

SOLUTIONS FOR AUTOSAR



User Manual

Vector AUTOSAR Solution

A Step by Step Introduction

Version 4.1.2 for Release8

English

Imprint

Vector Informatik GmbH
Ingersheimer Straße 24
D-70499 Stuttgart

The information and data given in this user manual can be changed without prior notice. No part of this manual may be reproduced in any form or by any means without the written permission of the publisher, regardless of which method or which instruments, electronic or mechanical, are used. All technical information, drafts, etc. are liable to law of copyright protection.

© Copyright 2010, Vector Informatik GmbH. Printed in Germany.
All rights reserved.

Table of Contents

1	Manual Information	4
1.1	Manual History	5
1.2	Reference Documents	5
1.3	About this user manual	6
1.3.1	Certification	7
1.3.2	Warranty	7
1.3.3	Registered trademarks	7
2	Welcome to Startup Vector AUTOSAR Solution	9
2.1	What do you Learn from this Manual	10
2.2	Use Demo Application to Make your First Experience	10
3	ECU Development with AUTOSAR	12
3.1	ECU with AUTOSAR Approach	13
3.2	The Means of AUTOSAR	13
3.2.1	Software Components	13
3.2.2	Runnables	13
3.2.3	Ports	14
3.2.4	Data Elements	14
3.2.5	Connections	14
3.2.6	RTE	14
3.2.7	BSW	14
3.3	Data Exchange Formats	14
3.3.1	System Description	15
3.3.2	ECU Extract of System Description (ECUEX)	15
3.3.3	ECU Configuration Description (ECUC)	15
3.3.4	Complete Overview	15
4	Vector AUTOSAR Solution Step by Step	18
4.1	Situation before Starting	20
4.2	Vector AUTOSAR Solution	21
4.2.1	Project Assistant	21
4.2.2	The Tools	21
4.2.3	Three Tools Access one File - N:1 Scenario	21
4.2.4	Development Project	22
4.3	ECU Project Preparations – the Project Assistant	24
4.4	To Design Software Components and more...	28
4.4.1	Software Components	29
4.4.2	Ports, Port Init Values and Data Elements	31
4.4.3	Connect Ports	35
4.4.4	Delegation Ports	35
4.4.5	Define your Runnables	38
4.4.6	Can be Invoked Concurrently	39
4.4.7	Triggers for the Runnables	39
4.4.8	Port Access of the Runnables	40
4.4.9	Tasks and Task Mapping	41
4.4.10	The RTE and the RTE Generator	43
4.4.11	Data Mapping	44

4.5	BSW Configuration	46
4.5.1	Synchronize ECUC with DaVinci Developer	47
4.5.2	Configure BSW with DaVinci Configurator Pro	48
4.5.3	Tips when working with DaVinci Configurator Pro	49
4.5.4	Default settings for OS	49
4.5.5	Configuration of Generated Tasks, Events and Alarms	50
4.5.6	The SCHM - Create Task, Event and Alarm	52
4.5.7	Settings for the ECUM	53
4.5.8	Service Ports or not?	54
4.5.9	Generate Configuration Files and EcuM_swc.arxml	55
4.5.10	Usage of I/O - DIO as example	55
4.5.11	Settings for DIO	55
4.5.12	Configure ADC	55
4.5.13	Configure PWM	56
4.5.14	Settings for IOHWAB	57
4.5.15	Generate Configuration for DIO and IOHWAB	59
4.5.16	Memory via Service Ports	60
4.5.17	NVM and Fee Configuration	62
4.5.18	Configure Watchdog	62
4.5.19	Configure WDG	62
4.5.20	Configure WDGIF	63
4.5.21	Configure WDGm	63
4.5.22	Switch from Configurator Pro to GENy	63
4.5.23	Configure BSW for Communication with GENy	63
4.5.24	BSWs for Communication and Diagnostics	65
4.5.25	Settings for CAN – Can_CanoeemuCanoe	65
4.5.26	Settings for COM – PDUs	65
4.5.27	Settings for COM – Indication Functions when working with DBC file	65
4.5.28	Settings for CANTP	65
4.5.29	Settings for CANNM	65
4.5.30	Setting for CANIF	66
4.5.31	Settings for the COMM	66
4.5.32	Settings for DCM	67
4.5.33	Settings for DEM	67
4.5.34	Generate Configuration Files	68
4.6	Service Components	69
4.6.1	Client Server Theory	70
4.6.2	Service Components	71
4.6.3	Configure and Create Service Components	71
4.6.4	Import a Service Component into DaVinci Developer	72
4.6.5	What is within a Service Component	72
4.6.6	How can your Software Component use a Service Component?	73
4.6.7	Use Service Component DEM as an example	74
4.6.8	Manually	76
4.6.9	Port Access to Service Ports	76
4.6.10	IOHWAB is Application Component – an Exception	77
4.6.11	Port Access to new IOHWAB Services	78
4.6.12	Inter-Runnable Variables	78
4.6.13	Inside IoHwAb.c	79
4.6.14	Store Information non-volatile – NVM Manager	79
4.6.15	Per-Instance Memory	79
4.6.16	Usage of WDGm	80
4.7	Code Generation	81
4.7.1	Code Generation with DaVinci Developer	82
4.7.2	Component Templates and Runnables	84
4.7.3	Skeleton for runnables	84
4.7.4	A Skeleton Example – Runnable MySWC_Code	86
4.7.5	RTE	87

