return of address to static (or global) non-constant data
- Only work on data provided by caller
- No modification of own code at runtime
- No call of non-re-entrant runnables

6.7 Triggers for the Runnables

The trigger decides when a runnable is executed.

[1](CAN Driver) The CAN Driver abstracts access to the CAN hardware for sending and receiving messages and for switching between controller states (sleep, stop, etc.)

Select the tab Trigger. On this tab you select when your runnable should be executed.



Note

A runnable can be triggered periodically or via an event.

Periodical: Select the checkbox and enter the cycle time.

On Data Reception: As soon as data is received at the appropriate port the runnable is activated. (Indication)

The trigger **On Operation Invocation** belongs to service ports and will be explained later.

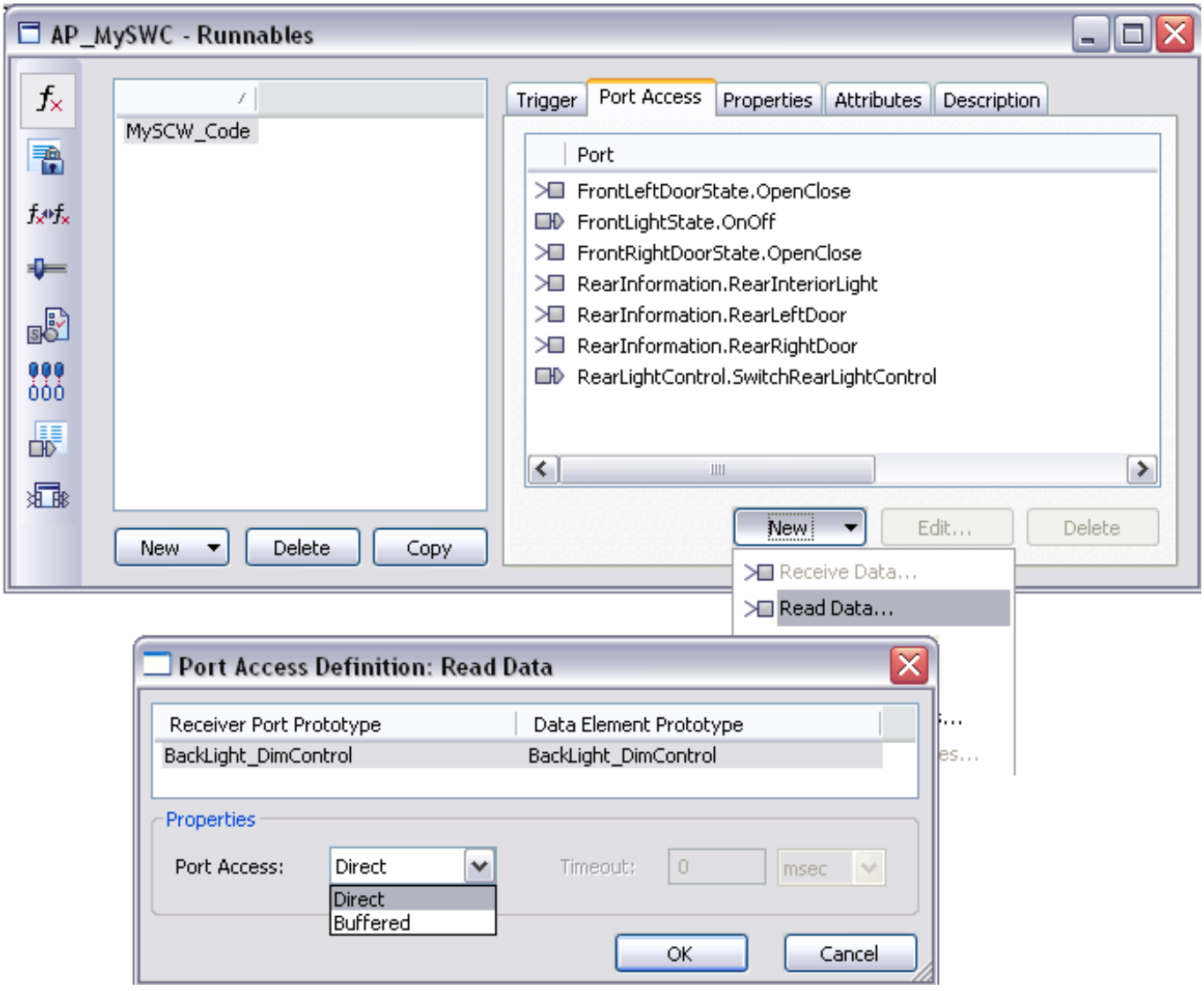
The runnable of the demo application only needs to be called when the state of the doors is changed. This can be realized via a cyclic polling or directly by a trigger from the doors. In the demo, **Periodical**... is chosen.

Note

When experimenting with the demo, replace the cyclic trigger with two triggers On Data Reception..., one for the left and one for the right front door.

6.8 Port Access of the Runnables

Define which port information each runnable should be able to read or write. Select the tab Port Access. You only get displayed accessible ports.



You can define access to

- Read Data… and
- Write Data…
- and later when using client server port also to operations (Invoke Operations).

Click e.g. **Read Data**… and the Port Access Definition:Read Data window will open.

Caution! Direct – Buffered

Direct Write: as soon as you write the information to the data, it is changed immediately.

Buffered Write: the data information is changed at the end of the runnable runtime just before leaving it.

Direct Read: if you access to the data multiple time, it could be changed in the meantime. You always read the current information.

Buffered Read: with the start of the runnable, the data is copied to a buffer. Every time you access the data, it has the same value until the runnable is left.

Note

In the header of the runnable's skeleton that will be generated by the DaVinci Developer, you will find a list with all available API functions for this runnable. If any access is missing, go back to the DaVinci Developer and check whether the Port Access is set correctly.

Summary

Summary of the settings for your runnable MySWC_Code:

- The runnable is called MySWC_Code
- Its functional representation is called: MySWC_Code.
- It is triggered periodically
- The runnable has access to the state of the left and right door and to the data of the front interior light. It also has access to the LightDimControl.

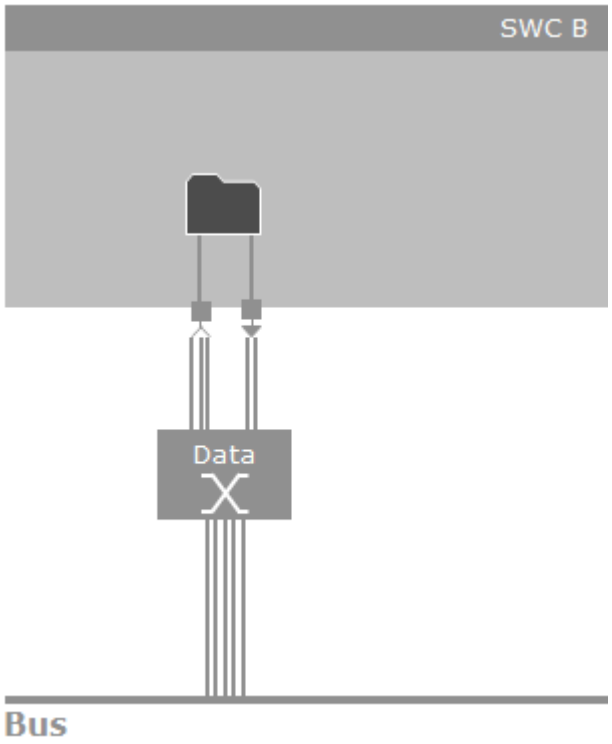
7 Mappings

Generally spoken Mapping stands for assignment of one object to another. There are different mappings in the context of AUTOSAR. What is meant here is:

- Data Mapping
- Task Mapping
- Service Mapping
- Memory Mapping

7.1 Data Mapping

Your ECU has to communicate with other ECUs, the external communication. Signals have to be sent and received via the connected bus system. The signals and the types of data transported via the signals is defined in the data base like DBC, LDF, FIBEX, and also in the SYSEX.



From the view of software components, communication information is exchanged via ports that carry so-called data elements. The definition of a port contains the assignment of data elements that can pass the port. Via the Data Mapping you define, which data element belongs to which signal. For short, you assign data element to bus signals. That’s called Data Mapping.

Note

Data Mapping can also be done in DaVinci Configurator Pro!

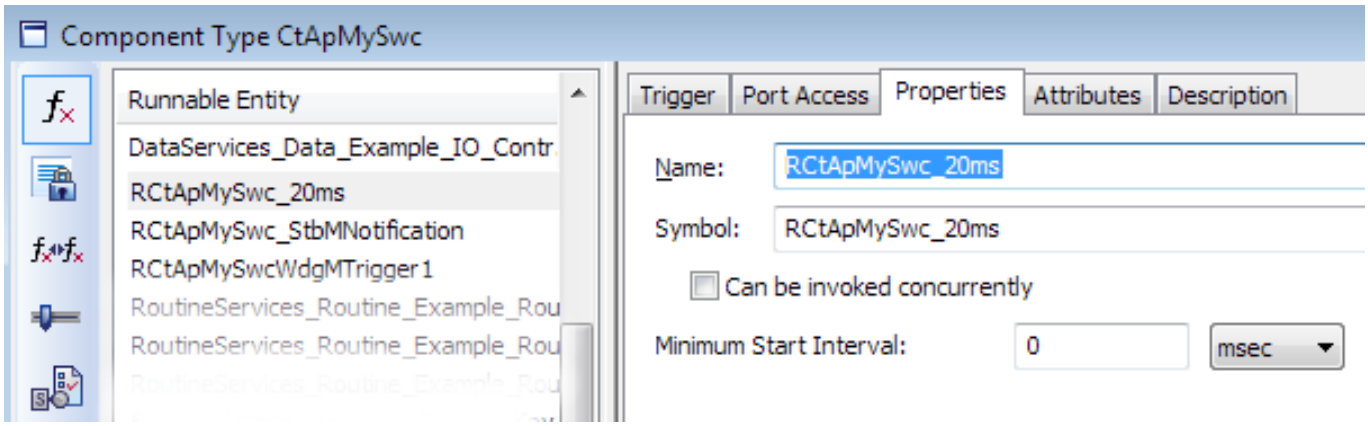
Data Mapping in DaVinci Developer has a higher priority than Data Mapping in the DaVinci Configurator Pro.

Data Mappings done via DaVinci Developer cannot be overwritten by the ones done via DaVinci Configurator Pro.

7.2 Task Mapping

Tasks are means of the operating system. You can define as many tasks as you like and need. You define the name of the tasks, its priority and its type like **Auto**, **Basic** or **Extended**. You can also define the task as non- or full preemptive.

With Task Mapping, you have to map all runnables to tasks, expect **On operation invocation** triggered runnables. Those are normally called only from one task and have no problem with reentrance.



A runnable has to be mapped to a task if it is not re-entrant but could be called re-entrantly during system operation.

What happens if you do not map a runnable that should be mapped?

The Vector AUTOSAR Solution provides an intelligent RTE generator.The RTE generator checks necessary conditions and will warn you if there are unmapped runnables that have to be mapped.

What happens if you map a runnable that does not have to be mapped?

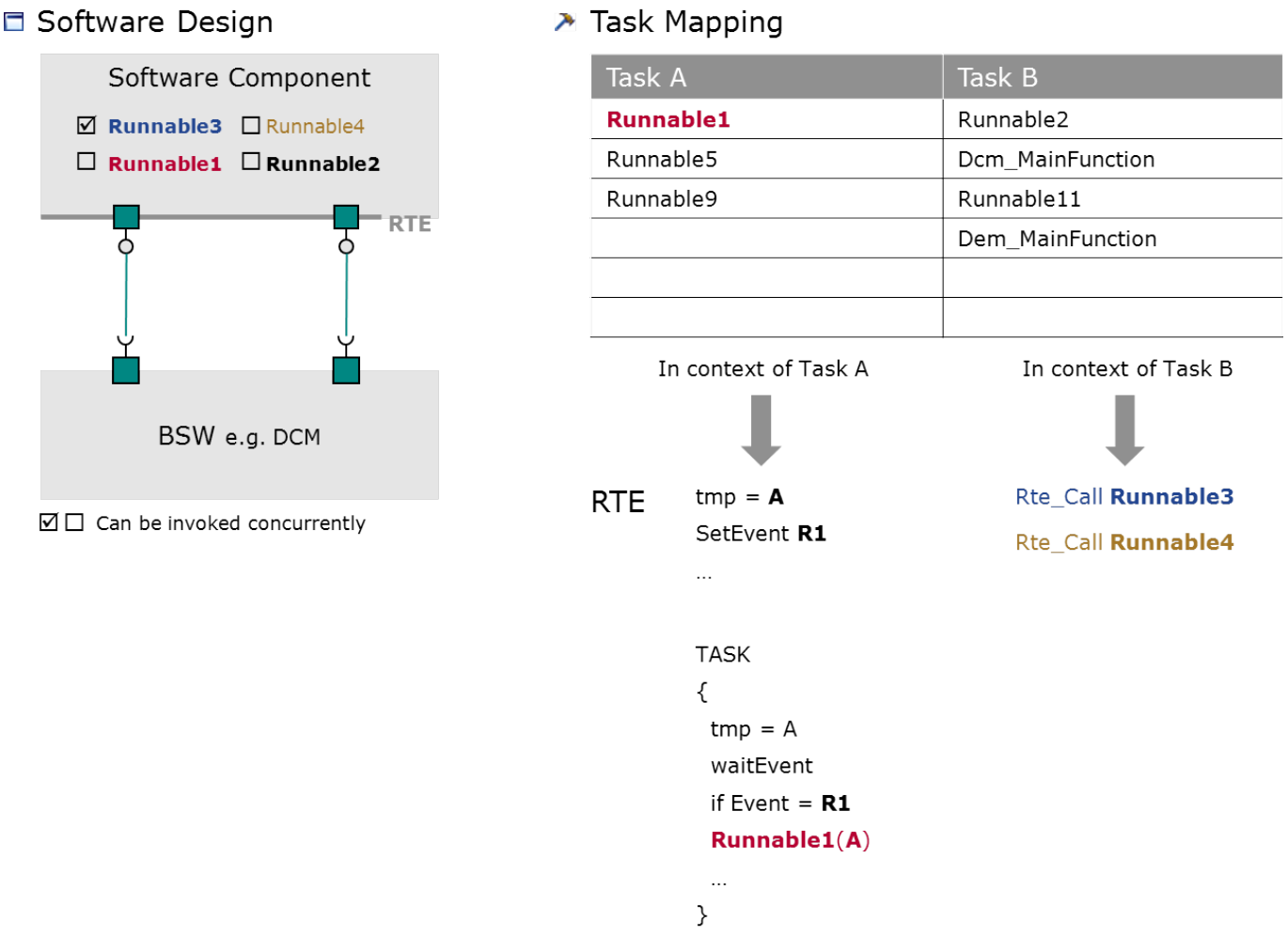
Nothing will happen. It will still work. But the code efficiency and RAM consumption could be less good.

Is there a disadvantage of the RTE generator automatism?

It could be less comfortable to figure out the call context of the runnables and the stack consumption could increase.

7.2.1 Information about Interaction between Runnable, Re-entrance and Task Mapping

Your goal is a configuration of the system that results in a highly runtime-optimized and memory consumption-optimized code. Here is a little information to estimate what the RTE generator does.



This illustration shows a software component with four runnables. Only runnable 3 is set to Can[1] **be invoked concurrently**. On the right-hand side you see which runnable is mapped to which task and the result of this mapping for the generated code. Here is the details.

Can be invoked concurrently is set and the runnable is not mapped to a task

Example: Runnable 3

Runnable 3 is called directly in the context of the calling BSW (DCM in this example – DCM_MainFunction, TASK B).

[1](CAN Driver) The CAN Driver abstracts access to the CAN hardware for sending and receiving messages and for switching between controller states (sleep, stop, etc.)

Caution

Make sure that your runnable is really re-entrant (see section **Can Be Invoked Concurrently**). If not, errors can occur that are difficult to debug.

Can be invoked concurrently is NOT set and the runnable is not mapped to a task

Example Runnable 4

If the RTE generator finds out that the runnable is not called multiple times in parallel, Runnable 4 is called directly in the context of the caller BSW (DCM). Otherwise an error message will be shown.

Can be invoked concurrently is NOT set and the runnable is mapped to a task different from the task where the main function of the caller (e.g. Dcm[1]_MainFunction) is mapped.

Example: Runnable 1

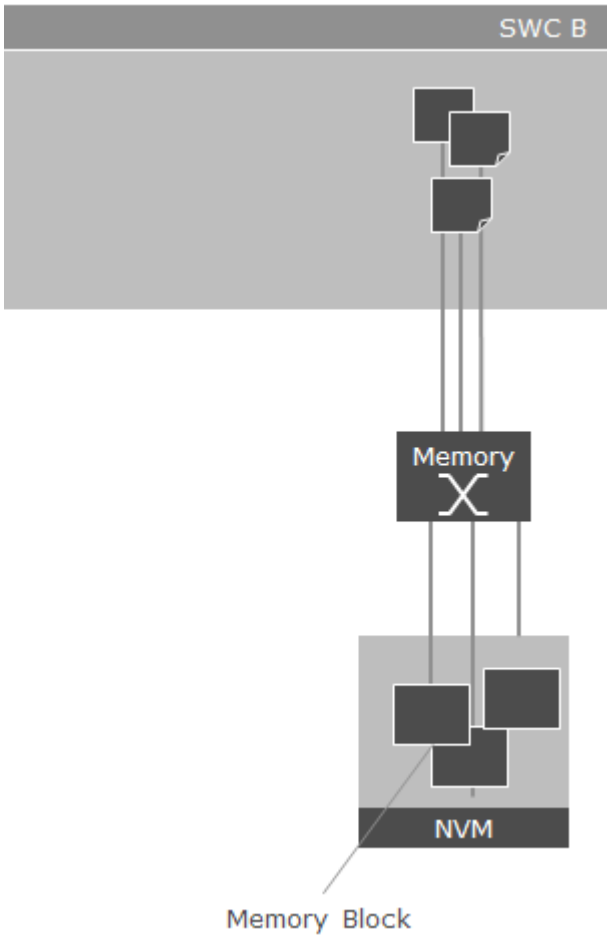
When the caller BSW now wants to activate the runnable via the RTE call, an event is set and the temporary variables of the runnables are stored to RAM (context of calling BSW, e.g. Task B). When the point in time has come, the WaitEvent in Task A is triggered by the event and starts the Runnable1(A) and hands over the parameters previously stored to RAM.

As you see, runtime is longer than using direct call and the RAM consumption is higher. E.g. for diagnostics: DCM runnables always have array types that are completely stored to RAM. For e.g. 100 DIDs à 4 bytes you need 400 bytes!

[1](Diagnostic Communication Manager) The Dcm module implements diagnostic communication according to ISO 14229-1:2006 (UDS). Some diagnostic requests are processed directly in the Dcm (management of diagnostic sessions, reading of error codes, EcuReset, etc.) and some are routed to the SWCs via port interfaces (reading, writing and controlling of data elements within a data identifier, execution of routines, etc.). Legal requirements of OBDII / SAE J1979 are also supported.

7.3 Memory Mapping

Within ECUs there is data that has to be persistent during power-off. Therefore this data cannot be stored simply to RAM, it must be stored to EEPROM or Flash, i.e. to non-volatile memory. As an example for this data, think of the DEM information.



There are two possible ways how this non-volatile data will be used. Either your application always accesses a RAM copy of the NV data where the data is copied at start-up. Or data can be read/written directly from/to NV memory on request.

The Memory Mapping assigns your defined PIM, per instance memory to the memory blocks of the NVM.

A per-instance memory object is mappable if it is referenced by a service need object. The definition of per-instance memory objects or service needs is part of the component type definition and is not yet editable by DaVinci Configurator Pro.

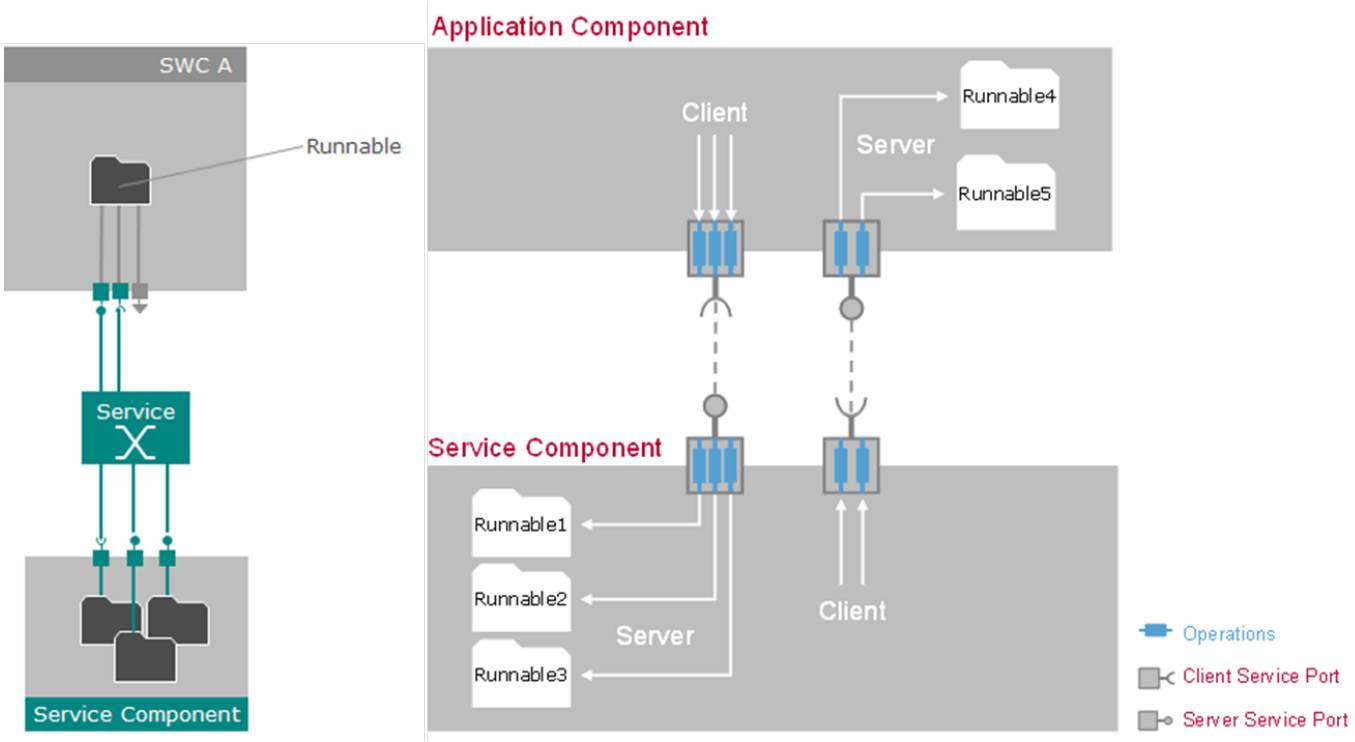
7.4 Service Mapping

The base for the service mapping is the theory about clients and servers. The client uses a service of the server. The server itself provides the operation to the client. The client server communication between your application software component and a service component is done via service ports – client ports and server ports. A server port provides services (one or more operations) and a client port uses these services.

A service component can have server ports and client ports.

- **Service port of the service component is a server port**
  The service is provided by the service component and your application software component just uses the operations (services, functions)
- **Service port of the service component is a client port**
  Your application software component must be server and has to provide the operation (service, function). In this case, you have to program code, i.e. a runnable. This runnable must be created by you and it is triggered by the request of the service.

Besides the sender and receiver ports, there are also client and server ports. Sender and receiver ports carry data elements. Via client ports you can access so-called operations (or functions) of the servers.



Within one service port there could be n operations. Every single operation has to be assigned to a runnable.

The servers provide the runnables that contain the code. Here Runnable1, Runnable 2, Runnable 3 of the service component and Runnable 4 and Runnable 5 of application component.

When performing the service mapping, it is almost the same as the data mapping – but now you deal with services instead. To be able to access the services of a service component, you have to add this service component to your software component.