4.8	Generate Final Code	88
4.8.1	Setup Your Build Environment	89
4.9	Compile, Link and create your Executable	90
5	Additional Topics	91
5.1	SIP Check, Version Check, Generator Check	92
5.1.1	SIP Check	92
5.1.2	Generator Check	92
5.1.2.1	Libraries are delivered	92
5.1.2.2	Postbuild is used	92
5.1.3	Error Function in EcuM_CalloutStubs.c	92
5.2	Flash Bootloader Topics	93
5.2.1	Valid Application or Flash Bootloader and Invalid Application	93
5.3	DCM Template for Transition into Bootloader	93
5.4	Update ECUC via Project Assistant	94
6	What's new, what's changed	95
6.1	Version 3.7	96
6.1.1	What's new	96
6.1.2	What's changed	96
6.2	Version 3.8	97
6.2.1	What's new	97
6.2.2	What's changed	97
6.3	Version 3.9.0	98
6.3.1	What's new	98
6.3.2	What's changed	98
6.4	Version 4.0.0	99
6.4.1	What's new	99
6.4.2	What's changed	99
6.5	Version 4.1.x	100
6.5.1	What's new	100
6.5.2	What's changed	100
7	Address table	101
8	Glossary	103
9	Index	104

1 Manual Information

In this chapter you find the following information:

1.1	Manual History	Seite 5
1.2	Reference Documents	Seite 5
1.3	About this user manual	Seite 6
	Certification	
	Warranty	
	Registered trademarks	

1.1 Manual History

Author	Date	Version	Remarks
Klaus Emmert	2006-01-10	1.00	Created and reviewed
Klaus Emmert	2006-01-23	1.01	Updates and modifications
Klaus Emmert	2006-02-07	1.02	Some screenshot updates
Klaus Emmert	2006-09-29	1.03	Changes for new version of DaVinci
Klaus Emmert	2006-11-15	1.04	DaVinci Tool Suite Version 2.1
Klaus Emmert	2007-05-25	2.00	DaVinci Tool Suite 2.2, Port init values, service mapping
Klaus Emmert	2007-10-29	2.01	Manufacturer setting changed to VECTOR
Klaus Emmert	2008-08-22	3.0	Totally reworked, new template
Klaus Emmert	2008-08-27	3.1	Typos and issues fixed.
Klaus Emmert	2009-01-30	3.2	Typos and issues fixed.
Klaus Emmert	2009-02-27	3.3	<ul style="list-style-type: none"> > new handling of triggers and port access for runnables > Creation of ECU Extract of System Configuration via ND.
Klaus Emmert	2009-04-16	3.4	<ul style="list-style-type: none"> > DaVinci Configurator Pro > New file structure after installation > Document Rename UserManual_MICROSAR
Klaus Emmert	2009-04-27	3.5	Generator settings changed
Klaus Emmert	2009-06-16	3.6	DBC attributes file
Manuela Scheufele. Klaus Emmert	2009-06-23	3.7	(see section Version 3.7 on page 96)
Manuela Scheufele. Klaus Emmert	2009-09-10	3.8	(see section Version 3.8 on page 97)
Klaus Emmert	2010-01-13	3.9.0	(see section Version 3.9.0 on page 98)
Klaus Emmert	2009-01-20	4.0.0	(see section Version 4.0.0 on page 99)
Manuela Scheufele, Klaus Emmert	2010-03-01	4.1.x	(see section Version 4.1. on page 100)

1.2 Reference Documents

No	Source	Title
[1]	Vector	TechnicalReference_<BSW module>.pdf
[2]	Vector	Online Help of DaVinci Developer, DaVinci Configurator Pro, GENy
[3]	Vector	TechnicalReference _Wake-up_Sleep_with_AUTOSAR.pdf

1.3 About this user manual








Finding information quickly

The user manual provides the following access help:

- > At the beginning of each chapter you will find a summary of the contents,
- > In the header you can see in which chapter and paragraph you are,
- > In the footer you can see to which version the user manual replies,
- > At the end of the user manual you will find an index, with whose help you will quickly find information,
- > Also at the end of the user manual you will find a glossary in which you can look up an explanation of used technical terms.

Conventions

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

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	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.
	Step-by-step instructions provide assistance at these points.
	Instructions on editing files are found at these points.
	This symbol warns you not to edit the specified file.

1.3.1 Certification

Certified Quality Management System

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

Spice Level 3

The Embedded Software Components business area at Vector Informatik GmbH achieved process maturity level 3 during a HIS-conformant assessment.

1.3.2 Warranty

Restriction of warranty

We reserve the right to change the contents of the documentation and the software without notice. Vector Informatik GmbH assumes no liability for correct contents or damages which are resulted from the usage of the documentation. We are grateful for references to mistakes or for suggestions for improvement to be able to offer you even more efficient products in the future.

1.3.3 Registered trademarks

Registered trademarks

All trademarks mentioned in this documentation and if necessary third party registered are absolutely subject to the conditions of each valid label right and the rights of particular registered proprietor. All trademarks, trade names or company names are or can be trademarks or registered trademarks of their particular proprietors. All rights which are not expressly allowed are reserved. If an explicit label of trademarks, which are used in this documentation, fails, should not mean that a name is free of third party rights.

> Outlook, Windows, Windows XP, Windows 2000, Windows NT, Visual Studio are trademarks of the Microsoft Corporation.



Caution: All application code in any of the Vector User Manuals is for training purposes only. They are slightly tested and designed to understand the basic idea of using a certain component or a set of components.

2 Welcome to Startup Vector AUTOSAR Solution

In this chapter you find the following information:

2.1	What do you Learn from this Manual	Seite 10
2.2	Use Demo Application to Make your First Experience	Seite 10

2.1 What do you Learn from this Manual

The intention of the following descriptions is to get you familiar with the Vector AUTOSAR Solution. You learn to

- > create Software Components
- > configure BSW modules
- > generate code
- > program your runnables
- > ...

The aim is to provide you with the knowledge of how the AUTOSAR concept works and how these concepts are realized with the Vector AUTOSAR solution.

Additionally you should use the **demo application** from your delivery to play with and to gain valuable information and first experience.

Finally you will be able to setup and create your own project.

2.2 Use Demo Application to Make your First Experience

The demo application deals with a vehicle, its doors and interior lights. The lights are switched to on if at least one door is opened. The lights are switched to off if all doors are closed.

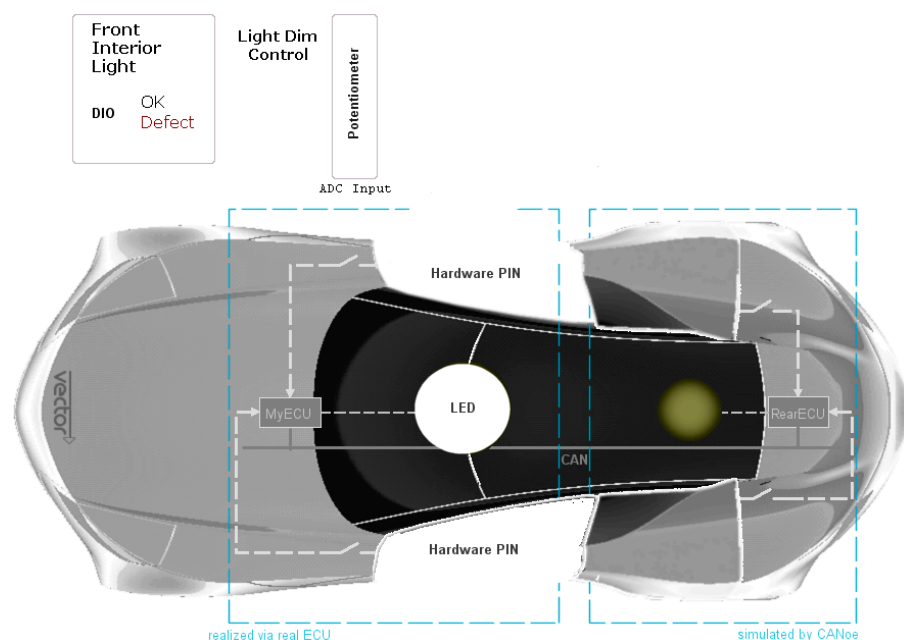
The lights have a dimmer functionality. The speed of the dimmer can be set via a slider (Light Dim Control).

It is a very simple application and good for your first approach.

CANoe Simulation of the vehicle

In the following illustration you see the vehicle as CANoe simulation. The slider **Light Dim Control** is an **ADC Input** and determines how fast the light is dimmed. The dimmer function itself is realized via a PWM.

Two ECUs, communication via CAN, defect front interior light can be simulated with your hardware PINs and causes DEM event.





Cross reference: You will find more about installation and usage of the demo application in the document **UserManual_AUTOSAR_Demo.pdf**.