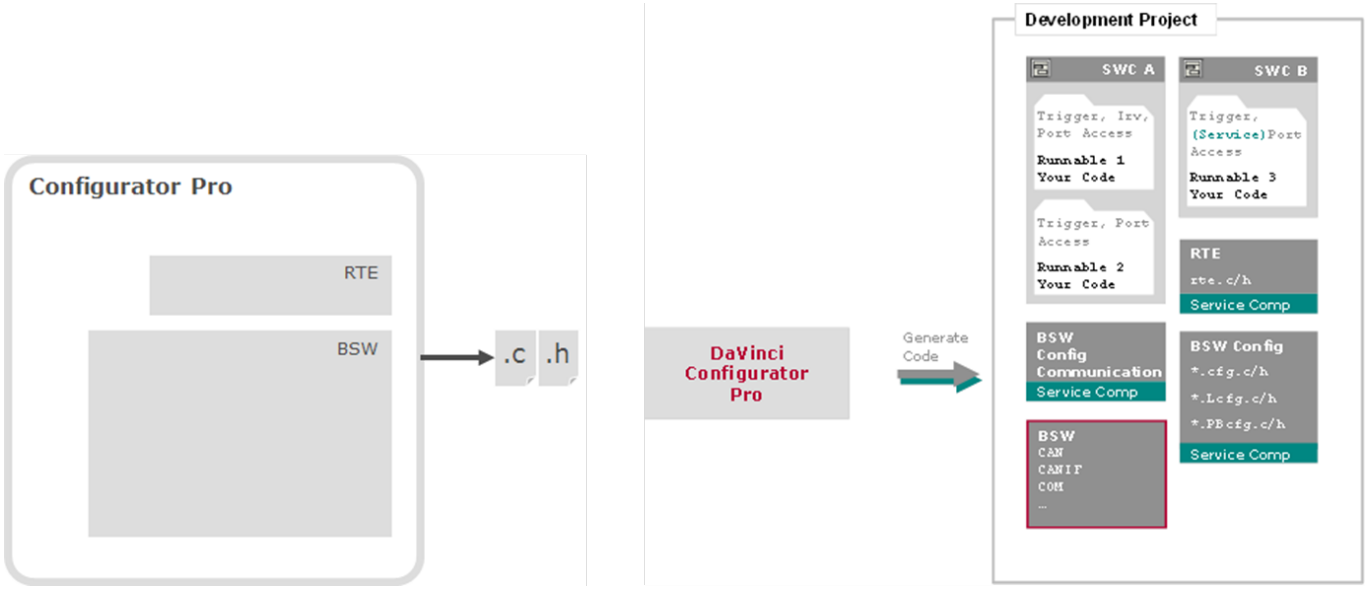
Then you can use all services that are provided by the service component. In this case your application is the client and the service component is the server.

But using a service component not only brings benefit – it also costs. Almost every service component does not only provide services – there can also be client ports on the service component side. And if you add this service component to your software component, you have to provide all the services where the software component provides a client port interface.

8 Generation

With the generation step DaVinci Configurator Pro generates the code for your project and also concerns your settings in STEP2 Define Project Settings on page 27. DaVinci Configurator Pro clings to your settings and works off the "to be generated" list.

The DaVinci Configurator Pro provides an automation interface for remote-controlled validation and code generation.



8.1 MICROSAR Rte Gen

The RTE is generated by the **MICROSAR** Rte[1] **Gen**. Its generation depends on many settings in the configuration tools and how runnables are mapped to tasks. The files start with **Rte_** or SchM[2]**_**.

[1](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

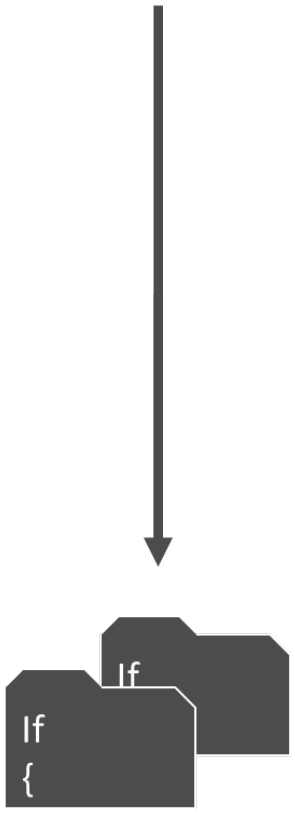
[2](BSW Scheduler) The SchM module is integrated in the (RTE, calls the periodic "main" function of the individual BSW modules, and provides functions for critical sections. For the distribution of the BSW (master satellite concept) via partitions and core boundaries, the SchM provides nearly identical communication interfaces like the (RTE.

9 Runnable Code

Until now you don’t have written one line of C code. And that’s typical for the AUTOSAR concept of software development. There is much more configuring work with tools than programming or implementing as you are normally used to from previous ECU software projects with e.g. CANbedded.

Example

This is an example showing the runnable called MySWC_Code with the function name (Symbol) MySWC_Code




```
/*******************************************************************************************************************
 *
 * Runnable Entity Name: SwitchFrontLight
 *
 *------------------------------------------------------------------------------------------------------------------
 *
 * Executed if at least one of the following trigger conditions occurred:
 *   - triggered on DataReceivedEvent for DataElementPrototype <OnOff> of PortPrototype <FrontLightState>
 *
 *******************************************************************************************************************
 *
 * Input Interfaces:
 * =================
 *   Explicit S/R API:
 *   -----------------
 *   Std_ReturnType Rte_Read_FrontLightState_OnOff(Boolean *data)
 *
 * Client/Server Interfaces:
 * =========================
 *   Server Invocation:
 *   ------------------
 *   Std_ReturnType Rte_Call_FrontInteriorLight_DEFECT_OP_GET(IoHwAb_BoolType *signal)
 *     Synchronous Server Invocation. Timeout: None
 *     Returned Application Errors: RTE_E_env_FrontInteriorLight_DEFECT_E_NOT_OK
 *
 * Service Calls:
 * ==============
 *   Service Invocation:
 *   -------------------
 *   Std_ReturnType Rte_Call_Event_DTC_0x000002_SetEventStatus(Dem_EventStatusType EventStatus)
 *     Synchronous Service Invocation. Timeout: None
 *     Returned Application Errors: RTE_E_DiagnosticMonitor_DEM_E_QUEUE_OVERFLOW, RTE_E_DiagnosticMonitor_E_NOT_OK
 *
 *******************************************************************************************************************/

FUNC(void, RTE_SA_FRONTINTERIORLIGHT_APPL_CODE) SwitchFrontLight(void)
{
/*******************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << Start of runnable implementation >>            DO NOT CHANGE THIS COMMENT!
 * Symbol: SwitchFrontLight
 *******************************************************************************************************************/

IoHwAb_BoolType lightDefect;


Rte_Call_FrontInteriorLight_DEFECT_OP_GET(&lightDefect);

if (lightDefect)
{
  /*Light is detected to be defect. So any light information from MySWC_Code is turned to light off.*/
  Rte_Call_Event_DTC_0x000002_SetEventStatus(DEM_EVENT_STATUS_FAILED);  /*set event*/
}
else
{
  /*Light is working well. No special treatment necessary*/
  Rte_Call_Event_DTC_0x000002_SetEventStatus(DEM_EVENT_STATUS_PASSED);  /*set event*/
}
```

This is the header of a runnable showing all necessary information about the runnable like:

- When it is triggered
- Input and output interfaces
- Service interfaces

Note

If some access is missing, go back to the DaVinci Developer, add the necessary port access for your runnable, go back to DaVinci Configurator Pro, synchronize the system description and generate the component templates again.

Then the missing interface should be there and can be used.

10 Compile And Link

10.1 Using your "real" hardware

This step is highly dependent on your compiler / linker / project settings. Compile and link everything together and create a file that can be downloaded to your hardware.

III Additional Information

This section includes additional information dealing with special topics like e.g. multiple user concept or support request package.

1 Update Input Files

Open Input Files editor via **Project** | **Input Files** or Input Files icon [1] at DaVinci Configurator Pro toolbar.

Note

Any change of the input files requires an update of the configuration. Update configuration updates all input files, whether they are changed or not.

1.1 System Description Files

If necessary, replace your File at Input Files | System Description File.

Select the Input File, click Replace button [2] and choose the new **Input File**.

Note

Any change of the input files requires an update of the configuration. Update the configuration via Update Configuration icon [3].

1.2 Diagnostic Data Files

If necessary, update your Diagnostic Description file and select ECU and Variant.

In case of **ODX 2.0.1** as Diagnostic Description file, it is necessary to update the state description. Therefore click **[Synchronize State Description...]**.

Note

Any change of the input files requires an update of the configuration. Update the configuration via Update Configuration icon [4]

1 

2 

3 

4 

2 Update Project Settings

Open Project Settings Editor via **Project** | **Settings** or Project Setting icon [1] at DaVinci Configurator Pro toolbar. Select **Project Settings** to open the view.

1 

3 Support Request Via DaVinci Configurator Pro

If you request support for Vector products, you have to provide project and environment information. Therefore, use the **Support Request Package** function of the DaVinci Configurator Pro.

The **Support Request Package** compiles the necessary project and environment information in a ZIP-file. Send this file to the Vector Support via E-mail. Just follow the description below.

Note

The DaVinci Configurator Pro does not send any data automatically!

1. Open the Support Request Editor via **Help** | **Create Support Request Package...**

Note

The entry **Create Support Request Package**... is only enabled if a project is loaded.

2. Add your First Name, Last Name and E-Mail address.
3. Click **[Next >]**
4. Select project information which has to be added to the support request ZIP-file.

Note

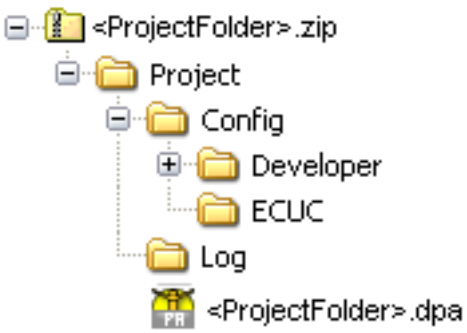
PC Info, Tool Version Info and SIP Info are default and can not be deselected.

5. Click **[Next >]**
6. Enter the path where the created ZIP-file has to be stored.
7. Click **[Finish]**

3.1 Result

In addition to the Zip-File [1], the Project Assistant creates the following folders and files.

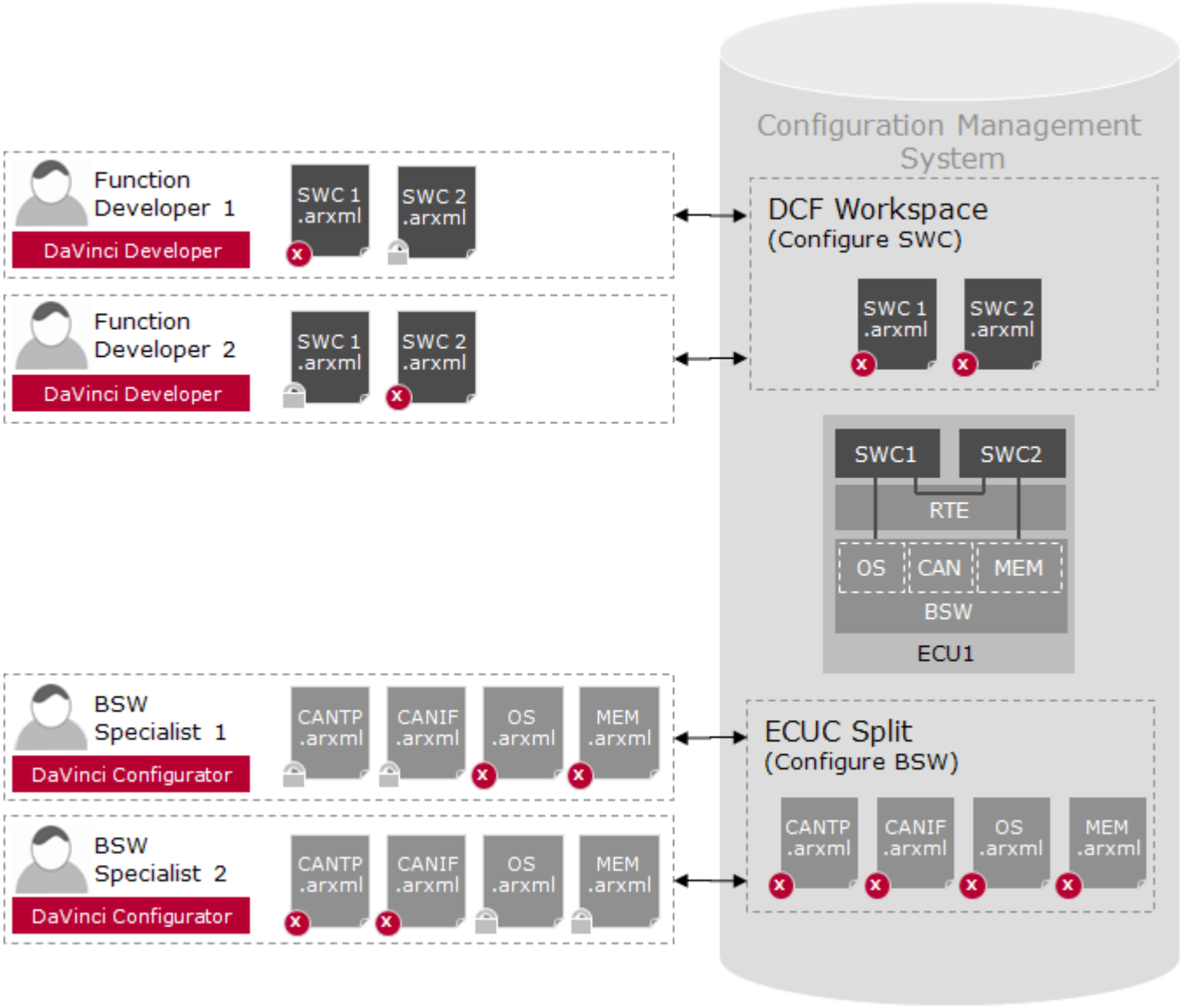
1 <Target Path>.zip

Folder/File	Description
	The Project folder includes all defined project files
	The file includes all necessary project and system information (e.g. version of tools)
	The file includes all necessary project and system information (e.g. version of tools)

Send **<Target Path>.zip** via e-mail to the Vector Support Address embeddedSupport@de.vector.com.

4 Multiple User Concept

This concept helps you to realize the configuration management of the design and configuration data on a fine-grained level in combination with a configuration management (CM) system. It is available for software components as well as for BSW configurations. Split files also help you in multi-user projects: you can control the read/write access on fine-grained level. This helps you to prevent the users from making unwanted changes and therefore reduce the necessary diffs and merges.



4.1 General

The decision whether to use a single configuration file or split configuration files is done in the Project Assistant settings. The resulting split files are ideally managed through a central CM system. Each user needs the full set of split files to work with the project.

The user has to obtain a writeable working copy to his local file system.

The user has to obtain a read-only working copy to his local file system.

Caution!

After a SIP update, all configuration files may have to be updated. This requires all files to be writeable while loading the configuration with a new SIP for the first time.

It is necessary to lock all files which will be edited during the configuration session. The DaVinci tools load all working copy files. The locked files are writeable; all other files are write-protected.

Note

If you prevent parallel editing e.g. by according settings in the CM system you can ensure that only one user can obtain a writeable copy at a time. If you allow parallel editing in the CM system, several users can get a writeable copy. When both users make changes, you can use the Project Merge function to merge the modifications.

After finishing configuration, the locked files have to be checked in (commited) to the CM system.

4.2 Split Files for Software Component and ECU Project Configuration

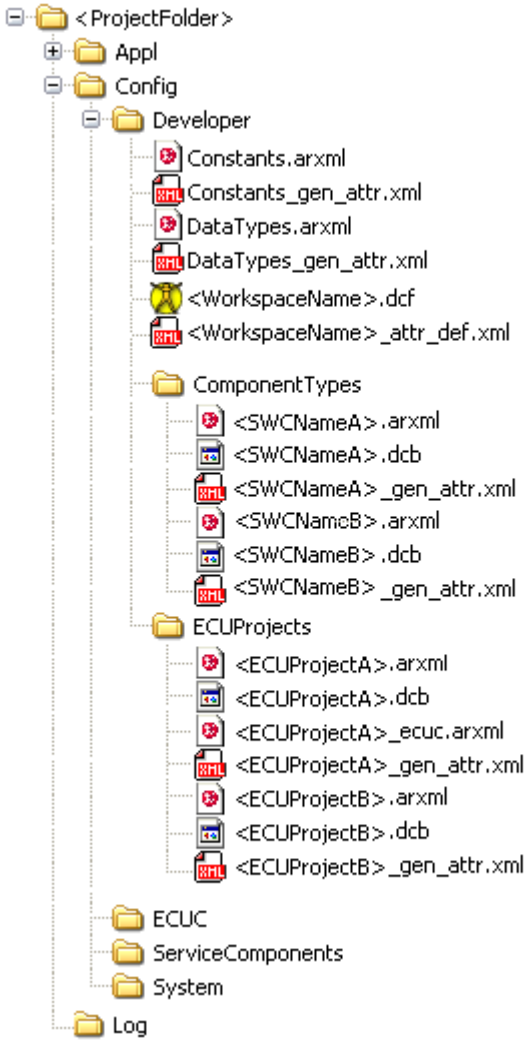
The split of the **DaVinci Developer workspace** is necessary to enable working in parallel for software component configuration.

Note

Workspace splitting is only supported by the DCF workspace format. You chose this option during the project setup and you cannot change it afterwards.

Result

Additionally to the workflow file the Project Assistant creates multiple files for each Software Component Type and ECU Project in the Developer folder.



File	Description
<WorkspaceName>.dcf	This file includes references to all single software component and ECU Project files.
<SWCNameA>.arxml	General software component file
<SWCNameA>.dcb	This file includes the binary part of the software component.
<SWCNameA>_gen_attr.xml	This file includes the generic attributes of the software component.
<ECUProjectA>.arxml	General ECU project file
<ECUProjectA>.dcb	This file includes the binary part of the ECU Project.
<ECUProjectA>_ecuc.arxml	Project-specific ECUC file
<ECUProjectA>_gen_attr.xml	This file includes the generic attributes of the ECU Project.

4.3 Configure Software Component Prototype

To configure a software component prototype it is necessary to lock all files, specific for this software component. (**<SWCNameA>.arxml**, **<SWCNameA>.dcb** and **<SWCNameA>_gen_attr.xml**).

4.4 Configure ECU project

To configure an ECU project it is necessary to lock all files, specific for this ECU project. (**<ECUProjectA>.arxml**, **< ECUProjectA >.dcb**, **<ECUProjectA>_ecuc.arxml** and **<ECUProjectA>_gen_attr.xml**).

4.5 Split Files in DaVinci Developer

Note

Parameters of write-protected files are greyed out in the DaVinci Developer.

If Parameters of write-protected modules are changed because of module dependencies, the DaVinci Developer informs about the write-protection during the synchronization process.

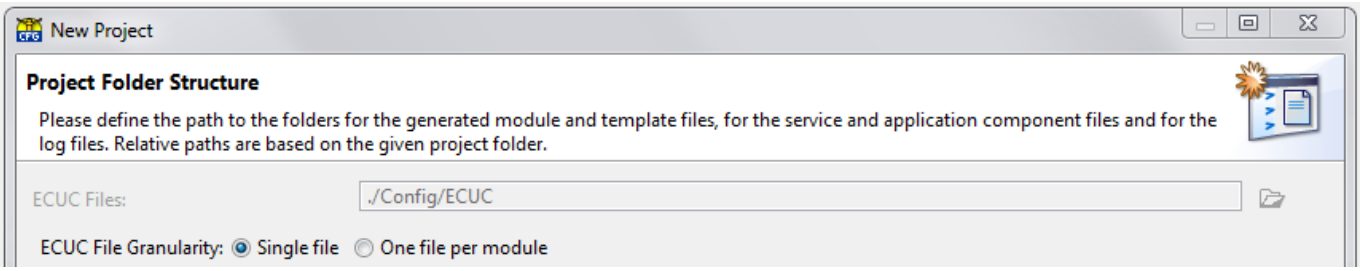
You have the possibility to make the file writeable and repeat the synchronization process.

4.6 Split Files for BSW Configuration

The split of the ECUC is necessary to enable working in parallel for BSW configuration (one file per module).

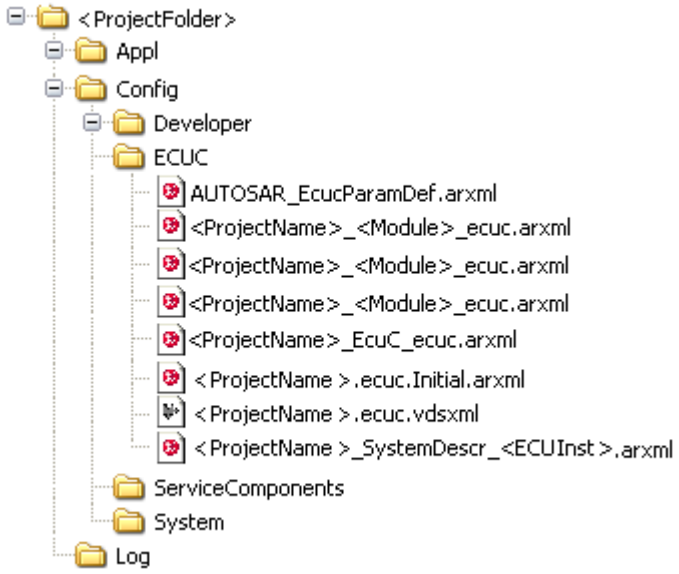
Note

Please note that further .arxml files are created when activating further modules.



Result

Additionally to the standard ECUC files and the AUTOSAR parameter definition file, the Project Assistant creates multiple ECUC files **<ProjectName>_<Module>_ecuc.arxml** to the ECUC folder.



Note

Additionally to the module ECUC files the Project Assistant creates a **<ProjectName>_ EcuC.ecuc.arxml** file, containing references to all module ECUC files. This file is used as project file for the DaVinci Developer and the DaVinci Configurator Pro.

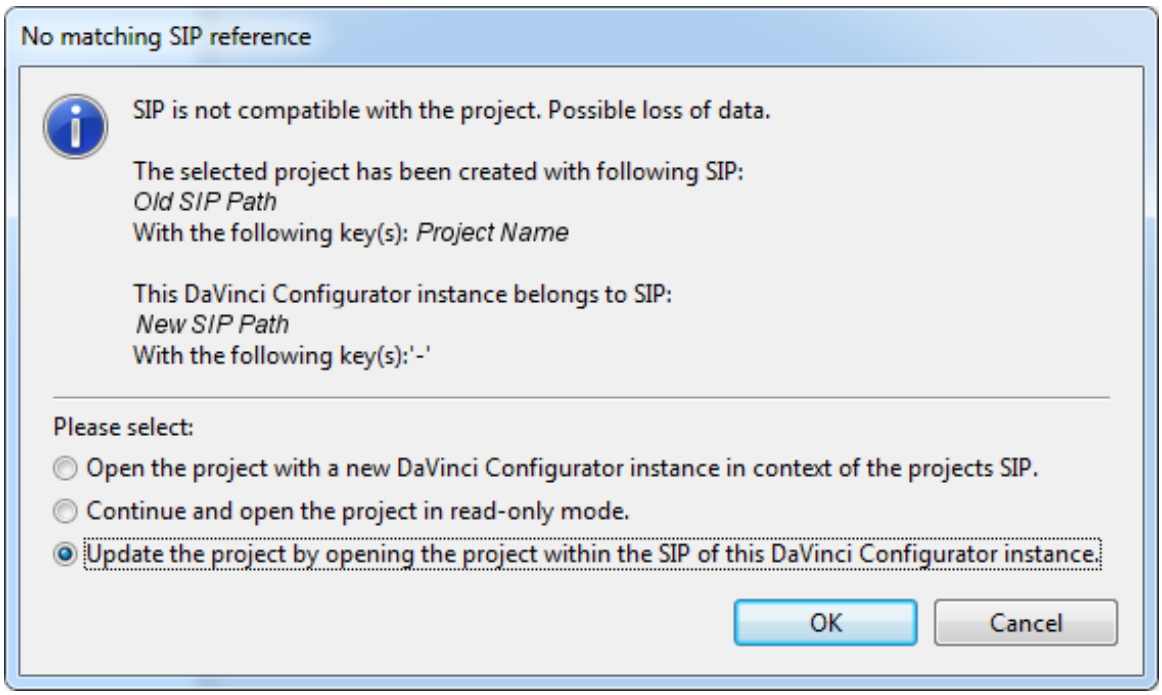
5 Configuration Update

If you get a new delivery with Vector BSW Release x, it is necessary to update some parameter values of your configuration to the new release. There are some steps, that are necessary for most of the updates and there are special steps, that are only necessary for a certain update.

First there are described the necessary steps for any update followed by the special update steps dependent on the release version.

5.1 Release x-1 to Release x (necessary steps for any update)

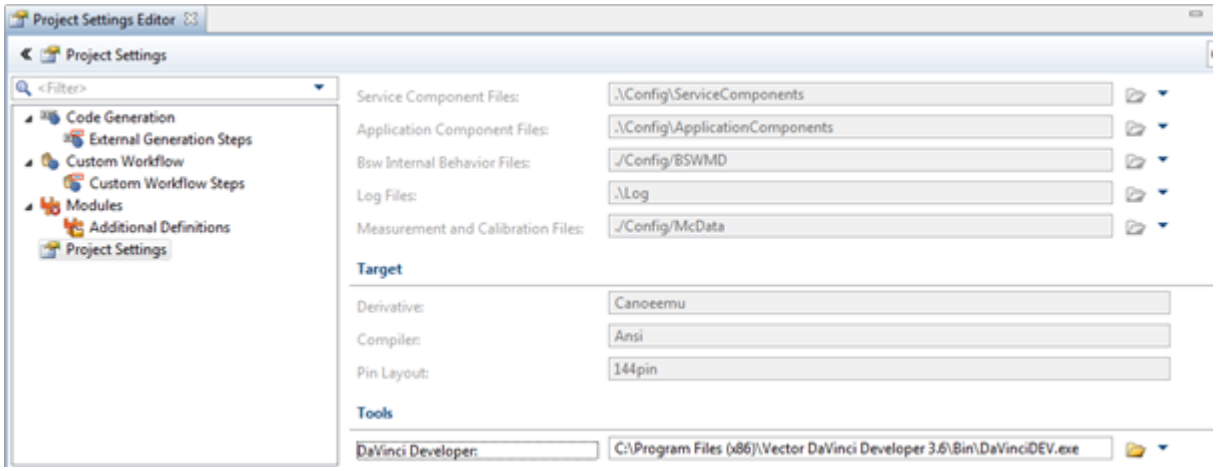
1. Install the latest DaVinci Developer
2. Open your configuration with the DaVinci Configurator of the new SIP .\DaVinciConfigurator\Core\DaVinciCFG.exe and select the following option:



Note

The path within the *.DPA file will be updated to the new SIP.

3. Change the path to the new DaVinci Developer if it was installed to a different location than the former DaVinci Developer version.



Note

C:\Program Files (x86)\Vector DaVinci Developer **<version>**\Bin\DaVinciDEV.exe

4. Handle remaining errors and warning. The following might occur for any update

Errors Invalide/Incorrect Definitions	
> AR-ECUC02008 Invalid multiplicity > AR-ECUC03019 Incorrect definition of configuration element	
Solution	Delete parameter in ECUC file not existing in BSWMD file any more.
Example	Some parameters has been replaced. E.g. **/MICROSAR/**Dem[1]**/DemGeneral/DemEventStorageTrigger** has been replaced by **/MICROSAR/Dem/DemGeneral/DemEventMemoryEntryStorageTrigger**

Errors Inconsistent Connector Prototypes	
> RTE51027 Connector prototype inconsistent. (1 message) > RTE51031 Connector prototype inconsistent. (1 message)	
Solution	Due to changes within the Service Components the ports can be become incompatible Application Software Components. Adapt your Application Software Components with the DaVinci Developer to match the new port definitions. The changes are normally resulting from

[1](Diagnostic Event Manager) The Dem module implements a fault memory. The standardized interface for "DiagnosticMonitors" enables uniform development of manufacturer-independent SWCs. The Dem module is responsible for administering the DiagnosticTroubleCode states, environmental data, and for storing the data in NVRAM. The legal requirements of OBDII / SAE J1979 are also supported.

Errors Inconsistent Connector Prototypes	
	> Correction (the old definition was wrong) > Changed implementation based on AUTOSAR Bugzilla entries / RfC > Changed implementation due to support of newer AUTOSAR Version (e.g. AUTOSAR 4.0.3 to AUTOSAR 4.1.2)

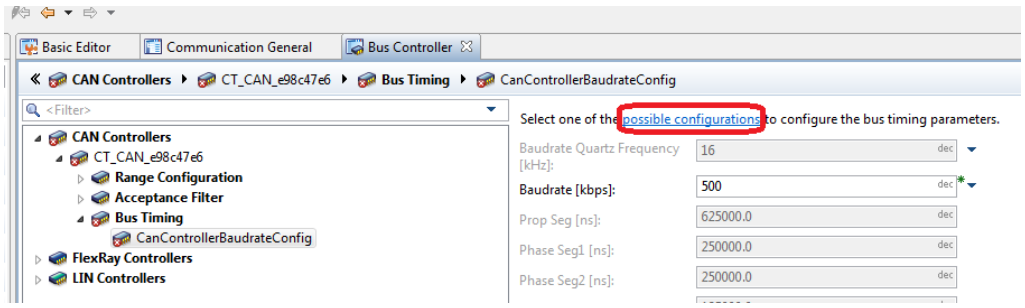
5. Process all further Errors and Warnings given within the Validation View of the DaVinci Configurator
6. Check for updating of the input data base files.
   Even though they might not have changed, the deriving of parameters might have improved. Thus an update now would reduce the changes when receiving changed input data base files.

Cross Reference

Additional information about necessary configuration update steps are described within the Release Notes (**Category: Change**).

The following describes what to do when updating especially from one release to another.

5.2 Release 8 to Release 9

Errors Invalid Baud Rate	
> CAN02012 An invalid baud rate value is configured > CAN02012 invalid baudrate settings: [MANUAL ACTION]: change clock or baudrate. /ActiveEcuC/Can[1]/CanConfigSet/CT_CAN_e98c47e6/CanControllerBaudrateConfig	
Solution	The tool now checks the values of the baud rate configuration, they might have been wrong before. Check valid settings of McuClockReferencePointFrequency referenced by **/MICROSAR/Can_Canoeemu Canoe/Can/CanConfigSet/CanController/CanCpuClockRef** Then select one of the offered baud rate configurations by clicking on 'possible configurations'. 

5.3 Release 7 to Release 8

If you get a new delivery with Vector BSW Release 8, it is necessary to update some parameter values of your configuration to the new release.

After you have updated your project with the new SIP the following steps are necessary:

1. Select ECUC parameter which is not defined by a BSWMD Parameter definition.
   (Error number AR-ECUC03019)
2. Check whether additional parameters exist.
   (Parameters without value definition or with default value)
3. Define value of the new parameter with value from the existing parameter
4. Delete existing parameter

Use the same steps for container updates.

[1](CAN Driver) The CAN Driver abstracts access to the CAN hardware for sending and receiving messages and for switching between controller states (sleep, stop, etc.)

Note

Select all **VectorCommonData** containers, which are marked with an error, use multi-selection in the DaVinci Configurator Basic Editor and delete them via **Delete Container**.

All **Use RTE** switches and **Production Error Detection** switches can be deleted without check.

6 Update DaVinci Tools

Cross Reference

The DaVinci service packs are located at the **Vector Download Center**.
https://vector.com/vi_downloadcenter_en.html

The Update of the DaVinci Tools is described within the installation guide.

Cross Reference

You will find the installation guide document at the Vector Knowledge Base:
https://vector.com/kbp/entry/640/

6.1 DaVinci Configurator Pro

If necessary , update DaVinci Configurator Pro service packs,therefore you have to download them from the Vector Download Center and install them via DaVinci Configuration Service Pack Installer. Select the installed SIPs which should be updated.

6.2 DaVinci Developer

Service Packs of DaVinci Developer are also available in the Vector Download Center. The service packs include an installer to update your DaVinci Developer installation.

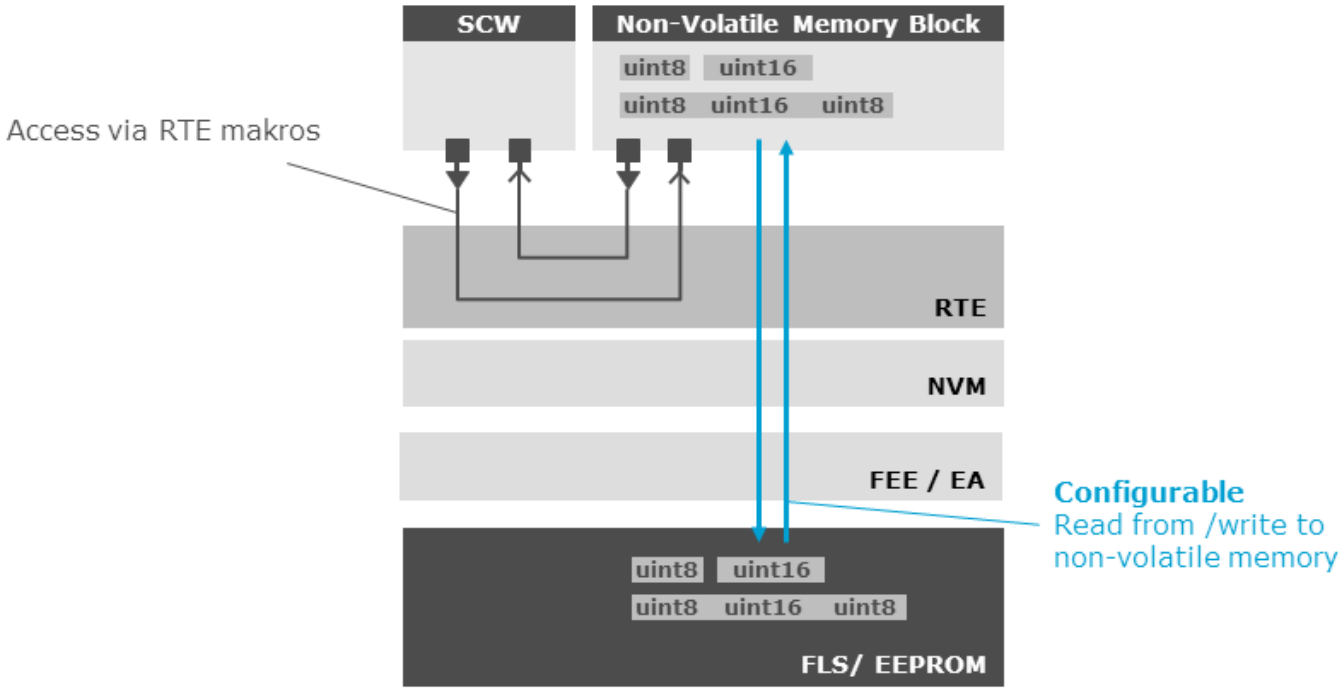
Note

Please note that compatibility is only granted for service packs provided for the same major and minor version.

7 Non-Volatile Memory Block

This is a description how to handle data that has to be stored to non-volatile memory like FLASH or EEPROM.

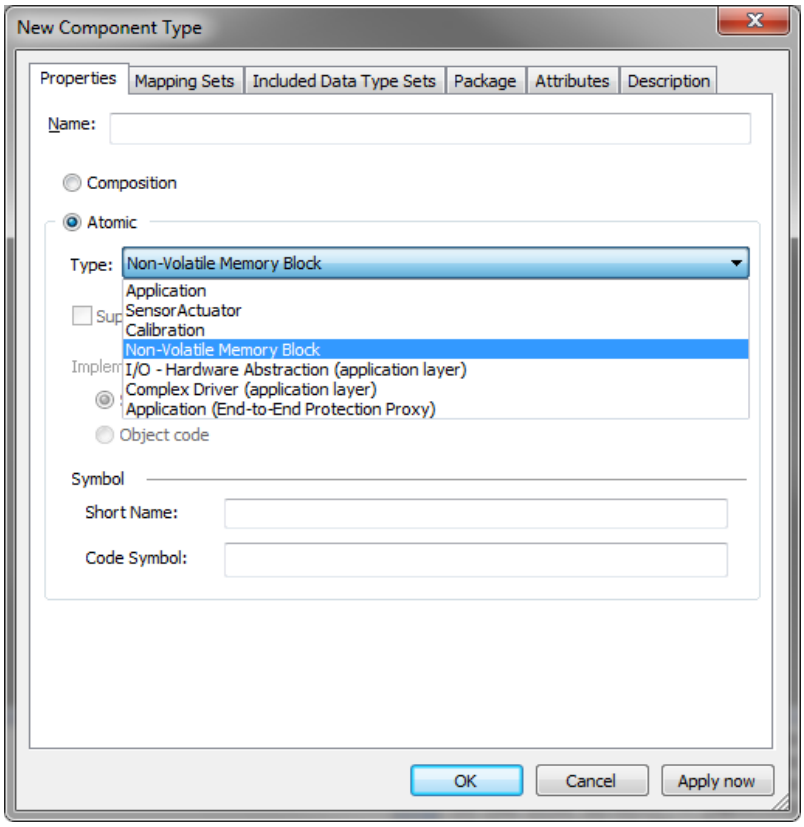
The illustration below shows briefly the concept behind. Data is defined in a Non-Volatile Memory Block. The application SWC has access to this data via the RTE. The data is stored and addressed in RAM. At configuration time you can define which data in which format you need and when and how this data is written to and read from the non-volatile memory. You can also trigger the immediate writing of this data to the FLS or EEPROM.



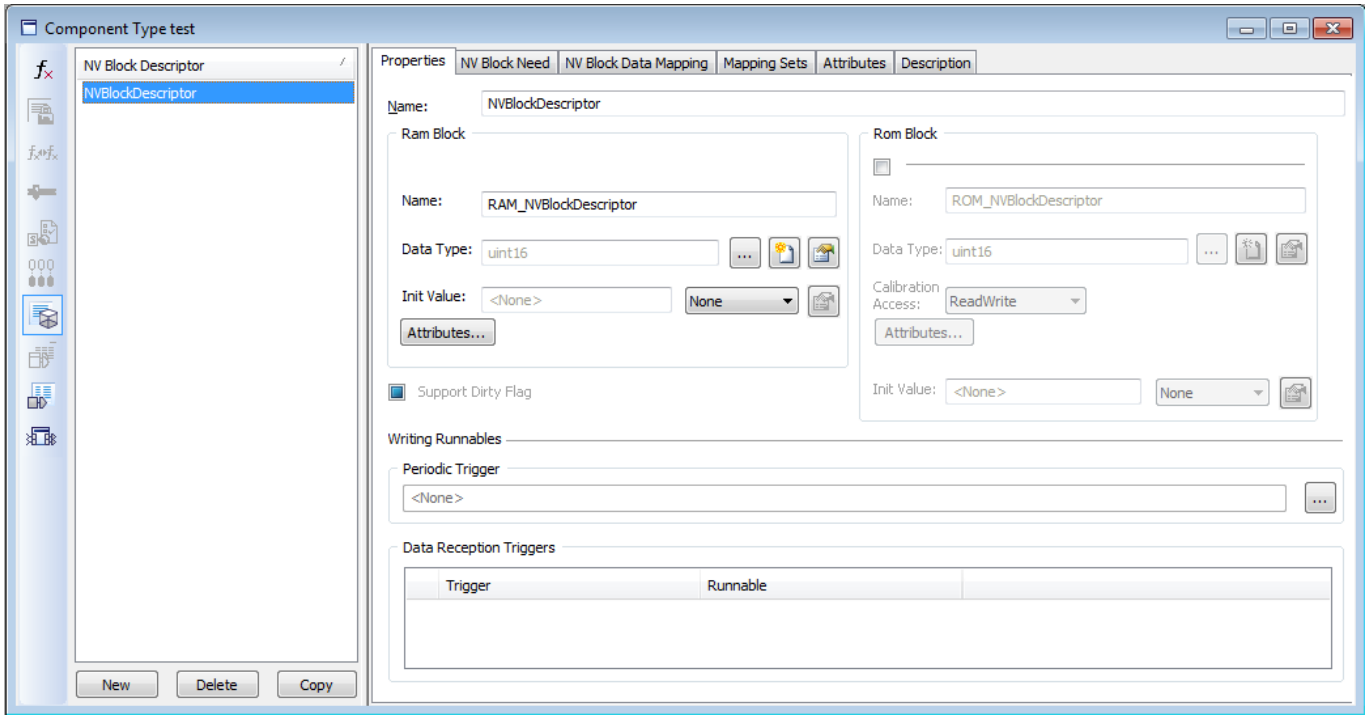
7.1 Configure and use Non-Volatile Memory Block

The following description shows, how the Non-Volatile Memory Block is configured and used.

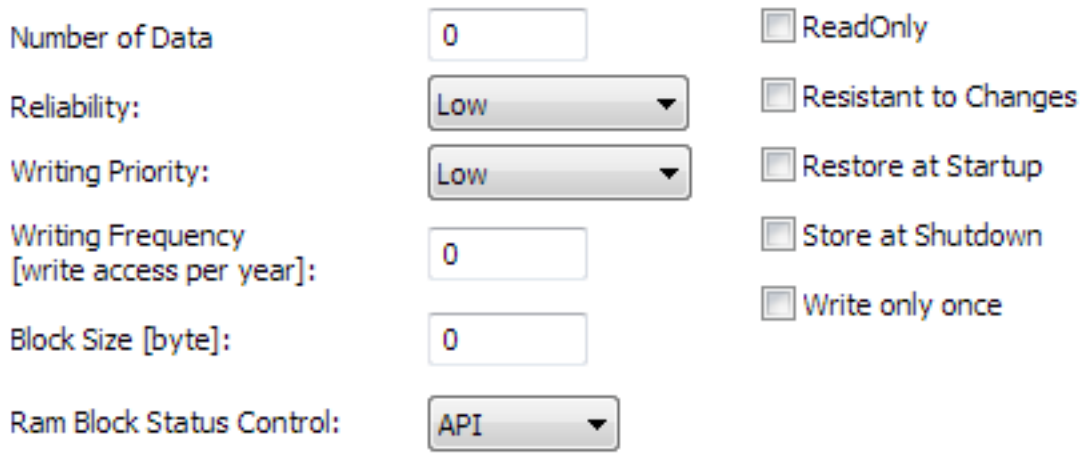
In the Library of DaVinci Developer, add a **New Application Component Type**, select **Non-Volatile Memory Block** as type and give a **Name** to this new component type. Double-click the new **Non-Volative Memory Block** in the **Library** and select **NV Block Descriptors** 



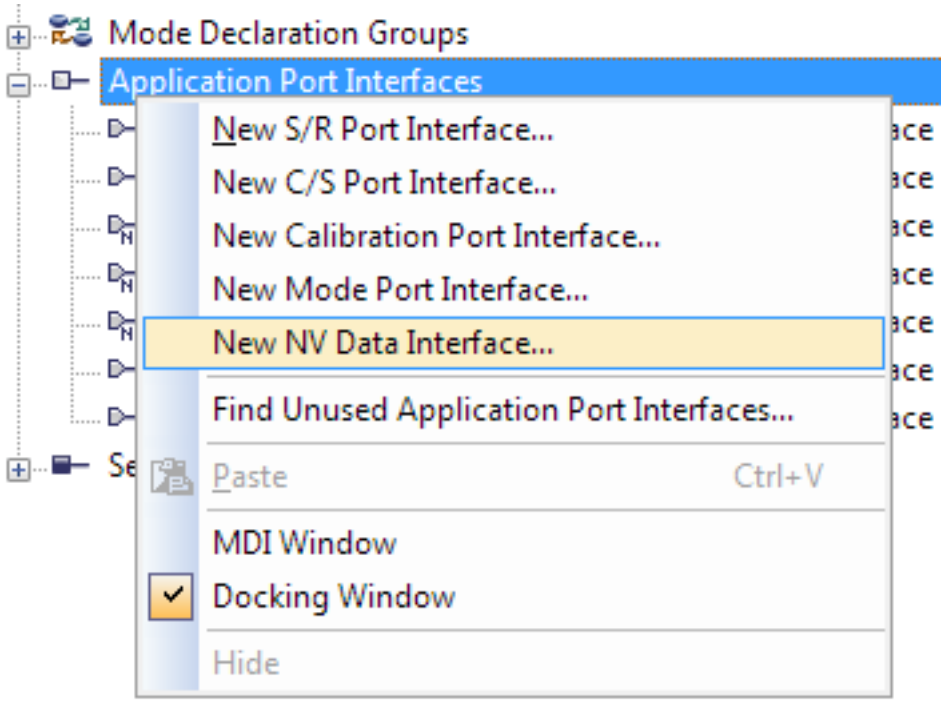
Use **[New]** to define a new NV Block Descriptor. On the Properties tab define the data you want to store to non-volatile memory and its type. This can be a single variable like integer or boolean. Also complex data types like records or arrays can be defined.



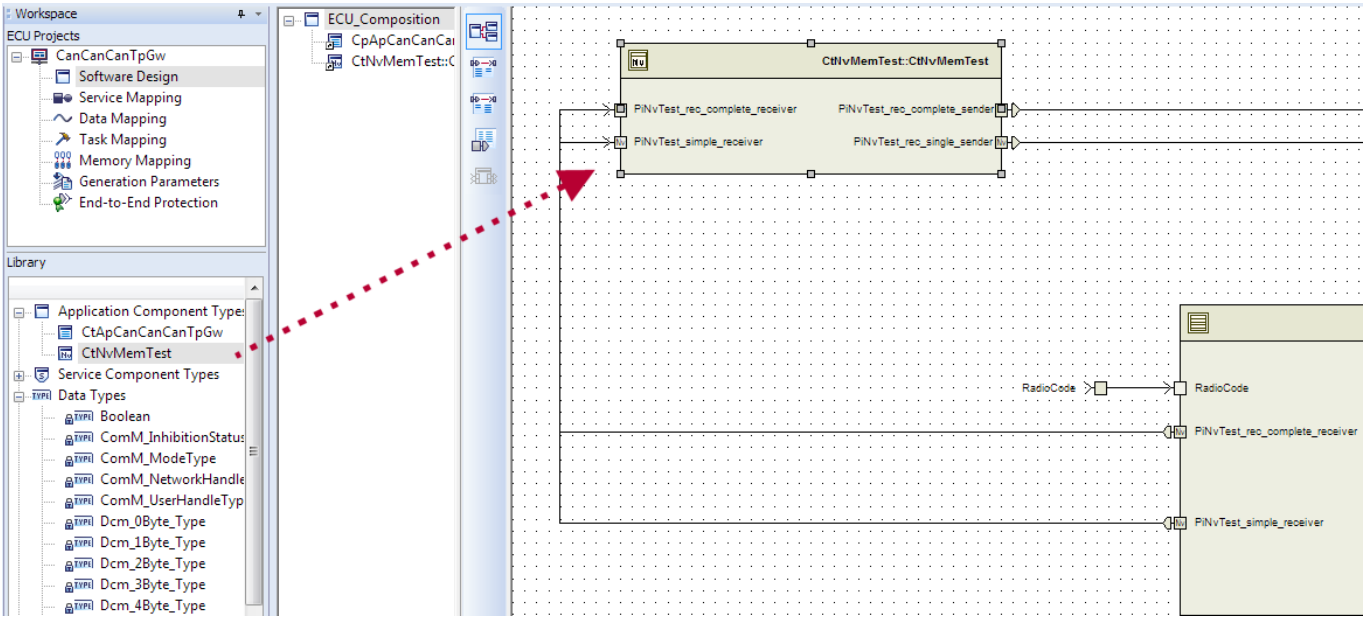
Select the tab NV Block Need and define, when and how your data should be finally written to NV memory (FLS or EEPROM).



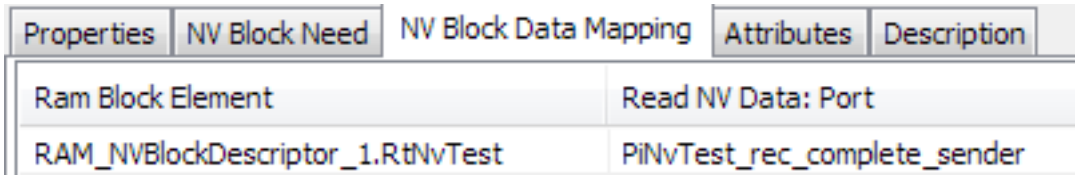
Now add New NV Data Interface… to define a port to access the data defined before. Select suitable Data Element Prototypes.



Add the NV component type to your Software Design (drag'n'drop from Library), assign the NV Data Interface to the NV component prototype and another NV Data Interface to your application SWC. Draw the necessary connectors by hand. Define the Init Values of all ports.



Double-click NV component prototype and perform the internal mapping on the NV Block Data Mapping tab (right-click to open context menu).



7.2 Port Access of your Runnables

What is left now – the access of your application runnable to the **NV Data Interface**(s). Open your application software component, select a runnable and then the tab **Port Access**. Use the **[New]** button to add a read or write data access to the **Non-Volatile Memory Block**.

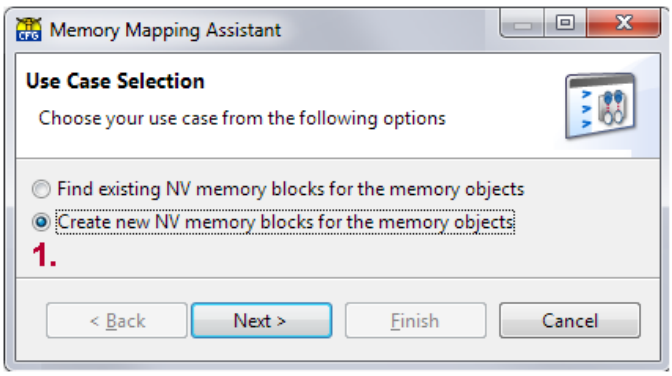
Save your project and switch to DaVinci Configurator Pro.

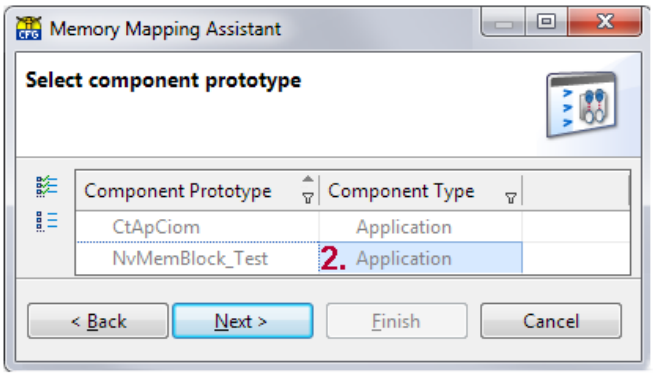
7.3 Memory Mapping in DaVinci Configurator Pro

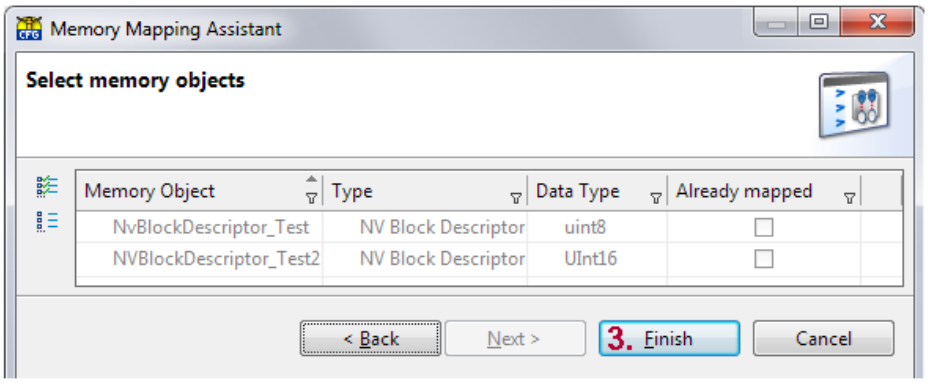
Synchronize the system using the synchronization message in the **Validation** area.

Open **Runtime System**. Click **Add Memory Mapping**.

1. Select **Create new NV memory blocks for the memory objects**.
2. Select the block created before in the DaVinci Developer
3. Get information about unmapped blocks and click **[Finish]**.

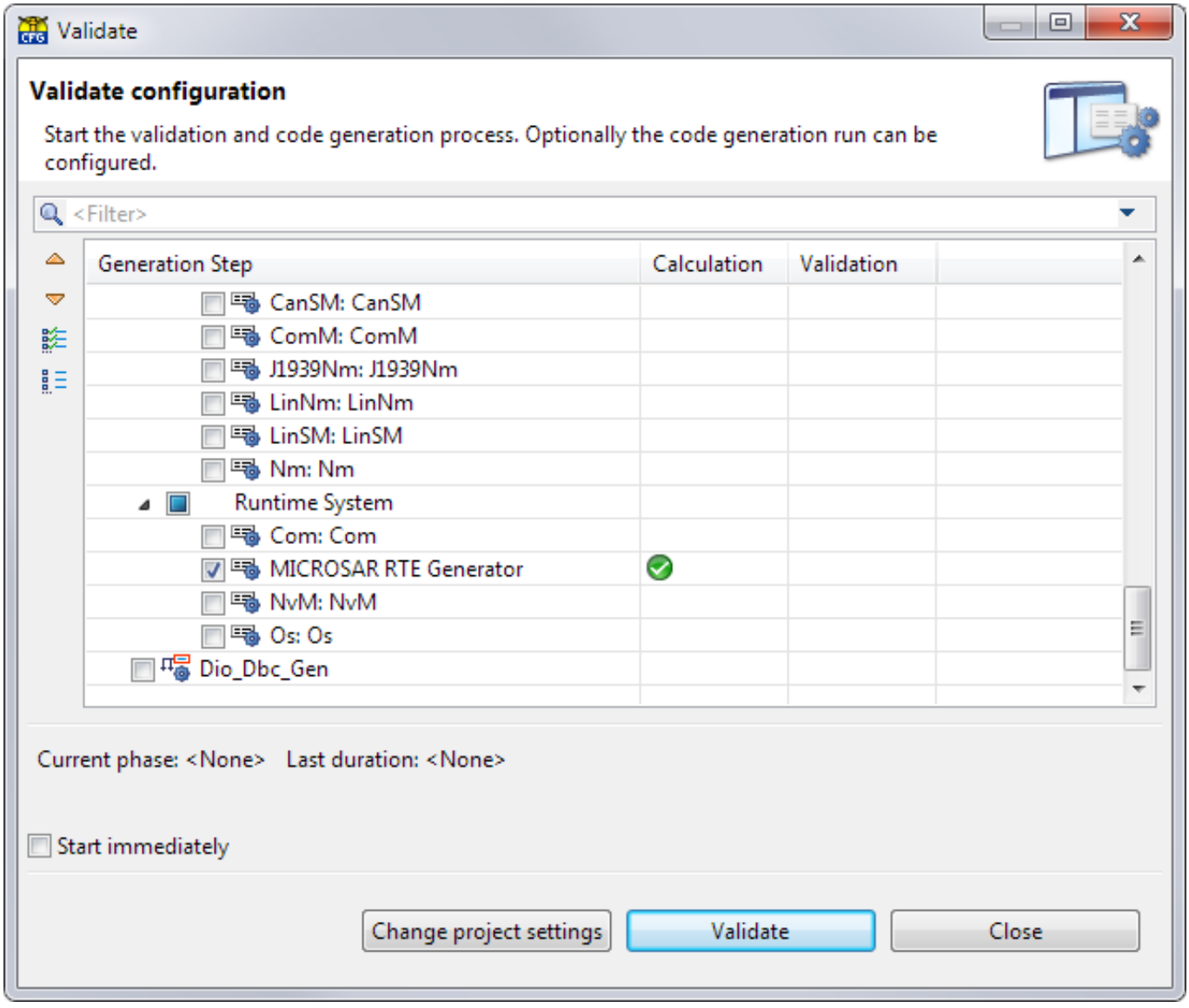

Memory Mapping Assistant
Use Case Selection
Choose your use case from the following options
Find existing NV memory blocks for the memory objects
Create new NV memory blocks for the memory objects
1.
< Back
Next >
Finish
Cancel


Memory Mapping Assistant
Select component prototype
Component Prototype
Component Type
CtApCiom
Application
NvMemBlock_Test
2.
Application
< Back
Next >
Finish
Cancel


Memory Mapping Assistant
Select memory objects
Memory Object
Type
Data Type
Already mapped
NvBlockDescriptor_Test
NV Block Descriptor
uint8
NVBlockDescriptor_Test2
NV Block Descriptor
UInt16
< Back
Next >
3.
Finish
Cancel

7.4 Validate the RTE

1. Open the validation window via 
2. Deactivate all boxes via 
3. Select RTE and click **[Validate]**.



When implementing you runnable code, you can now access the defined variables by using RTE read and write macros.

8 Basic Software Modules

This chapter deals with BSW modules, Basic Software Modules. First they are introduced theoretically then configured with the DaVinci Configurator Pro.

8.1 Generic BSW Modules

Before you start to configure the BSW module with DaVinci Configurator Pro you have to know something about BSW modules.

This chapter introduces to you a generic BSW module and wants you to get a basic understanding of it:

- How it looks like
- Of what it is formed
- How it is configured
- How it works
- What is fix
- What is generated and

some special topics like

- Memory sections and
- Exclusive areas

Cross Reference

Find the detailed description of the BSW module in the folder **Doc|TechnicalReferences** of your delivery.

8.2 What is a BSW Module?

A Basis Software Module (BSW module) consists of:

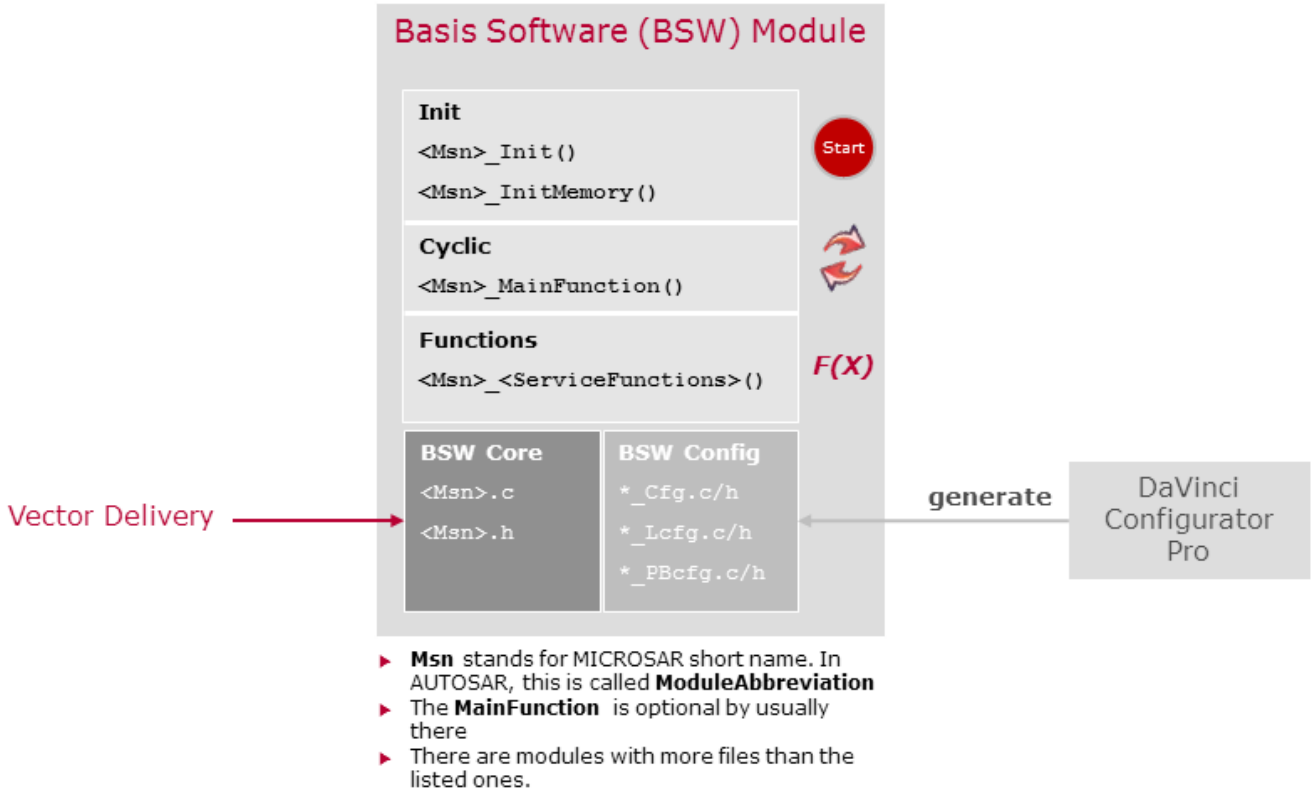
- Static BSW files (algorithms and functions)
- Generated files (data and sometime algorithms and functions)

To get it to work it has to be

- Included
- Initialized
- Its main function has to be called cyclically (in most of the cases)

The mapping of its code and variables can be configured as well as their exclusive areas, to protect the module's internal data.

- Memory Mapping
- Exclusive Areas



Example

Examples for the short names: Dem[1], Dcm[2], EcuM, etc.

[1](Diagnostic Event Manager) The Dem module implements a fault memory. The standardized interface for "DiagnosticMonitors" enables uniform development of manufacturer-independent SWCs. The Dem module is responsible for administering the DiagnosticTroubleCode states, environmental data, and for storing the data in NVRAM. The legal requirements of OBDII / SAE J1979 are also supported.
[2](Diagnostic Communication Manager) The Dcm module implements diagnostic communication according to ISO 14229-1:2006 (UDS). Some diagnostic requests are processed directly in the Dcm (management of diagnostic sessions, reading of error codes, EcuReset, etc.) and some are routed to the SWCs via port interfaces (reading, writing and controlling of data elements within a data identifier, execution of routines, etc.). Legal requirements of OBDII / SAE J1979 are also supported.

8.3 BSW Module Configuration

The amount and properties of configurable parameters is defined within the <Msn>.bswmd file. BSWMD stands for Basic Software Module Description file. This file is read by the BSW configuration tool (DaVinci Configurator Pro) and used to create the Basic Editor. The Basic Editor is the tool expression of all configurable parameters of the BSW module, displayed in a generic way.

Note

In addition to the Basic Editor, the Vector BSW configuration tool DaVinci Configurator Pro provides comfortable views that make configuration of BSW modules very comfortable.

The DaVinci Configurator Pro generates the configuration files for every used BSW module. These are files like

- <Msn>_cfg.c/h,
- <Msn>_Lcfg.c/h,
- <Msn>_PBcfg.c/h

8.4 BSW Initialization

8.4.1 <Msn>_InitMemory()

Most BSW module needs variables that have to be initialized before the call of `<Msn>_Init()`. This can be global or static variables. According AUTSOAR concept, all necessary variables will be initialized by the start-up code. Where this is not done or not possible the function `<MSN>_InitMemory` has to be called as an alternative.

8.4.2 <Msn>_Init()

Every BSW module has to be initialized at start-up. This is done with the start-up of the ECUM. The function to initialize a BSW module is called: `<Msn>_Init()`

Note

Our modules and functions are protected against multiple initializations.

8.5 BSW Module Version (xxx_GetVersionInfo)

Each BSW module provides an API **<MSN>_GetVersionInfo**.This API returns the BSW published information

- BSW version
- Module ID
- Vendor ID

The BSW versions are BCD-coded. The availability of this API can be individually configured for each module.

8.6 Cyclic Calls

8.6.1 <Msn>_MainFunction()

To run and to work, most of the BSW modules (except for those, that are triggered by interrupt), have to be called cyclically with a configured call cycle. The module derives its time base from those cyclic calls. This cyclically called function is called main function and its name is formed like: `<MSN>_MainFunction()`

Note

In most of the cases it is mapped to a cyclic task that is called cyclically triggered by an Alarm of the Operating System.

8.7 Service Functions

Cross Reference

For information about **Client Server Theory**, refer to Data Mapping (section Service Mapping).

8.8 Critical Sections - Exclusive Areas

BSW modules use so-called exclusive areas to protect their resources (module internal data) from concurrency. Each module defines its own exclusive areas, up to 5 can be defined per AUTOSAR definition.

Whether a critical section needs to be handled or not depends on the possibility of a concurrent access onto a given resource of the BSW. It therefore depends on e.g. OS or hardware configuration constraints. If a critical section needs to be handled, different measurements are possible, ranging from a global interrupt lock to the usage of OS semaphores.

Example

As an example, a definition of an exclusive area could look like:

```
SchM[1]_Enter_Com[2]_COM_EXCLUSIVE_AREA_2();

com_TxModeHdlr_DelayTimeCnt[i]--; /* Protected object */

SchM_Exit_Com_COM_EXCLUSIVE_AREA_2();
```

There are different locks available i.e. different interrupt sources could be locked. Global interrupt, peripheral interrupt like CAN or no lock at all.

Cross Reference

Refer to the technical reference document of the BSW module to get information how exclusive areas are defined and used.

8.8.1 Memory Section

Every generated code and variable of the <MSN> modules is enclosed by #defines. Via the template file `compiler_memMap.h` you can define the mapping of each single section of a <MSN> module.

Cross Reference

See more about the memory mapping via `memMap.h` in the technical references of each <MSN> module.

8.8.2 Switch <MSN> Modules Off

Most of the modules can be switched off via: `<Msn>_Shutdown`

[1](BSW Scheduler) The SchM module is integrated in the (RTE, calls the periodic "main" function of the individual BSW modules, and provides functions for critical sections. For the distribution of the BSW (master satellite concept) via partitions and core boundaries, the SchM provides nearly identical communication interfaces like the (RTE.

[2](Communication) The Com module provides a signal-based data interface for the RTE. It places signals in messages and sends them according to the defined send type. The module contains various notification mechanisms for receiving signals. It also manages initial values, update bits and timeouts on the signal level. In multi-channel ECUs, the integrated signal gateway offers the ability to route signals between the communication buses.

9 Command Line Parameters Of The DaVinci Configurator

9.1 Common parameters

Option	Arguments	Mandatory	Description
-h, --help	-	no	Shows command line usage information (use cases and available options).
--verbose	n <ERROR> <WARN> <INFO>	no	Enables the verbose output of logging messages. Possible arguments: ERROR, WARN, INFO Optional Argument: The Log-Level for the verbose output. Default: ERROR
-l, --logfile	1	yes	Set the file to log the Console output into.

Mandatory parameters missing. Please choose a use case. E.g. specify a Project and **--generate** to generate the Project.

9.2 Command Line Interface Use Cases

The DaVinci Configurator command line interface has different use cases:

9.3 Usecase With GUI (only DaVinciCFG.exe)

Opening the GUI is the default use case.

Option	Arguments	Mandatory	Description
-p, --project	1 <DPA-FILE>	no	Specifies the absolute path to the DPA project file to be opened in the GUI immediately
-clean	-	no	Start the DaVinci Configurator with clean eclipse cache, to solve Eclipse caching problems when DaVinci Configurator versions 5.10.xx and older are used on the same PC with DaVinci Configurator version 5.11.xx and younger.

9.4 Use Case DaVinci Configurator CodeGenerator

The identifying option for this use case is -g.

The DaVinci Configurator generates the BSW modules for a given ECU project.

Usage Example

DVCfgCmd --project Project.dpa --generate

DVCfgCmd --project Project.ecuc.arxml --generate --modulesToGenerate "/MICROSAR/Det[1]

Option	Arguments	Mandatory	Description
-p, --project	1 <DPA-FILE>	yes	Specifies the absolute path to the DPA project file or the ECUC
-g, --generate or -v	-	yes if-v is not used	Generate the given project specified in <DPA_FILE>. The -g option also executes --validate At least one of the options -g or -v must be specified.
-v, --validate or -g	-	yes if-g is not used	Validate the given project specified in <DPA_FILE>. At least one of the options -g or -v must be specified.
-m, --modulesToGenerate	1	no	Specifies the module definition references, which should be generated by the -g switch. Separate multiple the modules by a ','. Syntax: --modulesToGenerate "<identifier>,<identifier> ..." The identifier of a module is the AUTOSAR Definition path. E.g. /MICROSAR/Nm[2], /MICROSAR/CanIf[3] --modulesToGenerate "/MICROSAR/Nm,/MICROSAR/CanIf".

[1](Development Error Tracer) The Det module supports error debugging during software development. It provides an interface for error notification, which is called by the individual BSW modules in case of error.
[2](Generic Network Management Interface) The Nm module offers a general and network-independent interface for accessing the bus-dependent network management modules ((CanNm, (LinNm, (UdpNm and (FrNm). In addition, the module handles synchronous, inter-network shutdown of the communication system in coordination with the other ECUs.
[3](CAN Interface) The CAN Interface offers abstracted (PDU-based) access to the CAN Driver. It controls the CAN Driver ((Can) as well as the transceiver driver ((CanTrcv).

Option	Arguments	Mandatory	Description
			You could also pass the shortname of the module, like "Rte[1]", but be aware, if multiple modules have the same shortname, all modules will be generated!
-x, --modulesToExclude	1	no	Specifies the module definition references, which should not be generated by the -g switch. Separate multiple the modules by a ','. Syntax: --modulesToExclude "<identifier>,<identifier> ..." The identifier of a module is the AUTOSAR Definition path. E.g. /MICROSAR/Nm, /MICROSAR/CanIf --modulesToExclude "/MICROSAR/Nm,/MICROSAR/CanIf" You could also pass the shortname of the module, like "Rte", but be aware, if multiple modules have the same shortname, all modules will be excluded!
--extGenStepsToGenerate	1 <GEN_STEPS>	no	Specifies the External Generation Steps, which should be generated by the -g switch. If empty (""), then no External Generation Steps is executed. Syntax: --extGenStepsToGenerate "<identifier>,<identifier> ..." The identifier of a step is the step name. Separate multiple steps by a ','. E.g. MyStep,DioExtStep,SwcGenStep --extGenStepsToGenerate "DioExtStep,SwcGenStep"
--swcsToGenerate	<SWCs>	no	Specifies the applications software components (SWC) a template should be generated for. Separate multiple SWCs by a ','. Syntax: --swcsToGenerate "<iden-

[1](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

Option	Arguments	Mandatory	Description
			tifier>, <identifier>..." The name is the ShortName of each SwComponentType used within the ECU composition E.g. SWC_A, SWC_B --swcsToGenerate "SWC_A, SWC_B"
--genArg	n <GEN_ARGS>	no	Passes arguments to the specified code generators. Syntax --genArg <DEFINTION>:<Argument> or --genArg="<DEFINTION>:<Argument>" The Definition is the AUTOSAR Definition path of the module to generate. E.g. /MICROSAR/Nm, /MICROSAR/CanIf You could also pass the shortname of the module, like "Rte", but be aware, if multiple modules have the same shortname, all modules will get the argument! You can pass multiple --genArg parameters. E.g. --genArg /MICROSAR/Nm:DoSomethingSpecial=true --genArg="Rte:DoAnotherThing=true" Multiple arguments for one definition can be passed in one --genArg argument or in several. If several --genArgs are used for the same definition, the values are concatenated with a blank as separator. E.g. --genArg /MICROSAR/Nm:Arg1=true Arg2=false or --genArg /MICROSAR/Nm:Arg1=true --genArg /MICROSAR/Nm:Arg2=false
--saveProject	-	no	Saves the project after validation or generation. Generators may modify the project in the calculation phase before the actual generation has started. This option will persist these modifications.
--synSystemDescription	-	no	Performs the synchronization of the System Description after loading the project. This might modify the project, so please consider also the usage of the --saveProject option.

Option	Arguments	Mandatory	Description
--genType	1 <GEN_TYPE>	no	Specifies the generation process type (""NORMAL"" => Real Target or ""VTT"" => vVIRTUALtarget) Syntax: --genType <GEN_TYPE> E.g. NORMAL If no --genType is set, the default depends on the <TargetType>-setting from dpa file.
--keepGenTempFiles	-	no	Keeps the temporary files created during generation after the generation process is finished.

The options **--generate** and **--validate** are mutual exclusive but one of them has to be specified.

9.5 Use Case DaVinci Configurator Execute Converter

The identifying option for this use case is -g.

The DaVinci Configurator converts the ECU configuration.

Usage Example

DVCfgCmd --project Project.dpa --convert

DVCfgCmd --project Project.ecuc.arxml --convert

Option	Arguments	Mandatory	Description
-p, --project	1 <DPA-FILE>	yes	Specifies the absolute path to the DPA project file or the ECUC file.
-c, --convert	-	yes	Convert the given project specified in <DPA-FILE>.
--convertArg	n <CONVERT_ARGS>	no	Passes arguments to the specified converter. Syntax: --convertArg=""<NAMESPACE>:<Argument>"". The Namespace is the namespace defined by the converter

9.6 Use Case EcucUpdater (without GUI)

The identifying option for this use case is -u.

Usage Example

DVCfgCmd -u D:\TEMP\MyProj\MyProject.dpa

DVCfgCmd -u D:\TEMP\MyProj\MyProject.dpa -e D:\TEMP\MyProj\extract1.arxml,D:\TEMP\MyProj\extract2.arxml --ecu myInstance

Option	Arguments	Mandatory	Description
-u, --updateProject	1 <DPA-FILE>	yes	The DPA file of the project which shall be updated
-e, --extract	1 <FILE>	no	The system extract file(s). Separate multiple files by a ','
--ecuInstance	-	depend	The Ecu instance name, mandatory for system description. Not mandatory for system extract.
--psc	1 <FILE>	no	Path to the Project Standard Configuration file(s). Separate multiple files by a ','
--odx	3 <ODX-FILE> <ECU> <VARIANT>	no	Path to ODX file, ODX ECU, ODX VARIANT
--csv	1<FILE>	depend	Path to the State Description file If an ODX 2.0.1 Diagnostic Description file is used, an additional State description will be required.
--did	-	no	Import DIDs and RIDs as single signal. This settings is only relevant for ODX input files.
--cdd	3 <CDD[1]-FILE> <ECU> <VARIANT>	no	Path to CDD file, CDD ECU, CDD VARIANT
--patch	1 <FILE>	no	Diagnostic Description Patch File

[1](Complex Drivers) The Complex Drivers are software modules that are not standardized by AUTOSAR. They have access to other BSW modules, the RTE and direct hardware access. For example, these modules are not standardized communication drivers for SPI or legacy software.

Option	Arguments	Mandatory	Description
--onlyEcuc	-	no	If no argument is passed, the complete update is performed (including DaVinci Developer workspace). If --onlyEcuc is set, only ECUC is updated, without the DaVinci Developer workspace.

The options **--cdd** and **--odx** are mutual exclusive.

At least one of **--cdd**, **--odx** and **--extract** must be specified.

9.7 Use Case DaVinci Configurator Exporter (without GUI)

The identifying option for this use case is --exportDir.

The DaVinci Configurator exports different types of AUTOSAR arxml files. For example files per variant in a variant project.

Usage Example

DVCfgCmd --project Project.dpa --exportDir ./exportDir --exportPostbuildVariants

Option	Arguments	Mandatory	Description
-p, --project	1 <DPA-File>	yes	Specifies the absolute path to the DPA project file
--exportDir	1 <DIR>	yes	Specifies the directory to export the data into.
--exportPostbuildVariants	-	yes	Export the Post-build variants into. This will export the Active-Ecuc and miscellaneous data - Active Ecuc export into one file (even for split DPA-projects) per variant <project-name>.<variant-name>.ecuc.arxml - Miscellaneous data into one file per variant. The files contain all data of the project except: * ModuleConfigurations, ModuleDefinitions * BswImplementations, EcuConfigurations * Variant information like EvaluatedVariantSet <project-name>.<variant-name>.misc.arxml
--listAvailableExporterIds	-	no	
--exportWithExporterId	1	no	Exports the loaded project

Option	Arguments	Mandatory	Description
	<EXPORTER_ID>		with the selected exporter. This will create a file named Exported_ <EXPORTER_ID>.arxml in the folder specified by --exportDir. An <EXPORTER_ID> could be retrieved with the --listAvailableExporterIds argument.

All options starting with --export (except --exportDir) are mutual exclusive.

At least one --export option or --listAvailableExporterIds must be specified.

9.8 Use Case DaVinci Configurator Sign Script

The DaVinci Configurator creates a signature for workflow scripts.

Usage Example

DVCfgCmd --sign MyScript.py

Option	Arguments	Mandatory	Description
--sign	1 <SCRIPT_FILE>	yes	Specifies the script file so sign.

9.9 Return Codes

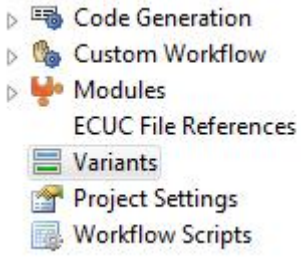
Name	Integer-Value	Description
EXIT_OK	0	ALL_OK in BaseEcuC, Updater and Gen: Configuration loaded without errors, generated without errors.
EXIT_NOT_OK	1	Exit object indicating abnormal termination
EXIT_NO_WORKSPACE_PATH	2	Indicating no workspace path
EXIT_NO_ACTION	3	Indicating no action specified
EXIT_INVALID_SIP	4	Configuration could not be opened.
EXIT_NO_PARAMETERS	5	Indicating missing parameters
EXIT_INVALID_COMMANDLINE_ARGS	6	Invalid generation paths.
EXIT_UNHANDLED_EXCEPTION	7	Unexpected error occured.
EXIT_BASE_ECUC_GENEARTOR_ ERROR	10	
EXIT_UPDATER_ERROR	11	

10 Variant Handling

10.1 Define Criterion and Variants

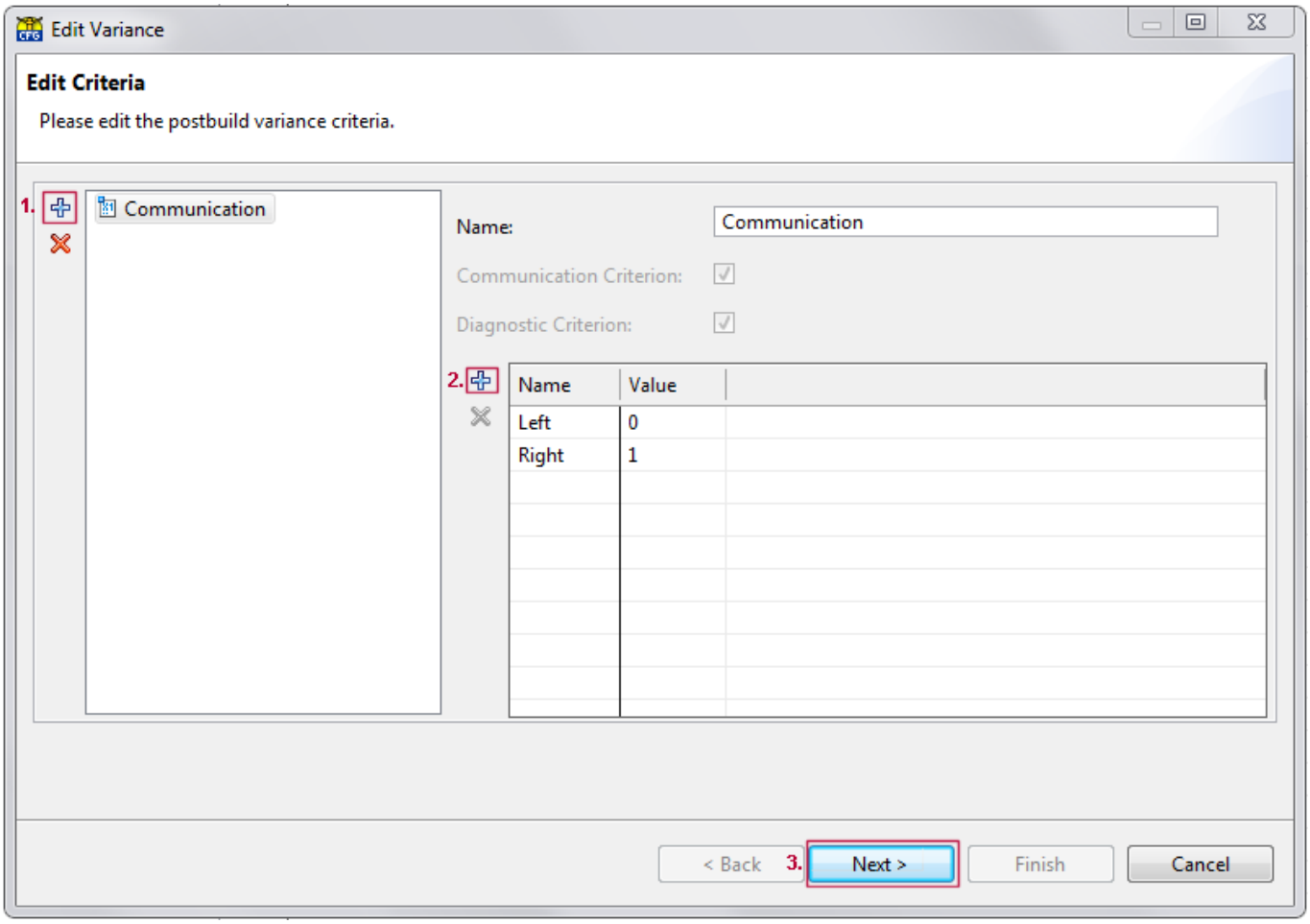
If you use variant handling within your project you have to define your variants first.

After you setup your project, open the **Project Settings Editor** via settings icon[1] and select **Variants**.



Select the Edit Variance icon[2] to add criterion, define criterion values and map criterion values to variants.

1 
2 

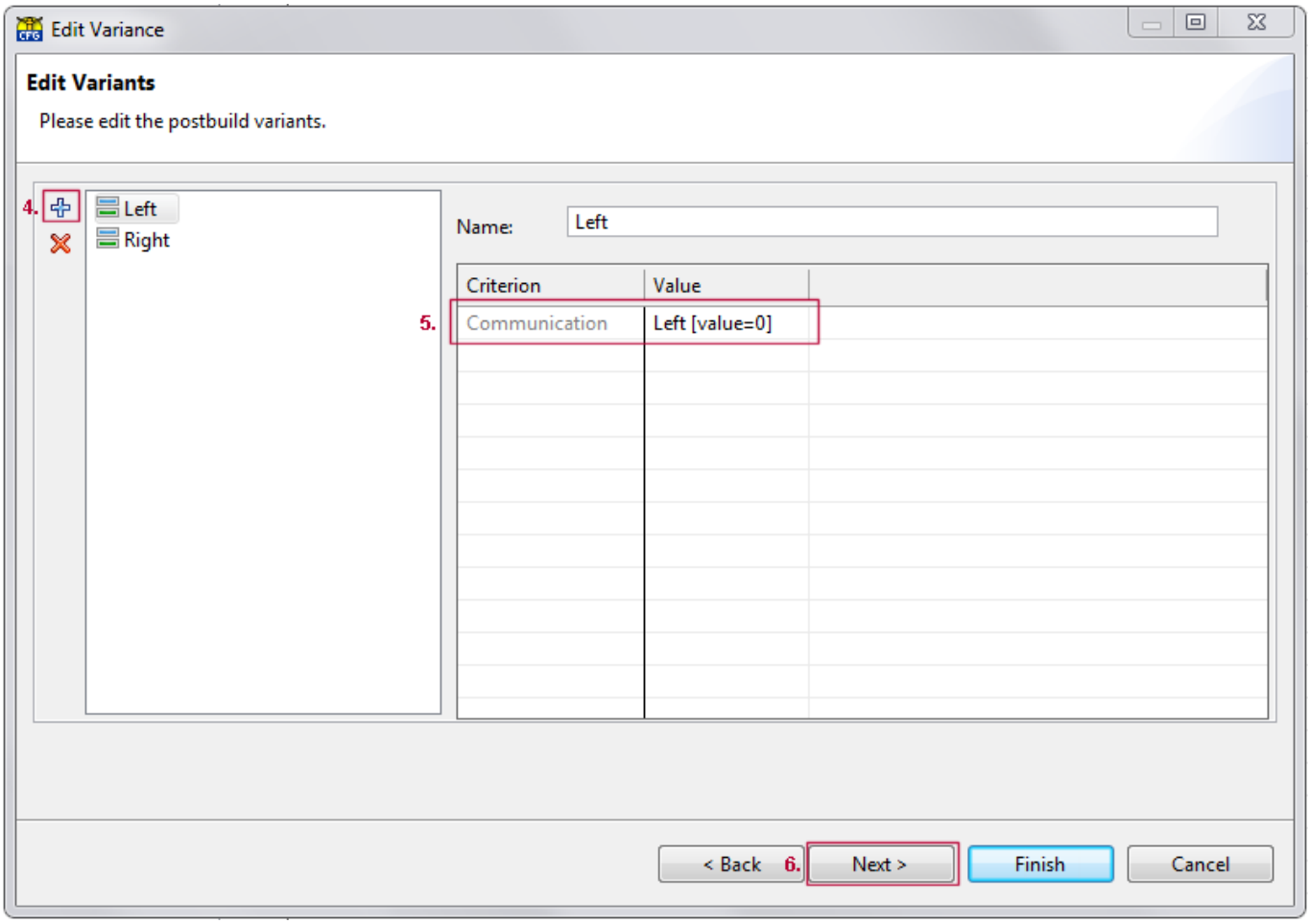


1. Add **[+]** new Criterion.

Note

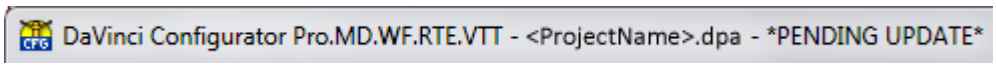
The DaVinci Configurator Pro is currently limited to a single criterion. This criterion can be used for **diagnostic** and **communication** variance.

2. Add **[+]** one criterion value for each variant.
3. Go on with **[Next >]**



4. Add **[+]** your Variants
5. Define Criteria Value for your Variants
6. Go on with **[Next>]** to see a summary of your definitions or **[Finish]** to come back to Variants view.

After finishing variant definition, the tool enters to **PENDING UPDATE** project state.



Note

The DaVinci Configurator Pro is currently limited to a single criterion. This criterion can be used for **diagnostic** and **communication** variance.

10.2 Add and Assign Input Files to Variants

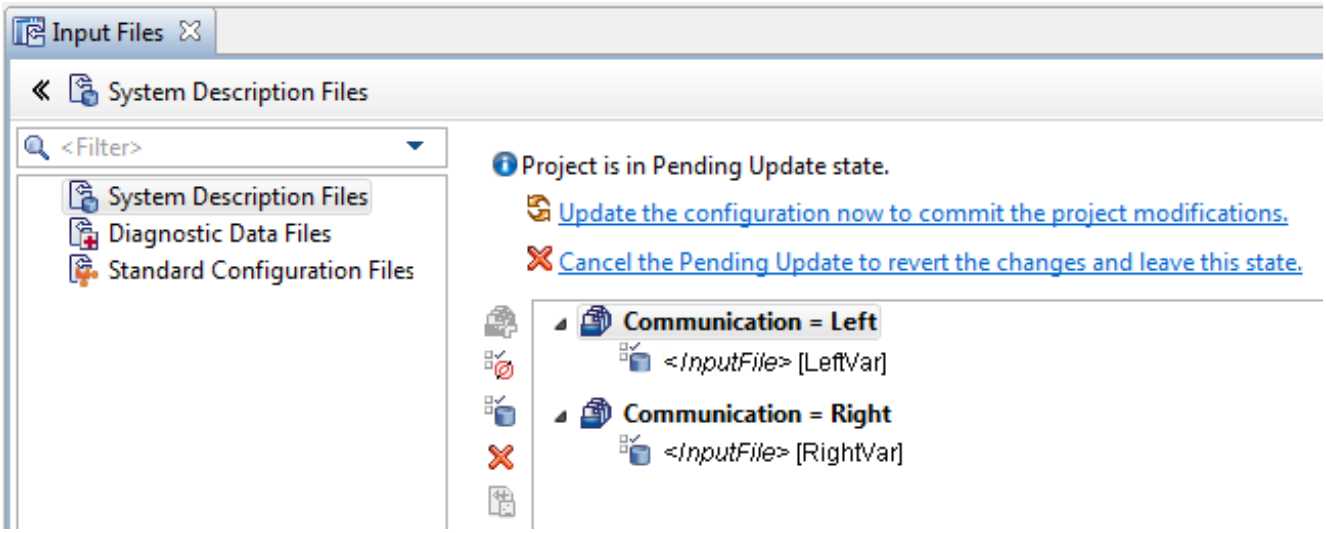
Open Input Files editor via **Project | Input Files** or use the input file button [1] of DaVinci Configurator Pro toolbar.

Select **System Description Files** and add your Input Files via Add File Sets icon[2]. The Add File Set Assistant opens.

Note

The following step is necessary for each variant.

Select the **Post-Build Criterion Value** (Variant) your input file will be valid and go on with **[Next >]**. Add **[+]** your input file(s), select your ECU Instance and confirm with **[Finish]**.



After adding and assigning your input files our project is still in PENDING UPDATE state.

Select **Update the configuration now to commit the project modifications** to start update process.

1 

2 

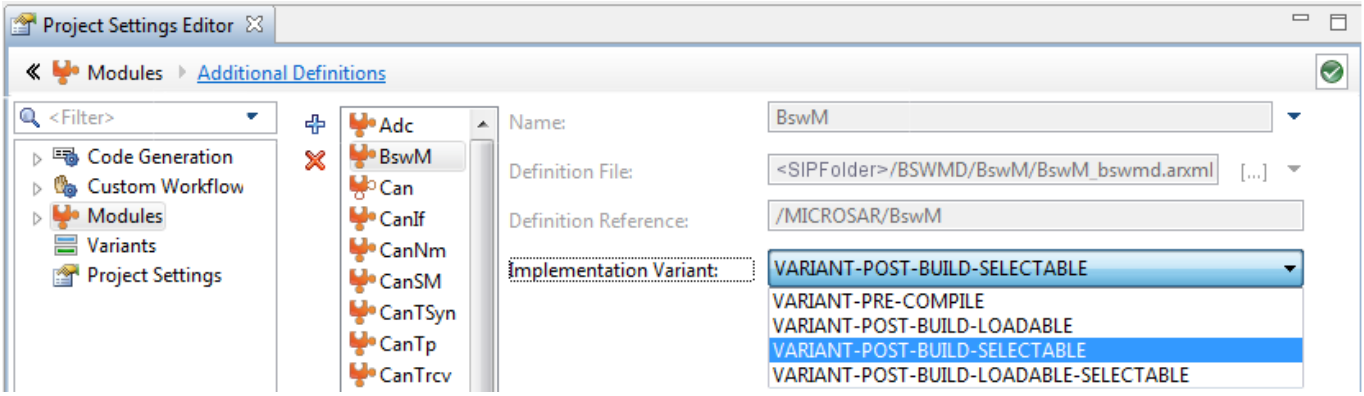
Cross Reference

What happens while Update Process? See section STEP2 Define Project Settings on page 32.

10.3 Define Variance for BSW Modules

Open Project Settings Editor via **Project** | **Project Settings** or use project settings icon [1] at DaVinci Configurator Pro toolbar.

Select **Modules** to see all modules currently activated for your project.



For each BSW module define if it supports variance or not. Therefore choose **Implementation Variant**.

- **VARIANT-POST-BUILD-SELECTABLE**
  No post-build loadable update of the configuration at post-build time. All variants are configured at pre-compile time.
- **VARIANT-POST-BUILD-LOADABLE-SELECTABLE**
  post-build loadable update of the configuration data is supported using MICROSAR Post-Build Loadable. This feature required dedicated licensing.

Note

Multi selection is possible within module view of the DaVinci Configurator Pro, this enables you to set variance for several BSW modules at the same time.

10.4 Configure and Validate BSW

The DaVinci Configurator Pro highlights variant parameter and container using a brown [V]. When changing the configuration it has to be considered whether the change shall be applied to all variants or if

12

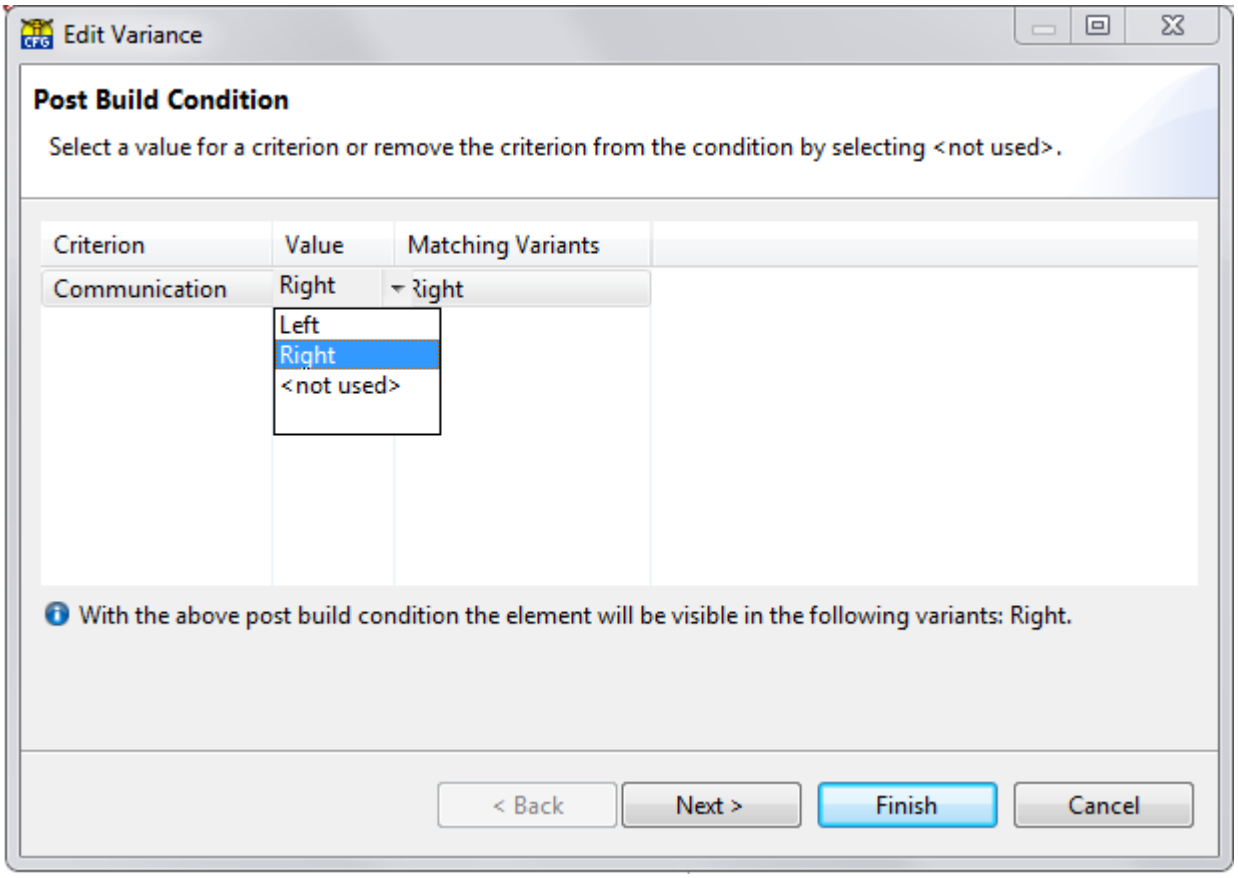
2

the change shall be done only for one variant.

Note

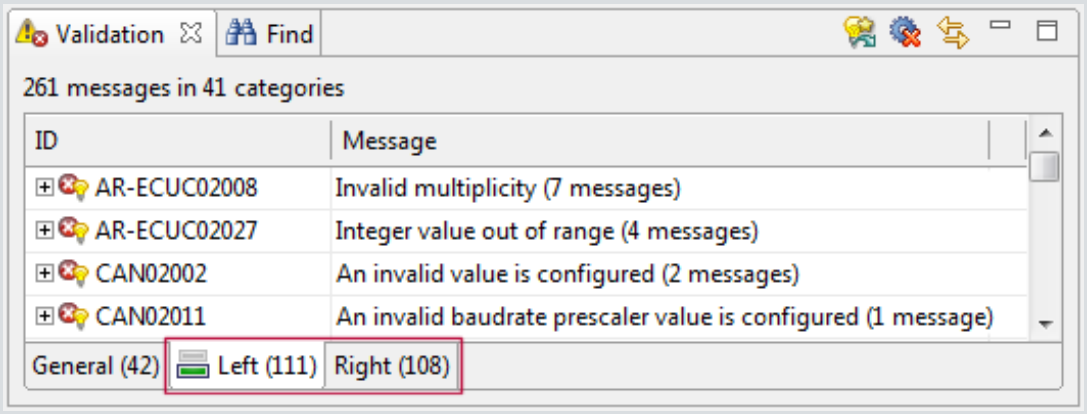
You can select your **Active Variant** at the DaVinci Configurator toolbar.

To define a value for a single variant only, right-click on parameter, select **Edit variance** and define variant (**Value**).



Note

The validation view includes views for each variant.

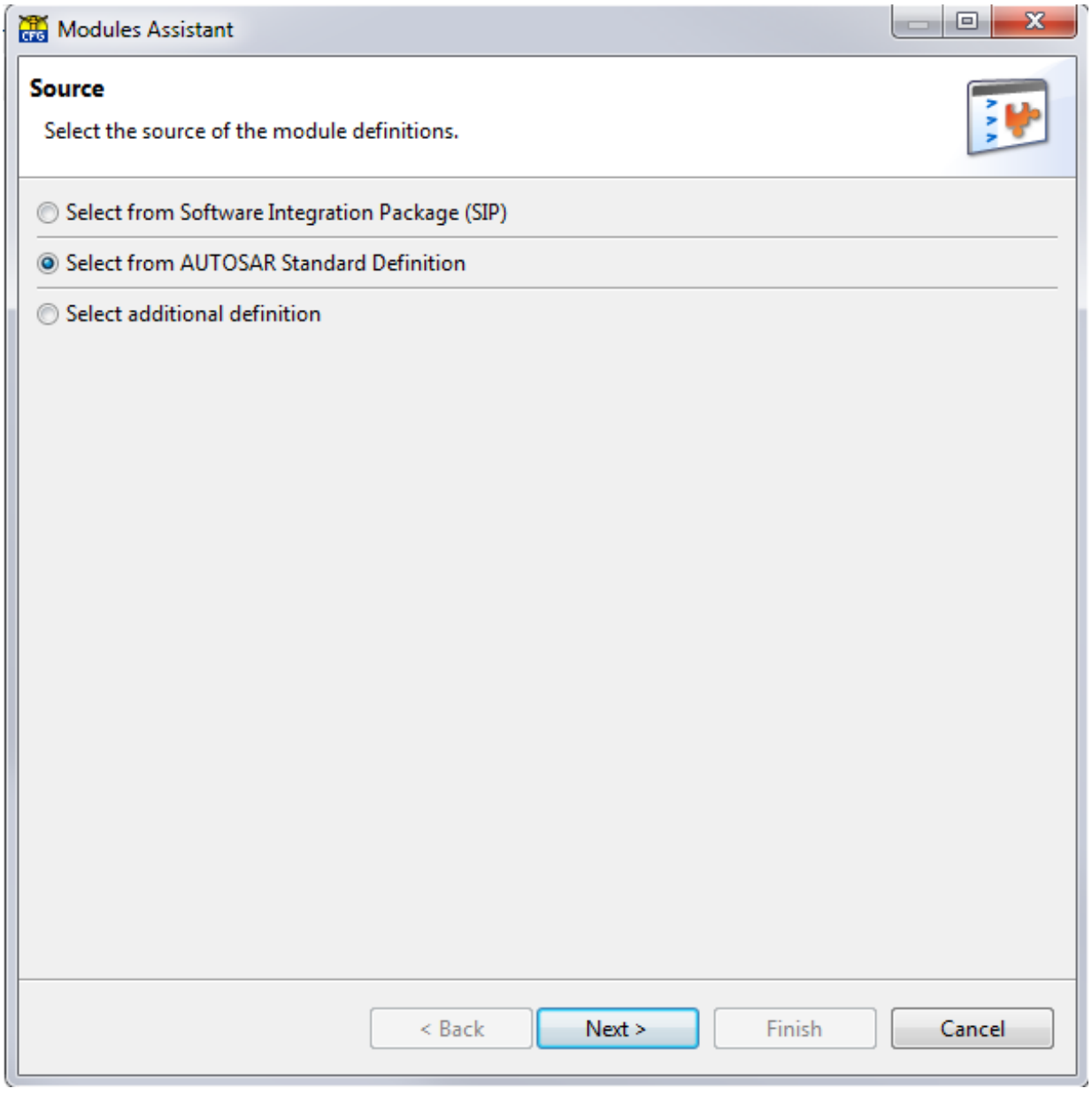


11 Add Module Stubs From AUTOSAR Definition

In some cases it is necessary to add a module, which is not delivered within your SIP, e.g. a delivered module (DCM) needs references and interfaces to a not delivered module (NVM), otherwise the validation fails.

To avoid validation and generation errors you have to add the required module according to AUTOSAR standard definition. Therefore open **Project Settings Editor**, select **Modules** and open Modules Assistant via add **[+]**.

Choose **Select from AUTOSAR Standard Definition** and go on with **[Next]**, select the required module and confirm with **[Finish]**.

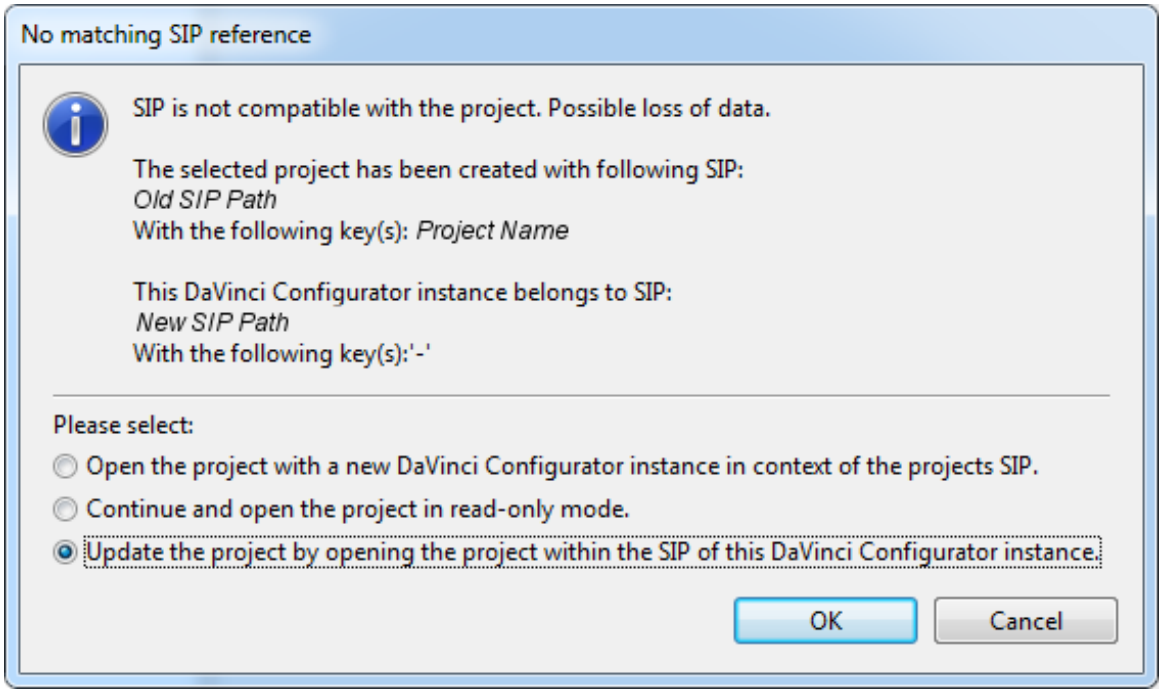


Now the module is available at the **Basic Editor**. The module based on AUTOSAR Standard definition needs to be configured basically with all necessary definitions required by the MICROSAR modules within your project.

12 SIP Update

You received a newer version of your SIP. This requires an update of your current project. Therefore open the project with the new version of the DaVinci Configurator Pro. The **DaVinciCFG.exe** is located within the folder *<UpdateSIP>\DaVinciConfigurator\Core*.

The DaVinci Configurator Pro identifies that the project was created with an older SIP and displays the following message. Select **Update the project by opening the project within the SIP of this DaVinci Configurator instance** and click **OK**.



The update will performed. Wait until the DaVinci Developer project opens. Click **[Accept]** when the warning message appears. Click **[Import]** in the next dialog.

The Signal Import Mode dialog opens, select **Use import mode for all remaining objects** and click **[OK]**. The dialog proposing to **Save and close now** opens, confirm with **[Yes]**.

The DaVinci Developer project is closed and the update of the DaVinci Configurator Pro project continues. Once the update finishes click **[Ok]**.

The project is now migrated to the new SIP.

13 Project Migration

13.1 Migration Steps from Release 15 SIP to Release 16

Open your existing configuration with the new SIP environment, i.e. Configurator 5.13.xx and execute **SolveAll**. After that, solve the remaining validation messages according to your knowledge. The following validation messages require further explanation:

1) **AR-ECUC02008 EthTSynGeneral contains a bigger number of parameter EthTSynGmCapable than upperMultiplicity specifies.**

This is a known issue appearing during SIP migration, which is easy to solve. Delete one of the two parameters.

2) **AR-ECUC02039 The target of the reference value StbMEthGlobalTimeDomainRef(value=<SomeEthTSynGlobalTimeDomainContainer>) must be a container of type DefinitionRef: /AUTOSAR/EcucDefs/**EthTSyn[1]**/EthTSynGlobalTimeDomain.**

This issue occurs, because of the introduced support of AR-4.2.x, which brings along the TimeSync modules. The manually configured reference of the previous SIP needs to be re-configured, i.e. select the appropriate container again from the list of available EthTSyn domains.

[1](Time Sync Over Ethernet) This module realizes the Ethernet specific time synchronization protocol and references to IEEE Standard 802.1AS ((PTP). An access to the synchronized time base by the SWCs requires the Synchronized Time-Base Manager ((StbM).

IV Appendix

1 Frequently Asked Questions

You have a certain question? You just want to know how to do e.g. a certain setting without reading the whole document again? Then go on reading the following list and use the links to get at the place in the document where your question will be answered. This chapter will be extended continuously.

1.1 Problems with using two different DaVinci Configurator Versions on the same PC

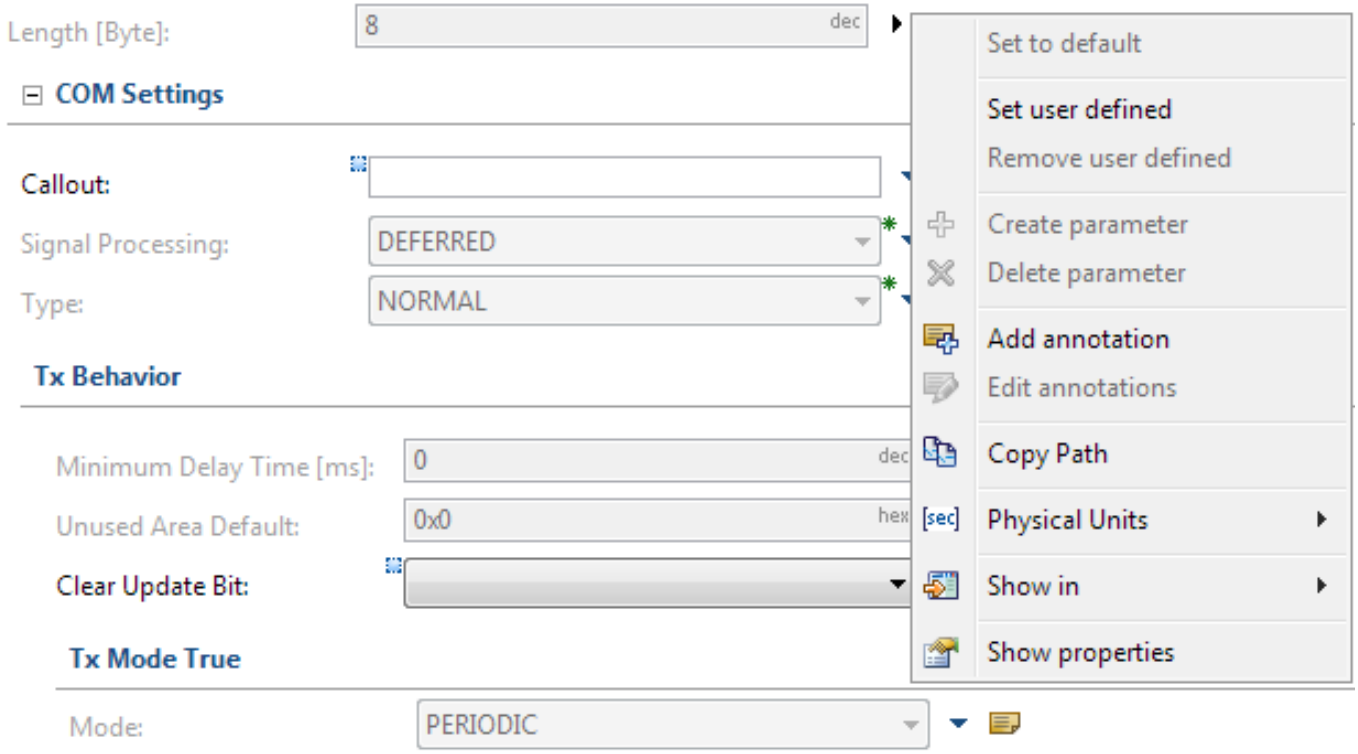
You get problems with using DaVinci Configurator versions 5.10.xx and DaVinci Configurator version 5.11.xx on the same PC?

Start your project with command line option **-clean** for solving Eclipse caching problems when DaVinci Configurator versions 5.10.xx and older are used on the same PC with DaVinci Configurator version 5.11.xx and younger.

1.2 Annotations for any parameter

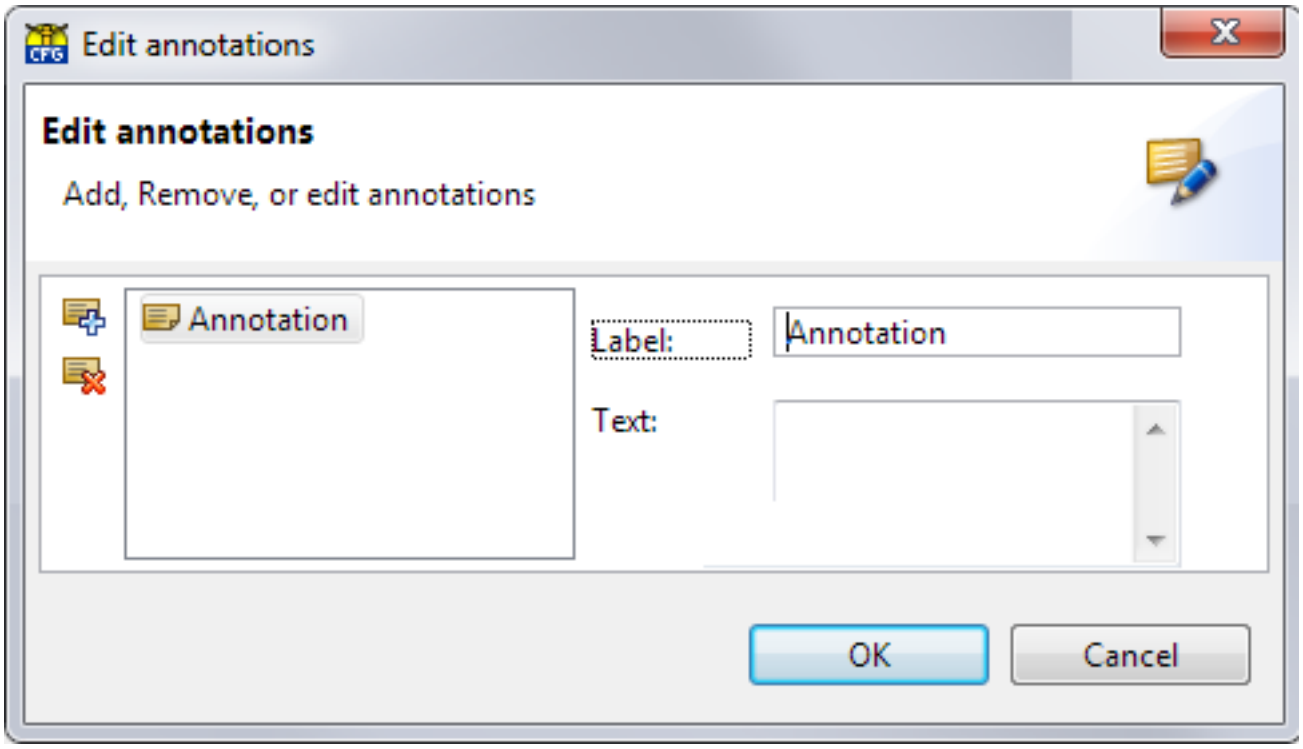
How to append an annotation to a parameter

For almost any parameter you can add individual annotations regardless whether you are in the Basic Editor or in the Configuration Editor. Open the context menu of a parameter by clicking the little right-pointing arrow and click **Add annotation**.

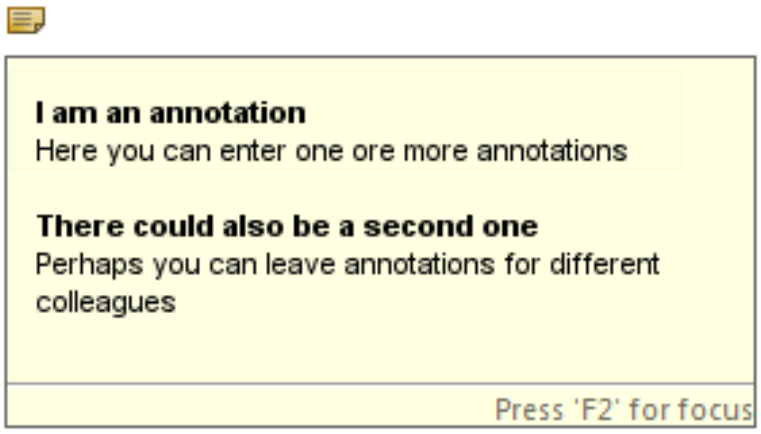


The annotation editor will open. You can enter one or more annotations each with a headline and floating text.

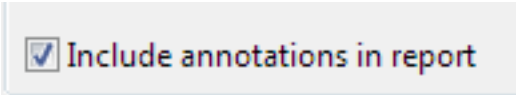
Via **Edit annotations** you can open and edit an already written annotation.



To read you annotations you can either open it via the editor or just by touching the little letter icon with the mouse, its content will be displayed like shown below as an example.



Via a report (**Project|Report...**) it is possible to see all your annotations. Make sure to activate the checkbox **Include annotations in report**



1.3 Find Reference Container

1.3.1 How to find references using the [Find] dialog

Open the **Find** dialog, type in **value==""** and give a container name between the quotes.

2 Release Notes

This section gives an overview of the changes which have been made on the MICROSAR Basic Software Modules (BSW) within the last releases.

Listed changes are new features or the extension of existing functionality. Please note that this document does not provide a list of fixed issues. Open issues are documented in the issue report document IssueReport_<CBD-Number>.pdf, which is part of the deliveries documentation.

This section refers to Vector internal tracking numbers (ESCAN, DAVID tickets and Feature List numbers) which uniquely identify the changes.

Please refer also to the Technical References for more information on the changes, modifications or extensions.

Changes to DaVinci Configurator Pro and DaVinci Developer are documented in Release Notes provided along with the tools.

2.1 Release 16

Breaking Changes

Modification of existing functionality or APIs that can have an impact on existing applications (such as SWCs, CDDs …).

Change ID	Affected Modules	Description
FEAT-1915	WDGM	A new feature for checkpoint generation has been added in this release of the WDGM. With the switch "WdgMGenerateCPIdAsPortDefinedArgument" it can be influenced if the checkpoint ID shall be generated as a port-defined argument or a single port per supervised entity is generated. Please note: To be compatible to older versions of the WDGM you need to enable this option. However, an AUTOSAR compliant behavior is achieved by disabling this functionality. By default this feature is disabled.
FEAT-1964	DEM	Removed post-build selectable from EnableConditionGroup and StorageConditionGroup. Variable groups of those types need to get duplicated and referenced by the events of a variant.

Extension

New APIs have been introduced or functionality has been added. There is no influence on existing implementations.

Change ID	Affected Modules	Description
FEAT-1327	J1939DCM	The J1939DCM DM1 transmission is now ISOBUS compliant (cyclically only if active instead of cyclic)
FEAT-1413	SOAD	Service oriented communication for Linux (BSD socket API enhancements for SOME/IP and Service Discovery).
FEAT-1498	DaVinci Configurator Pro 5 TCPIP	It is now possible to mark TcpIp[1] (IPv6) address prefixes as on-link during configuration time.
FEAT-1516	TCPIP	It is now possible to assign a static IPv4 address in addition to an address assigned by DHCP (priority based). Limitations: > one assignment method of each type per address > one IPv4 address per logical controller (VLAN, physical controller) **Beta Implementation. This feature shall not be used for production.**
FEAT-1574	CAL CPL CRY SECOC	New SECOC functionality: > Generic Freshness value interface > Provision of SecOC[2]_VerifyStatusOverride API > SecOCAuthInfoTxLength and SecOCFreshnessValueTxLength can have an arbitrary size but must result in a full byte length
FEAT-1641	SD	The following AUTOSAR 4.3 features are now supported: > [Issue 68746] Impact of configuration options on service matching algorithm is unspecified > [Issue 68823] Switching of EventHandler

[1](TCP/IP Stack) This module contains all protocols for UDP and TCP-based communication. It supports the versions IPv4 and IPv6 as well as parallel operation of IPv4 and IPv6 in one ECU. It contains the following protocols: - IPv4, ICMPv4 and ARP - IPv6, ICMPv6 and NDP - UDP, TCP, DHCPv4 (client) and DHCPv6 (client)
[2](Secure Onboard Communication) The SecOC is used to send or receive authenticated messages. Unauthorized, repeated or manipulated messages are detected. The SecOC is part of the AUTOSAR security solution.

Change ID	Affected Modules	Description
		from Unicast to Multicast
FEAT-1678	FLS (VTT) VTT Control Layer	VTT memory stack now supports the AURIX Fee[1] in case of dual target projects.
FEAT-1723	DEM	Support for OBD major monitors. Please note: This feature need to be licensed separately. OBD major monitors require a project specific analysis of the requirements to the basic software. **Beta Implementation. This feature shall not be used for production.**
FEAT-1724	DCM DEM	Support for DTR (Diag Test Results) for OBD projects. **Beta Implementation. This feature shall not be used for production.**
FEAT-1726	DaVinci Configurator Pro 5 DaVinci Developer RTE	RTE Nv ports can now be connected to S/R ports. **Beta Implementation. This feature shall not be used for production.**
FEAT-1738	ETHIF	Allow concatenated receive buffers that can be smaller than max frame size.
FEAT-1770	DEM	Up to 255 Enable- and Storage-Conditions are now supported.
FEAT-1791	RTE SOMEIPXF	Support autonomous error responses from transformers based on AUTOSAR 4.2.2 for SOMEIPXF.
FEAT-1807	CFG5 (DCM ARXML Converter) CFG5 (DEM ARXML Converter) DCM Diagnostic Data Model	DCM diagnostic data now can be variant in configurations that use the MICROSAR Identity Manager. **Beta Implementation. This feature shall not be used for production.**
FEAT-1818	COM	Support the configuration switch minimum delay time for cyclic transmission (Parameter "ComEn-

[1](Flash EEPROM Emulation) The Fee module offers a hardware-independent interface for accessing flash data and uses a flash driver ((Fls) for this. In addition to reading, writing and clearing data, the Fee module also distributes write accesses to different areas of flash memory, so that all flash cells are uniformly stressed, which increases their life-time.

Change ID	Affected Modules	Description
		ableMDTForCyclicTransmission").
FEAT-1820	Base EcuC Converter DOIP	Improved Vehicle Announcement Handling for DoIP[1] based on AUTOSAR [Issue 67021].
FEAT-1846	RTM	> RTM measurements can now be controlled via CAPL without using the test feature set UI. This allows integrating RTM into complex test environments. > Net runtime measurement support on multi-core ECUs (Beta) > Response time measurement on multi core ECUs (Beta)
FEAT-1864	DRM	Release of DRM (Diagnostic Request Manager) (CDDDRM)
FEAT-1889	CAN (VTT) LIN (VTT) VTT Control Layer VTT TechnicalReference	VTT now supports the MICROSAR Identity Manager and PB-Loadable for CAN and LIN. The PB-Loadable update process is not supported in VTT. Note: FlexRay is also supported (since Release 15). Note: This feature requires an IDM / PB-Loadable license.
FEAT-1896	IPDUM	The nPduToFrame mapping as used to pack several COM I-PDUs into one CAN-FD frame has been extended: > Last is best semantics for Tx PDUs > Runtime optimization: Rx pathway only handles PDUs that are actually required by the ECU **Beta Implementation. This feature shall not be used for production.**
FEAT-1899	DaVinci Configurator Pro 5 DaVinci Developer DCM DIAGXF	Introduction of DIAGXF module to support S/R interaction with DCM. This feature mandates a MICROSAR RTE. **Beta Implementation. This feature shall**

[1](Diagnostics over IP) Since AR 4.1.1 the DoIP module contains diagnostic functionality according to ISO 13400-2 like vehicle discovery. Up to and including AR 4.0.3, this functionality is part of the Socket Adaptor ((SoAd).

Change ID	Affected Modules	Description
	RTE	**not be used for production.**
FEAT-1908	All-MCALs BSWM CAN FR LIN WDG	Access to interrupt controller register has been updated to better support safety use-cases. Per driver it is now possible to configure two options via the parameter UseOsInterruptControl > OS is in charge of the interrupt control register. This new functionality mandates the latest MICROSAR OS. The module no longer initialized the interrupt control register by itself but informs the OS. Register values are derived through OS APIs. This setting is recommended for SafeBSW projects. > Same behavior as today: Depending on the driver implementation (see its technical reference) a driver may perform the initialization of the interrupt control register and accesses the relevant registers directly.
FEAT-1912	ETH Switch Driver (Bcm8953x)	The switch driver has been extended to support the AUTOSAR 4.2.1 frame mirroring and forwarding extensions. > CFP Rules for VLAN-filtering (mirroring) > Reserved MAC multicast packets shall be forwarded by the switch > Mirroring VLAN double-tagging > Enabling/disabling mirroring during runtime > Mirroring filters Please note that these features may not be available for each switch hardware.
FEAT-1922	DCM	Support for OBD2 Mode 0x09 with variable length as required by primary OBD nodes with multiple secondary nodes.
FEAT-1938	CRY	Software CRY now supports the CMAC AES-128 algorithm.
FEAT-1987	TSYNC ETH	The IEEE 802.AS Pdelay message configuration is more flexible and allows the following options: > Dynamic Pdelay protocol for time master and slave

Change ID	Affected Modules	Description
		> Optimized Pdelay configuration by time slave Pdelay_Req. Time master answers with Pdelay_Res and Pdelay_res_Follow_Up. > Optionally a static Pdelay configuration can cause Pdelay requests to be ignored.
FEAT-451	DaVinci Configurator Pro 5 DaVinci Developer RTE	The RTE implements automatic scaling (offset and factor) between a FixedPoint and Float. Use-Case: network representation is FixedPoint (with factor and offset) while the internal processing is Float based.

Information

The internal behavior of the BSW has been improved without any change in the API. Only for information purpose.

Change ID	Affected Modules	Description
FEAT-1506	ETH Switch Driver (Bcm8953x)	The Ethernet switch configuration has been improved. Thus only used ports have to be configured and pre-configured ports can be mapped to any used port. **Beta Implementation. This feature shall not be used for production.**
FEAT-1513	CAN CANIF XCP On CAN	Release of CAN-FD functionality.
FEAT-1640	COM	Runtime improvement in the COM Rx signal notification handling: > COM Rx signal notifications are now always called with the interrupt not being locked by the COM module > The Rx signal notifications are called after a configured amount of notifications has to be called. > An ISR lock threshold is configurable per timing domain.
FEAT-1701	FEE	The API ForceSectorSwitch now always causes the sector to be switched. This shall result in a more robust FEE in case of errors.

Change ID	Affected Modules	Description
FEAT-1779	SECOC	Release of SECOC module.
FEAT-1839	FEE (Small Sector Flash)	An alternative FEE implementation is now available that is specifically designed to support devices with very small flash sectors. On such devices (currently RH850) this implementation is more robust and has less overhead.
FEAT-1886	IPDUM	Feature Release: 16bit IPDUM selector field values are now supported for production projects
FEAT-1888	NVM	The problem that RTE invokes the DET during NvM ReadAll (ESCAN00087644) is now solved.
FEAT-1910	SOAD	Release of BSD-Socket API for series production
FEAT-1948	OSEKNM	Release of OSEK-NM implementation.

2.1.1 Required AUTOSAR Tools

This MICROSAR release requires the usage of the following tools:

Tool	Version
DaVinci Developer	**3.13.0** Please use the latest service pack available from the Vector download center
DaVinci Configurator Pro	The latest service pack version is delivered as part of the SIP. Upcoming service packs can be retrieved from the Vector download center

2.2 Release 15

Breaking Changes

Modification of existing functionality or APIs that can have an impact on existing applications (such as SWCs, CDDs …).

Change ID	Affected Modules	Description
FEAT-1631	Asr4NmStMgr_Renault CANNM COM Communication Interface Complex Driver FRIF FRNM	The semantics of the TriggerTransmit APIs have been changed and are now in line with AUTOSAR 4.2.2: > The PduInfo.SduLength parameter is now a in/out parameter > The lower layer provides the maximum available buffer to the higher layer module > The higher layer module copies the data (as in the past) and informs the lower layer module on the actual size of the data by updating

Change ID	Affected Modules	Description
	IPDUM LDCOM OSEKNM PDUR RTE SECOC SOAD TSYNC FR UDPNM	the SduLegth parameter Impact to existing projects: If the application implements TriggerTransmit APIs the handling of the SduLength parameter must be adapted as described above.
FEAT-1644	Communication Interface Complex Driver IPDUM J1939DCM J1939NM J1939RM	Support of Request2 support for complex drivers and RTE. IPDUM now supports the meta data handling as required by J1939 services. **Impact on existing projects:** The API of J1939Rm[1] have been changed and need to be adapted if being used. The following APIs have been an updated signature: > J1939Rm_SendRequest > J1939Rm_SendAck

Extension

New APIs have been introduced or functionality has been added. There is no influence on existing implementations.

Change ID	Affected Modules	Description
FEAT-1347	SOAD TCPIP TLS[2]	TLS is now a plug-in of the TCPIP module > TLS handling / usage is largely transparent for upper layers > TLS can be configured per socket separately
FEAT-1485	CANIF	Support of N:1 and 1:N multi-frame transport protocol routing paths. During runtime all of the

[1](SAE J1939 Request Manager) The J1939Rm module implements requesting of data via Request Handling that is defined in the SAE J1939 protocol.
[2](Transport Layer Security) This module contains a Transport Layer Security client. The TCP-based communication is encrypted with TLS. The encryption algorithm used can be selected.

Change ID	Affected Modules	Description
	CANTP Communication Interface Complex Driver FRIF FRTP IPDUM J1939TP LINIF LINTP PDUR SOAD	possible N sources can be active. This feature is useful for gateways that are e.g. implemented in varying EE architectures.
FEAT-1503	IPDUM	Added several new features to the IPDUM (Container PDU Handling): > FlexRay now supported > Big endian container headers > Tx confirmation and timeout handling > Runtime optimization (deferred event caching)j **Beta Implementation. This feature shall not be used for production.**
FEAT-1605	DEM J1939DCM	Support for additional J1939 diagnostic messages: > DM5: Diagnostic Readiness 1 > DM27: All Pending DTCs > DM53: Active Service Only DTCs > DM54: Previously Active Service Only DTCs > DM55: Clear All Service Only DTCs **Beta Implementation. This feature shall not be used for production.**
FEAT-1637	OS	The new MICROSAR OS generation now supports the OS Service Interface towards the application. Note: This feature is only available for projects using

Change ID	Affected Modules	Description
		the new (Gen7) OS.
FEAT-1657	Base EcuC Converter CANIF COM Communication Interface Complex Driver DaVinci Configurator Pro 5 FRIF IPDUM PDUR SOAD	If the system description maps a single PDU to multiple frames this results not in the same mapping within the ECUC configuration i.e. there will now be a single PDU only. Such kind of mapping is supported for CANIF, FRIF or SOAD PDUs. Up to now, this resulted in multiple higher layer PDUs. > reduction of configuration effort > RAM and runtime reduction
FEAT-1689	TSYNC ETH	Support Time Master & Time Slave functionality in parallel (required for Time Gateway). Note: > Only a single slave port is supported **Beta Implementation. This feature shall not be used for production.**
FEAT-1691	TSYNC ETH	Improved time synchronization by EthTSyn[1]: > RateCorrection as OffsetCorrection > FrequencyRateCorrection below and above SynchronizationThreshold **Beta Implementation. This feature shall not be used for production.**
FEAT-1694	ETH Switch Driver (Bcm8950x) STBM TSYNC CAN TSYNC ETH TSYNC FR	Added synchronized time features > Time gateway is now supported > EthTSyn supports boundary clock on each time master port **Beta Implementation. This feature shall not be used for production.**

[1](Time Sync Over Ethernet) This module realizes the Ethernet specific time synchronization protocol and references to IEEE Standard 802.1AS ((PTP). An access to the synchronized time base by the SWCs requires the Synchronized Time-Base Manager ((StbM).

Change ID	Affected Modules	Description
FEAT-1698	DOIP	DoIP now supports UUDT (Unacknowledged Unsegmented Data Transfer, ISO13400) as defined by AUTOSAR 4.2.2
FEAT-1731	LINIF	J2602 frame tolerance support allowing to specify. For LIN rx frames a specific tolerance as one common tolerance for the LIN tx frames. Frame tolerance is considered in the definition and validation of the schedule table slot delays.
FEAT-1741	TLS	TLS extensions for in-vehicle use case > Support of Certificate Revocation List > Support of TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (Secp256r1) > Support of server-certificate validation and client-certificate presentation (TLS Client)

Information

The internal behavior of the BSW has been improved without any change in the API. Only for information purpose.

Change ID	Affected Modules	Description
FEAT-1201	TSYNC ETH	QM release of EthTSyn module
FEAT-1427	DLT	QM release of DLT (AUTOSAR) module.
FEAT-1449	DOIP SD UDPNM	QM release of the following Ethernet related modules: > UdpNm[1] > DoIP > Sd[2]
FEAT-1619	DaVinci Configurator Pro 5 RTE VTT Control Layer	The configuration workflow of RTE configuration for VTT hooks has been changed: > The VTT tool no longer adapts the generated code of the RTE. Instead the VTT tool requirements are communicated to the RTE using dedicated VTT VFB Trace Function con-

[1](UDP Network Management) You use network management over UDP to implement synchronous transition to sleep mode for Ethernet ECUs.
[2](Service Discovery) Service Discovery was first specified in AR 4.1.1. An ECU communicates the availability of its services to communication partners via the publish-subscribe protocol implemented in this module. In addition, ECUs can register to receive automatic notifications, e.g. on a signal update.

Change ID	Affected Modules	Description
		figuration elements within the RTE configuration prior to RTE generation.
FEAT-1650	SECOC	Improved SecOC usability by additional validation rules in DaVinci Configurator Pro: > Truncated Freshness Value + Tx Authenticator + Authentic Pdu Length must be equal Secured Pdu Length > Auth Info Tx Length and Freshnessvalue Tx Length must be a multiple of 8 > A Freshness Value ID should not be used twice > Freshness Timestamp Feature is not supported > If a Verification Status Callout is configured at least one SecOCVerificationStatusPropagationMode must be != NONE > Correct types for SecOCMaxAlignScalarType parameter **Beta Implementation. This feature shall not be used for production.**
FEAT-1687	Fr[1] (VTT) VTT Control Layer VTT-Tool	VTTFr now supports PB-Loadable and PB-Selectable (Identity Manager).
FEAT-421	RTE	QM release of RTE inter ECU client/server communication.
FEAT-441	RTE	QM release of RTE S/R serialization.

2.2.1 Required AUTOSAR Tools

This MICROSAR release requires the usage of the following tools:

Tool	Version
DaVinci Developer	**3.12.x** Please use the latest service pack available from the Vector download center
DaVinci Configurator Pro	The latest service pack version is delivered as part of the SIP. Upcoming service packs can be retrieved from the Vector download center

[1](FlexRay Driver) The FlexRay Driver abstracts access to the FlexRay hardware for sending and receiving data and for switching between controller states.

2.3 Release 14

Breaking Changes

Modification of existing functionality or APIs that can have an impact on existing applications (such as SWCs, CDDs …).

Change ID	Affected Modules	Description
FEAT-10	Base EcuC Converter DOIP PDUR SOAD TCPIP	The DoIP implementation is now based on AUTOSAR 4.2.1. The following callback signatures have been changed and now comply with AUTOSAR 4.2.1: > <User_GetVin> > <User_GetDiagPowerMode> > <User_GetGid> the following user callbacks have been added: > <User_TriggerGidSync> > <User_ActivationLineState> > <User_RoutingActivationAuthentication> > <User_RoutingActivationConfirmation> For more details please refer to the DoIP Technical Reference document. NOTE: The upstream mapping is based on AUTOSAR 4.2.2 as it is not defined in earlier versions. Impact to existing projects: If DoIP user-callbacks are used, the implementation may have to be adapted according to the new interface.
FEAT-1713	FIM	The following FIM features have been discontinued as these are apparently not being used any more: > cyclic event evaluation: Supported value: FiMGeneral[FiMEventUpdateTriggeredByDem] = TRUE: MICROSAR DEM always uses TRUE as mandated value. > calibration support: Supported value: FiMGeneral[FiMDataFixed] = FALSE: Con-

Change ID	Affected Modules	Description
		figuration update can be realized using post-build loadable instead. **Impact on existing projects:** If the discontinued features have been used: Please check with your Vector contact in order to find a replacement for the discontinued functionality.
FEAT-1776	ETM	The Ethernet test module ETI has been renamed to ETM. Impact to existing projects: > API names have been changed. Access to APIs need to be adapted. > BSWMD names have been changed. Configuration need to be reworked.

Extension

New APIs have been introduced or functionality has been added. There is no influence on existing implementations.

Change ID	Affected Modules	Description
FEAT-1295	OSEKNM	OSEK-NM now supports post-build selectable (MICROSAR Identity Manager) and post-build loadable. Note: These features require dedicated licensing. **Beta Implementation. This feature shall not be used for production.**
FEAT-1357	XCP	For safety critical ECUs (ISO26262) it is now possible to deactivate the XCP module in a safe way during normal operation. Using a dedicated API the application can enable XCP handling. Once XCP is enabled the XCP module is no longer a safe module from implementation perspective. Note: This feature requires dedicated licensing of MICROSAR SafeBSW for XCP.

Change ID	Affected Modules	Description
FEAT-1370	Communication Interface Complex Driver	The CDD[1] interface for communication PDUs now supports PDUs to be variant. The implementation of the complex driver shall use the complex driver APIs in a variant specific way.
FEAT-1429	CAN CANIF	The CAN stack High-End option allows a more efficient CAN stack realization by the following new features: > Support of several CAN driver basic Tx objects > The queue type can be configured for each Tx BasicCAN object individually (e.g. including the depth of each queue) **Beta Implementation. This feature shall not be used for production.**
FEAT-1436	Base EcuC Converter COM Com[2] Validation Lib VASE Scripting	The COM module has been extended with several features that increase performance: > Support of multiple COM main functions with different timing domains. This can reduce the overall runtime of COM MainFunctions > Optimized handling of deferred events (queue instead of a linear search) > Optional Signals and Signal Groups. It is now possible to configure if a Signal is accessed or not > Description Based Routing allows a more efficient implementation of a signal gateway Note: Description based routing requires dedicated licensing **Beta Implementation. This feature shall not be used for production.**
FEAT-1481	TCPIP	IPv4 fragmentation support. Required to send and receive UDP packets with more than 1472

[1](Complex Drivers) The Complex Drivers are software modules that are not standardized by AUTOSAR. They have access to other BSW modules, the RTE and direct hardware access. For example, these modules are not standardized communication drivers for SPI or legacy software.
[2](Communication) The Com module provides a signal-based data interface for the RTE. It places signals in messages and sends them according to the defined send type. The module contains various notification mechanisms for receiving signals. It also manages initial values, update bits and timeouts on the signal level. In multi-channel ECUs, the integrated signal gateway offers the ability to route signals between the communication buses.

Change ID	Affected Modules	Description
		bytes. **Beta Implementation. This feature shall not be used for production.**
FEAT-1485	CANIF CANTP Communication Interface Complex Driver FRIF FRTP IPDUM J1939TP LINIF LINTP PDUR SOAD	Support of N:1 and 1:N multi-frame transport protocol routing paths from configuration point of view. At runtime only one of the N possible paths may be active at the same time. This feature is useful for gateways that are e.g. implemented in varying EE architectures.
FEAT-1486	ARXML(MC data) to a2l Converter DaVinci Configurator Pro 5 VTT Control Layer VTT MCAL Modules	vVIRTUALtarget (VTT) modules have been improved: > VTT MCAL modules do no longer mandate a configuration of a include file list. > VTTCNTRL provides an API that enabled complex drivers and IOHWAB to create and access CANoe system variables directly. This allows the stimulation and monitoring of I/Os[1] that are e.g. connected by SPI or I2C or hardware that cannot be accessed by AUTOSAR drivers. > VTT MCAL now supports more channels (e.g. ADC, DIO...). > VTTWDG can now also operate without GPT. > VTTADC has now a value range from 1:63 as

[1](Operating System) This module is the operating system of an AUTOSAR ECU. It is actually an extended OSEK operating system. Extensions are divided into so-called Scalability Classes (SC1-SC4). They cover the following functionalities: - SC1: schedule tables - SC2: timing protection + schedule tables - SC3: memory protection + schedule tables - SC4: memory protection + timing protection + schedule tables In safety-related ECUs, the OS is one of the safety-related modules. The operating system also supports multi-core microcontrollers.

Change ID	Affected Modules	Description
		defined by AUTOSAR. > VTTMCU allows the implementation of a ECU reset as it now supports also "normal" mode. > XCP module is now able to consider the address offset of the dynamically loaded DLL automatically. This allows online calibration of RAM variables of the virtual ECU without manual address translation. > RAMTST supports VTT with respect to APIs. No actual test algorithms are applied in case the target is VTT.
FEAT-1491	NVM	The NvM block size handling is now more flexible to simplify ECU development. For NVRAM Blocks it is now possible to specify a maximum block size. The maximum size defines an upper boundary for user data. At determines the actual sizes using the configured permanent RAM Blocks' types. Instead of calculating and maintaining exact block sizes, users may estimate them, and keep them even across changes during development. Once the ECU configuration is completed the block size can be reduced to the exact size of the type to be stored in order to limit overhead in NV memory.
FEAT-1505	DaVinci Configurator Pro 5 SD SOAD	Post-build loadable for Socket Adaptor and Service Discovery. Note: This feature requires dedicated licensing. **Beta Implementation. This feature shall not be used for production.**
FEAT-1519	DaVinci Configurator Pro 5 DaVinci Developer RTE	The RTE now supports Rte[1]_ActivatingEvent including the definition of an activation reason.

[1](Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

Change ID	Affected Modules	Description
FEAT-1529	ETH (MPC55xx) ETH Switch Driver (Bcm8950x) ETHIF TSYNC ETH	Upper layers, such as EthTSyn, need a port-view for transmission and reception of Ethernet messages. Ethernet Switch-Driver and EthIf[1] have been extended by additional API's to provide that functionality. **Beta Implementation. This feature shall not be used for production.**
FEAT-1531	STBM	Support of High Resolution Time Base Reference Clock based on GPT **Beta Implementation. This feature shall not be used for production.**
FEAT-1535	DaVinci Configurator Pro 5 RTE	The RTE allows the distribution of service layer BSW modules in several partitions as it is required by the multi-core master-satellite concept. Note: This feature need to be supported by each BSW module individually.
FEAT-1540	All-MCALs DaVinci Configurator Pro 5 ECUM	The initialization requirements of third party MCALs can now be defined more precisely by Vector. This allows the creation of more complete initialization sequences. Note: This feature need to be released for each MCAL separately and may therefore not be available for all MCALs.
FEAT-1553	Base EcuC Converter J1939RM	J1939Rm now supports basic handling of Request2 messages: > RQST2 reception > Extended identifier bytes in NACK after RQST2 > Evaluation of extended identifier bytes of received RQST2 to trigger multiplexed PDUs in COM **Beta Implementation. This feature shall not be used for production.**
FEAT-1593	DCM	Diagnostic Variants can now be supported by DCM using dedicated APIs that can be used to

[1](Ethernet Interface) The Ethernet Interface enables bus-independent control of the Ethernet driver ((Eth) and Ethernet transceiver driver ((EthTrcv). Since AR 4.1, this module is also responsible for VLAN handling.

Change ID	Affected Modules	Description
		control variance. API documentation is provided in the DCM technical reference (chapter 9.29 "How to Handle Multiple Diagnostic Service Variants").
FEAT-1618	DaVinci Configurator Pro 5 DaVinci Developer RTE	Mapping of Record Data Elements to Receiver Primitive Data Elements. This also allows to access individual signals from a signal group. Has to be configured on Connector Prototype based on Port Interface Mapping.
FEAT-1648	CANSM	Implementation of RfC 52550 - Reinitialization of CAN CC after NM-Timeout if PN is used. A CanSM[1]_TxTimeoutException is handled with the following steps: > set CAN Controller to stopped by invoking CanIf[2]_SetControllerMode(CtrlID,CANIF_CS_STOPPED) > set CAN Controller to the requested state by invoking CanIf_SetControllerMode (CtrlID,<requested mode>)
FEAT-1695	ETH Switch Driver (Bcm8950x)	Mirroring of Ethernet traffic on a configurable switch port for monitoring. Limited to BCM895xx/BCM892xx Ethernet switches. **Beta Implementation. This feature shall not be used for production.**
FEAT-1711	ETM	Implementation of the Ethernet Testability Module (ETM) for IETF RFC conformance testing according to AUTOSAR concept CONC_618_EthernetTestability. **Beta Implementation. This feature shall not be used for production.**
FEAT-1737	BSWM	Pending COMM requests are now considered in state machine of the "ECU State Handling". A Pending request causes the state machine to stay in RUN or change back from POST_RUN to RUN.

[1](CAN State Manager) The CAN State Manager is responsible for the bus-specific error handling.
[2](CAN Interface) The CAN Interface offers abstracted (PDU-based) access to the CAN Driver. It controls the CAN Driver ((Can) as well as the transceiver driver ((CanTrcv).

Change ID	Affected Modules	Description
FEAT-921	RTM	Runtime measurement in multi-core environments supported. Measurement results can be exported in a csv formant compatible with TimingArchitects tooling for further evaluation of multi-core runnable scheduling. **Beta Implementation. This feature shall not be used for production.**

Information

The internal behavior of the BSW has been improved without any change in the API. Only for information purpose.

Change ID	Affected Modules	Description
FEAT-1431	CANTP	Optimization of MainFunction runtime. This feature becomes relevant in gateway ECUs with many TP connections.
FEAT-1600	J1939NM J1939RM	Release of > J1939Nm[1] > J1939Rm
FEAT-1660	DCM	If Dcm[2]_ReadMemory returns DCM_READ_FAILED, the DCM module now triggers a negative response with NRC 0x10 (GeneralReject). Implementation based on [SWS_Dcm_00644].
FEAT-236	RTE	The union- and bit field data type handling have been released (initial implementation through FEAT-349).
FEAT-569	RTE	The implementation based on the concept "Combined require and provide semantic for ports (AUTOSAR Concept 584) has been released (initial implementation through FEAT-1227).
FEAT-82	RTE	Text Table Mapping resp. Bit Field mapping has

[1](SAE J1939 Network Management) J1939 supports adding ECUs to networks on-the-fly. The J1939Nm module is responsible for negotiating a unique ECU address ("AddressClaim") and unlike other NM modules for handling the wake-up or going to sleep of the bus.
[2](Diagnostic Communication Manager) The Dcm module implements diagnostic communication according to ISO 14229-1:2006 (UDS). Some diagnostic requests are processed directly in the Dcm (management of diagnostic sessions, reading of error codes, EcuReset, etc.) and some are routed to the SWCs via port interfaces (reading, writing and controlling of data elements within a data identifier, execution of routines, etc.). Legal requirements of OBDII / SAE J1979 are also supported.

Change ID	Affected Modules	Description
		been released to allow a more efficient NV data handling (initial implementation through AR4-782).

2.3.1 Required AUTOSAR Tools

This MICROSAR release requires the usage of the following tools:

Tool	Version
DaVinci Developer	**3.11.x** Please use the latest service pack available from the Vector download center
DaVinci Configurator Pro	The latest service pack version is delivered as part of the SIP. Upcoming service packs can be retrieved from the Vector download center

3 What's New, What's Changed

What's new and what's changed?

This section explains the changes within this document from the previous version to the one mentioned in the headline.

Version 4.x.x

Author	Date
Klaus Emmert	2016-09-13

What's new?

- Hint to Acknowledgment in validation view 03_Validation.htm
- Service Mapping in DaVinci Configurator Pro 06_Mapping.htm
- BswM[1]: activation of peripheral interrupt devices via callout 04_ConfigureBSW.htm.
- New view for comfortable task mapping 06_Mapping.htm
- Migration steps from Release 15 to Release 16

What's changed?

- Add necessary information for MICROSAR 4 release 14.x.
- New migration hints for updating to or over Release 15 concerning PDUs and DEM 20_ProjectMigration.htm
- New symbols in the chapter 00_About This User Manual.htm
- Topics concerning older releases deleted
- Minor changed, screenshots, typos, etc.
- Workflow illustration now visible

[1](BSW Mode Manager) The BswM module contains vehicle mode management and application mode management. It processes mode requests from SWCs or other BSW modules, and it performs actions based on the arbitration, such as control of deadline monitoring, switching schedule tables, and handling of IPDU groups. In conjunction with the EcuM, the BswM module is responsible for starting up and shutting down the ECU. The BswM also coordinates the multi-core partitions.

4 Glossary

Adc
(Analog Digital Converter Driver) The ADC driver abstracts hardware access to the analog-digital converter. For every input, the conversion parameters are configured (e.g. resolution, trigger source and trigger conditions).

AVTP
(Audio/Video Transport Protocol) The Audio/Video Transport Protocol is specified in IEEE 1722/1722a. In AVB networks it is responsible for the transport of audio/video data, including the Presentation Time.

Bfx
(Bitfield functions for fixed point) Library for bit handling in fixed point arithmetic functions

BMCA
(Best Master Clock Algorithm) The Best Master Clock Algorithm is specified in IEEE 802.1AS and is used to detect the device with the most precise time unit in an AVB network. After finding the device with the most precise time unit, it is used as the time base for the entire system.

BswM
(BSW Mode Manager) The BswM module contains vehicle mode management and application mode management. It processes mode requests from SWCs or other BSW modules, and it performs actions based on the arbitration, such as control of deadline monitoring, switching schedule tables, and handling of IPDU groups. In conjunction with the EcuM, the BswM module is responsible for starting up and shutting down the ECU. The BswM also coordinates the multi-core partitions.

Cal (Cpl)
(Crypto Abstraction Library) The Cal library offers SWCs and other BSW modules access to basic cryptographic functions. The individual cryptographic functions are implemented in software via the Cpl module.

Can
(CAN Driver) The CAN Driver abstracts access to the CAN hardware for sending and receiving messages and for switching between controller states (sleep, stop, etc.)

CanIf
(CAN Interface) The CAN Interface offers abstracted (PDU-based) access to the CAN Driver. It controls the CAN Driver ((Can) as well as the transceiver driver ((CanTrcv).

CanNm
(CAN Network Management) Within a CAN network, CAN Network Management is responsible for coordinated transitions between the wake up and sleep state.

CanSM
(CAN State Manager) The CAN State Manager is responsible for the bus-specific error handling.

CanTp
(CAN Transport Layer) The CanTp module conforms to ISO standard 15765-2. As the transport protocol for CAN, it is responsible for segmenting the data in the Tx direction, collecting data in the Rx direction and monitoring the data stream.

CanTrcv
(CAN Transceiver Driver) This driver is responsible for controlling the operating states of an external CAN transceiver. It contains control of wake up and sleep functions.

CanTSyn
(Time Sync Over CAN) This module realizes the CAN specific time synchronization protocol. An access to the synchronized time base by the SWCs requires the Synchronized Time-Base Manager ((StbM).

CanXcp / XCPonCan
(CAN XCP-Module) The CanXcp module contains CAN-specific contents of the XCP module ((Xcp).

CDD
(Complex Drivers) The Complex Drivers are software modules that are not standardized by AUTOSAR. They have access to other BSW modules, the RTE and direct hardware access. For example, these modules are not standardized communication drivers for SPI or legacy software.

Com
(Communication) The Com module provides a signal-based data interface for the RTE. It places signals in messages and sends them according to the defined send type. The module contains various notification mechanisms for receiving signals. It also manages initial values, update bits and timeouts on the signal level. In multi-channel ECUs, the integrated signal gateway offers the ability to route signals between the communication buses.

ComM
(Communication Manager) The ComM module coordinates the communication channels and Partial Network Cluster with the communication requirements of the application.

ComXf
(COM Based Transformer) The COM based transformer optimizes for signal groups the interaction between RTE and the COM module, because you can avoid the shadow buffer in the COM module. It de-/serializes the data identical to the COM module.

CorTst
(Core Test Driver) The CorTst module contains configuration and control of the test capabilities included in the microcontroller core. In addition, it offers a framework for extending these test capabilities. Specifically for safety-related software, the CorTst module tests critical units such as the ALU and the registers.

Crc
(CRC Routines) The Cyclic Redundancy Check library calculates CRC checksums.

Csm (Cry)
(Crypto Service Manager) The Csm module offers SWCs access to basic cryptographic functions. The individual cryptographic functions are implemented by the Cry module in software or hardware (access via (She).

Dbg
(Debugging) The debugging module enables external access to internal information of the basic software. It is also possible to modify memory data.

Dcm
(Diagnostic Communication Manager) The Dcm module implements diagnostic communication according to ISO 14229-1:2006 (UDS). Some diagnostic requests are processed directly in the Dcm (management of diagnostic sessions, reading of error codes, EcuReset, etc.) and some are routed to the SWCs via port interfaces (reading, writing and controlling of data elements within a data identifier, execution of routines, etc.). Legal requirements of OBDII / SAE J1979 are also supported.

Dem
(Diagnostic Event Manager) The Dem module implements a fault memory. The standardized interface for "DiagnosticMonitors" enables uniform development of manufacturer-independent SWCs. The Dem module is responsible for administering the DiagnosticTroubleCode states, environmental data, and for storing the data in NVRAM. The legal requirements of OBDII / SAE J1979 are also supported.

Det
(Development Error Tracer) The Det module supports error debugging during software development. It provides an interface for error notification, which is called by the individual BSW modules in case of error.

Dio
(Digital Input Output Driver) The Digital Input Output Driver provides read and write services for the DIO channels (pins), DIO ports and DIO channel groups.

Dlt
(Diagnostic Log and Trace) The Dlt module provides generic “Logging and Tracing” functionality for SWCs and for the BSW modules (Rte, (Det and (Dem.

DNS
(Domain Name System) The DNS module contains a DNS resolver. It is responsible for resolving a domain, e.g. vector.com, into a valid IP address.

DoIP
(Diagnostics over IP) Since AR 4.1.1 the DoIP module contains diagnostic functionality according to ISO 13400-2 like vehicle discovery. Up to and including AR 4.0.3, this functionality is part of the Socket Adaptor ((SoAd).

DrvExt
(External Driver) Upon request, you can obtain the implementation of drivers for externally connected components. They are already available for plenty of external devices, e.g. for driv-

ing certain EEPROMs (EEPEXT), flash chips (FLSEXT), watchdogs (WDGEXT), analog-digital converters (ADCEXT) and communication controllers (CANEXT, LINEXT, ETHSWTEXT).

E2E
(End-to-End Communication Protection Library) Library for secure data exchange according to ISO 26262 for safety-related ECUs. It is responsible for calculating the checksum and providing the message counter.

E2EPW
(End-to-End Protection Wrapper) The E2E Protection Wrapper extends the RTE by adding verification of safety-related signals.

E2EXf
(E2E Transformer) With the E2E Transformer, you integrate the protection of safety-relevant signals according to ISO 26262 in the RTE interface. In contrast to the E2E protection wrapper, the standard interfaces of the RTE are used.

Ea
(EEPROM Abstraction) The Ea module offers a hardware-independent interface for accessing EEPROM data and uses an EEPROM driver ((Eep) for this. In addition to reading, writing and clearing data, the Ea module also distributes write accesses to different areas of the EEPROM, so that all EEPROM cells are uniformly stressed, which increases their life-time.

EcuM
(ECU State Manager) The ECU State Manager is responsible for starting up and shutting down the ECU as well as waking it up. It is available in two variants: flexible and fixed. The EcuM Fix manages a number of defined, fixed operating states. Using the EcuM Flex, the user defines all operating states flexibly in the (BswM module. This lets you implement special energy-saving states and various types of startup. In a multi-core system, the EcuM coordinates the various cores.

Eep
(EEPROM Driver) The EEPROM driver enables hardware-independent access to EEPROM memory. It provides services for reading, writing and comparing data.

Efx
(Extended Fixed Point Routines) Library with extended mathematical functions for fixed-point values

Eth
(Ethernet Driver) The Ethernet Driver abstracts access to the Ethernet hardware for sending and receiving data and for switching between controller states.

EthIf
(Ethernet Interface) The Ethernet Interface enables bus-independent control of the Ethernet driver ((Eth) and Ethernet transceiver driver ((EthTrcv). Since AR 4.1, this module is also responsible for VLAN handling.

EthSM
(Ethernet State Manager) The Ethernet State Manager provides the Communication Manager ((ComM) with an abstract interface for starting up or shutting down communications in Ethernet clusters. EthSM accesses the Ethernet hardware via (EthIf.

EthSwt
(Ethernet Switch Driver) The module EthSwt provides a uniform and hardware independent interface for controlling and configuring Ethernet switches. It also coordinates the MAC learning when using multiple identical ECUs like cameras for the surround view.

EthTrcv
(Ethernet Transceiver Driver) EthTrcv offers a uniform and hardware-independent interface for driving multiple transceivers of the same kind. Configuration of EthTrcv is transceiver-specific and considers the properties of the physical network that is used.

EthTSyn
(Time Sync Over Ethernet) This module realizes the Ethernet specific time synchronization protocol and references to IEEE Standard 802.1AS ((PTP). An access to the synchronized time base by the SWCs requires the Synchronized Time-Base Manager ((StbM).

EthXcp/ XCPonEth
(Ethernet XCP-Module) The EthXcp module contains Ethernet-specific contents of the XCP module ((Xcp).

EXI
(Efficient XML Interchange) The EXI module is used to interpret XML document and to convert them to a binary format. This makes it more efficient to process the files and to transmit them, in order to economize on communication bandwidth. It is used in the Vehicle2Grid environment.

Fee
(Flash EEPROM Emulation) The Fee module offers a hardware-independent interface for accessing flash data and uses a flash driver ((Fls) for this. In addition to reading, writing and clearing data, the Fee module also distributes write accesses to different areas of flash memory, so that all flash cells are uniformly stressed, which increases their life-time.

FiM
(Function Inhibition Manager) Based on the active errors managed by the (Dem module, the FiM offers the ability to prevent execution of functionalities in SWCs.

Fls
(Flash Driver) The flash driver enables hardware-independent and uniform access to flash memory. It provides services for reading, writing and comparing data, and for deleting blocks (sectors).

FlsTst
(Flash Test) The Flash Test module offers algorithms for testing nonvolatile memory such as data or program flash, SRAM and protected cache.

Fr
(FlexRay Driver) The FlexRay Driver abstracts access to the FlexRay hardware for sending and receiving data and for switching between controller states.

FrArTp
(FlexRay AUTOSAR Transport Layer) FrArTp is a FlexRay transport protocol. Based on ISO 15765-2 ((CanTp), it contains a frame compatibility with the CAN bus.

FrIf
(FlexRay Interface) The FlexRay Interface offers abstracted (PDU-based) access to the FlexRay hardware. In addition, it offers support for synchronization with the global FlexRay time.

FrNm
(FlexRay Network Management) This module is responsible for network management in FlexRay. It synchronizes the transition to the bus sleep state.

FrSM
(FlexRay State Manager) The FlexRay State Manager controls and monitors the wake up and startup of nodes in the FlexRay cluster.

FrTp
(FlexRay ISO Transport Layer) FrTp is a FlexRay transport protocol and is based on the ISO 10681-2 standard.

FrTrcv
(FlexRay Transceiver Driver) The driver for an external FlexRay transceiver is responsible for switching the transceiver on and off.

FrTSyn
(Time Sync Over FlexRay) This module realizes the FlexRay specific time synchronization protocol. An access to the synchronized time base by the SWCs requires the Synchronized Time-Base Manager ((StbM).

FrXcp/ XCPonFr
(FlexRay XCP-Module) The FrXcp module contains FlexRay-specific contents of the XCP module ((Xcp).

Gpt
(General Purpose Timer Driver) The General Purpose Timer Driver provides an interface for accessing internal timers of the microcontroller. It is used for controlling periodic and singular events, for example.

HTTP
(Hypertext Transfer Protocol) The Hypertext Transfer Protocol, for instance, routes browser requests to a server. The module contains a HTTP client. It is used in the Vehicle2Grid environment.

Icu
(Input Capture Unit Driver) The Icu driver provides services for edge detection, measurement of periodic signals, assignment of edge time stamps and control of wake up interrupts.

Ifl
(Interpolation Floating Point) Library with interpolation functions for floating point values

Ifx
(Interpolation Fixed Point) Library with interpolation functions for fixed point values

IicDrv
(I²C Driver) The I²C Driver provides services for communication with external I2C chips.

IoHwAb
(I/O Hardware Abstraction) The I/O Hardware Abstraction represents the connection between the RTE and the ECU's I/O channels. It encapsulates access to the I/O drivers such as (Adc, (Dio or (Pwm and thereby makes the I/O signals of the ECU available to the SWCs.

IpduM
(I-PDU Multiplexer) This module is responsible for multiple use of fixed PDUs with different data contents.

J1939Dcm
(SAE J1939 Diagnostic Communication Manager) The module implements Diagnostic Messages of the SAE J1939-73 protocol, e.g. for reading out the fault memory.

J1939Nm
(SAE J1939 Network Management) J1939 supports adding ECUs to networks on-the-fly. The J1939Nm module is responsible for negotiating a unique ECU address ("AddressClaim") and unlike other NM modules for handling the wake-up or going to sleep of the bus.

J1939Rm
(SAE J1939 Request Manager) The J1939Rm module implements requesting of data via Request Handling that is defined in the SAE J1939 protocol.

J1939Tp
(SAE J1939 Transport Layer) The J1939TP module contains the transport protocols BAM (Broadcast Announce Message) and CMDT (Connection Mode Data Transfer) of the SAE J1939 standard.

LdCom
(Large Data COM) The LdCom module is optimized for routing large signals. Thereby it avoids an unnecessary copying of data. The LdCom is typically used together with (SomeIpXf as a serialization transformer.

Lin
(LIN Driver) The LIN Driver provides services for initiating frame transmission (header, response, sleep-mode and wake up), and for receiving responses, checking the current state and validating wake up events.

LinIf
(LIN Interface) The LIN Interface offers abstracted (PDU-based) access to the LIN hardware. It also handles schedule table processing and contains the LIN transport protocol ((LinTp).

LinNm
(LIN Network Management) The LinNm module contains a hardware-independent protocol, which coordinates the transition between normal operation and the bus sleep mode of the LIN network.

LinSM
(LIN State Manager) The LIN State Manager switches between schedule tables and PDU groups in the (Com module and services the LIN interface with regard to sleep and wake up.

LinTp
(LIN Transport Layer) The LIN transport protocol is responsible for segmenting data in the Tx direction, collecting data in the Rx direction and monitoring the data stream. According to the AUTOSAR specification, LinTp is part of (LinIf.

LinTrcv
(LIN Transceiver Driver) The module for an external LIN transceiver is responsible for monitoring and driving the wake up and sleep functions.

LinXcp
(LIN XCP-Module) The LinXcp module contains LIN-specific contents of the XCP module ((Xcp).

Mcu
(Micro Controller Unit Driver) The MCU driver provides the services for: - A microcontroller reset triggered by software - Selecting microcontroller states (STOP, SLEEP, HALT, etc.) - Configuring wake up behavior - Managing the internal PLL clock unit - Initializing RAM areas with predefined values.

MemIf
(Memory Abstraction Interface) This module provides uniform access to the services of (Ea and (Fee. This lets you use multiple instances of these modules.

Mfl
(Mathematical Floating Point) Library with arithmetic functions for floating point values

Mfx
(Mathematical Fixed Point) Library with arithmetic functions for fixed point values

Nm
(Generic Network Management Interface) The Nm module offers a general and network-independent interface for accessing the bus-dependent network management modules ((CanNm, (LinNm, (UdpNm and (FrNm). In addition, the module handles synchronous, inter-network shutdown of the communication system in coordination with the other ECUs.

NvM
(Non Volatile RAM Manager) The NvM module manages, reads and writes data to a nonvolatile memory ((Ea or (Fee). At system start and at shutdown, it synchronizes the data in the RAM

areas of the application. The module provides services such as saving of redundant blocks for a higher level of data protection. Since AR 4.0.3, the (RTE also provides a simpler and more flexible interface to Nv data (NvDataInterfaces).

Ocu

(Output Compare Unit Driver) The OCU driver standardizes initialization and access to the Output Compare Unit.

Os

(Operating System) This module is the operating system of an AUTOSAR ECU. It is actually an extended OSEK operating system. Extensions are divided into so-called Scalability Classes (SC1-SC4). They cover the following functionalities: - SC1: schedule tables - SC2: timing protection + schedule tables - SC3: memory protection + schedule tables - SC4: memory protection + timing protection + schedule tables In safety-related ECUs, the OS is one of the safety-related modules. The operating system also supports multi-core microcontrollers.

PduR

(PDU Router) The PDU Router handles distribution of the communication packets (PDUs) between the bus systems and the various BSW modules. In addition it offers gateway mechanisms for routing PDUs (FIFO) and TP-PDUs (on-the-fly) between the bus systems.

Port

(Port Driver) This module is responsible for initialization of the entire port structure of the microcontroller.

PTP

(Precision Time Protocol) The PTP module is used to distribute a precise system time to all devices participating in an AVB network. It is implemented according to IEEE 802.1AS. The implementation consists of microprocessor-dependent and independent parts.

Pwm

(Pulse Width Modulation Driver) The Pwm driver provides services for initialization and control of the PWM (pulse-width modulation) channels of the microcontroller.

RamTst

(Ram Test) This module tests internal microcontroller RAM cells. A complete test is triggered during startup and shutdown of the ECU or by a diagnostic command. During normal operation, a periodic test is performed (block-by-block or cell-by-cell).

Rte

(Runtime Environment) The RTE implements the Virtual Functional Bus and the execution of the SWCs. It also ensures consistent data exchange between the SWCs themselves and between the SWCs and the basic software. The execution of the basic software is realized by the integrated (SchM. The RTE also supports communication beyond partition boundaries (multi-core/trusted/untrusted). In addition, it offers simplified access to NVRAM data and calibration data. In safety-related ECUs, the RTE is one of the safety-related modules.

Rtm
(Runtime Measurement) You use the Rtm module to determine the runtime and CPU load of BSW modules and of application code. The module is typically used during development.

SCC
(Smart Charge Communication) This module is responsible for smart charge communication according to ISO 15118 or DIN 70121 and contains the V2GTP transport protocol that is used for this protocol. It supports DC and AC charging and the associated profiles Plug-and-Charge (PnC) and External Identification Means (EIM).

SchM
(BSW Scheduler) The SchM module is integrated in the (RTE, calls the periodic "main" function of the individual BSW modules, and provides functions for critical sections. For the distribution of the BSW (master satellite concept) via partitions and core boundaries, the SchM provides nearly identical communication interfaces like the (RTE.

Sd
(Service Discovery) Service Discovery was first specified in AR 4.1.1. An ECU communicates the availability of its services to communication partners via the publish-subscribe protocol implemented in this module. In addition, ECUs can register to receive automatic notifications, e.g. on a signal update.

SecOC
(Secure Onboard Communication) The SecOC is used to send or receive authenticated messages. Unauthorized, repeated or manipulated messages are detected. The SecOC is part of the AUTOSAR security solution.

She
(Secure Hardware Extension) The She driver implements the cryptographic hardware functions of a Hardware Security Module (HSM). This functionality is used by the (Csm module, for instance.

SoAd
(Socket Adaptor) The socket adaptor converts the communication via PDUs as it is defined in AUTOSAR into a socket-based communication. In AR 4.0, the SoAd also contains the diagnostic functionality defined in ISO 13400-2 ((DoIP). Since AR 4.1, this plug-in is removed and specified as a stand-alone module ((DoIP).

SomeIpXf
(SOME/IP Transformer) The module SOME/IP Transformer is an RPC and serialization protocol. It is used to call methods on other ECUs, which for example were made known in the system beforehand via Service Discovery ((Sd).

Spi
(SPI Handler/ Driver) The SPI Driver handles data exchange over the SPI interface. It is primarily used in conjunction with external hardware such as EEPROM, Watchdog, etc.

StbM
(Synchronized Time-Base Manager) The Synchronized Time-Base Manager enables time synchronization. Since AR 4.2 a time base prescribed by the Time Master can be synchronized

with other ECUs via bus systems.

TcpIp
(TCP/IP Stack) This module contains all protocols for UDP and TCP-based communication. It supports the versions IPv4 and IPv6 as well as parallel operation of IPv4 and IPv6 in one ECU. It contains the following protocols: - IPv4, ICMPv4 and ARP - IPv6, ICMPv6 and NDP - UDP, TCP, DHCPv4 (client) and DHCPv6 (client)

TLS
(Transport Layer Security) This module contains a Transport Layer Security client. The TCP-based communication is encrypted with TLS. The encryption algorithm used can be selected.

Tm
(Time Services) The Tm module is used for such tasks as measuring execution times and functions and implementing active waiting. It offers a resolution from 1µs to 4.9 days.

Ttcan
(TTCAN Driver) The TTCAN Driver offers the same functionality for a TTCAN controller (ISO 11898-4) as the CAN Driver ((Can) does for a CAN controller.

TtcanIf
(TTCAN Interface) The TTCAN Interface offers the same functionality for a TTCAN controller (ISO 11898-4) as the CAN Interface ((CanIf) does for a CAN controller.

UdpNm
(UDP Network Management) You use network management over UDP to implement synchronous transition to sleep mode for Ethernet ECUs.

Wdg
(Watchdog Driver) This module provides services for controlling and triggering the watchdog hardware. The trigger routine is called by the Watchdog Manager ((WdgM). For safety-related ECUs, the Wdg module must be developed according to ISO 26262.

WdgIf
(Watchdog Interface) This module enables uniform access to services of the Watchdog Driver ((Wdg), such as mode switching and triggering. For safety-related ECUs, the WdgIf module must be developed according to ISO 26262.

WdgM
(Watchdog Manager) The Watchdog Manager monitors the reliability and functional safety of the applications in an ECU. This includes monitoring for correct execution of SWCs and BSW modules and triggering of the watchdogs at the required time intervals. The WdgM module reacts to potential faulty behavior with multiple escalation stages. An important fact for safety-related functions according to ISO 26262 is the monitoring of correct flow sequences of critical tasks (logical supervision). For safety-related ECUs, the WdgM must be developed according to ISO 26262.

Xcp
(Universal Measurement and Calibration Protocol) XCP is a protocol for communication between a master (PC tool) and a slave (ECU). It is standardized by ASAM and is used

primarily for measuring, calibrating, flashing and testing ECUs. XCP supports the bus systems CAN ((CanXcp), FlexRay ((FrXcp), Ethernet ((EthXcp) and LIN ((LinXcp).

XML Security

(XML Security) You use this module to generate and validate XML signatures to EXI-encoded data based on the W3C XML security standard. It is used in the Vehicle2Grid environment.

5 Index

Get more Information!

Visit our Website for:

- News
- Products
- Demo Software
- Support
- Training Classes
- Addresses

www.vector.com