3 ECU Development with AUTOSAR

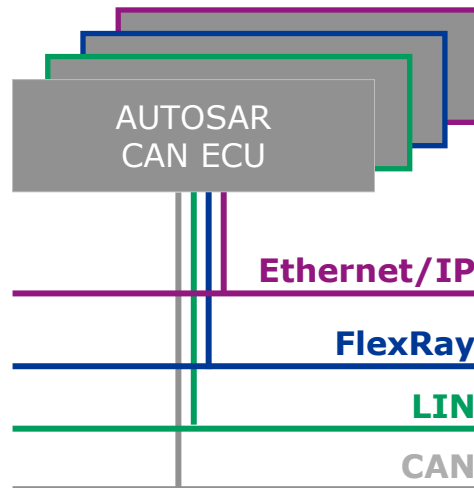
In this chapter you find the following information:

3.1	ECU with AUTOSAR Approach	Seite 13
3.2	The Means of AUTOSAR	Seite 13
	Software Components	
	Runnables	
	Ports	
	Data Elements	
	Connections	
	RTE	
	BSW	
3.3	Data Exchange Formats	Seite 14
	System Description	
	ECU Extract of System Description (ECUEX)	
	ECU Configuration Description (ECUC)	
	Complete Overview	

3.1 ECU with AUTOSAR Approach

Always keep in mind what you want to do. You want to create an ECU that is compliant with AUTOSAR and works for a certain bus system like CAN, LIN, FlexRay or IP. Or is it a gateway?

Your target is to create an ECU



3.2 The Means of AUTOSAR

In the following you find the means AUTOSAR provides to create an ECU. The illustration at the end of this chapter gives you a good overview.

3.2.1 Software Components

There are several kinds of software components.

The meanings will be explained later in this document.

In AUTOSAR everything starts with software components (SWC). Per definition they are independent from any ECU except for the sensor and actuator software components.

As you see there are several kinds of software components:

- > Application software components
- > Sensor and actuator software components
- > Service components
- > Compositions
- > Calibration (not explained here, please refer to online help)
- > I/O Hardware Abstraction (not explained here, please refer to online help)
- > Complex Drivers (not explained here, please refer to online help)

3.2.2 Runnables

Runnables – something that can RUN

Your code to define the behavior of the software components is realized within the runnables. You can configure when your runnable is called and which data it can access.

3.2.3 Ports

Ports are the gate to the outside world

Via ports a software component can communicate with the outside world. As you see there are several kinds of ports.

- > Sender port
- > Receiver port
- > Client port
- > Server port
- > Calibration port (not explained here, please refer to online help)
- > Mode port (not explained here, please refer to online help)

3.2.4 Data Elements

Data elements passes the port gates

Data elements of well-defined data types (e.g. 4bits, 1 byte, Booleans, records, etc.) can pass through the ports.

3.2.5 Connections

Connections

To define which software components communicate with each other their ports are connected via so-called connections.

A step towards the ECU

Doing the transition to real ECUs you have to decide which SWC runs on which ECU. Dependent on this mapping, the connections become internal ❶ or external ❷, ❸. (See the illustration at the end of this chapter).

3.2.6 RTE

VFB
(virtual function bus)
– RTE
(runtime environment)

The RTE controls the execution of the runnables, knows about external and internal way of data and controls access of runnables (your code) to the basis software modules (BSW).

3.2.7 BSW

BSW

The BSW is highly configurable and fulfils tasks like communication, diagnostics, fault memory, etc.

3.3 Data Exchange Formats

Data exchange formats

There are three main data formats.

- > System description
- > ECU Extract of System Description
- > ECU configuration description

3.3.1 System Description

System configuration description	<p>The System Configuration Description describes the complete system. It contains information dealing with:</p> <ul style="list-style-type: none">> Topology> Central software composition containing all software components of the system> Mapping> RTE and basic software> Communication> Gateways
---	---

3.3.2 ECU Extract of System Description (ECUEX)

ECU Extract of System Description	<p>Base for the ECU development is the ECU Extract of System Description. It contains the ECU-specific part of the system configuration description:</p> <ul style="list-style-type: none">> Collection of available SWC implementations (predefined by OEM)> Mapping of SWC to the selected ECU> System communication matrix, that completely describes the frames running on the networks and the contents and timing of those frames.
--	--

3.3.3 ECU Configuration Description (ECUC)

ECU configuration description	<p>The ECUC file will be created initially out of the ECU Extract of System Description and contains the following information:</p> <ul style="list-style-type: none">> Which BSW modules are within the ECU (only those that can be derived logically)> All configuration information for the communication BSW modules within the ECU> Communication matrix information (cannot be seen directly anymore, will be included into BSW module configuration).
--------------------------------------	---

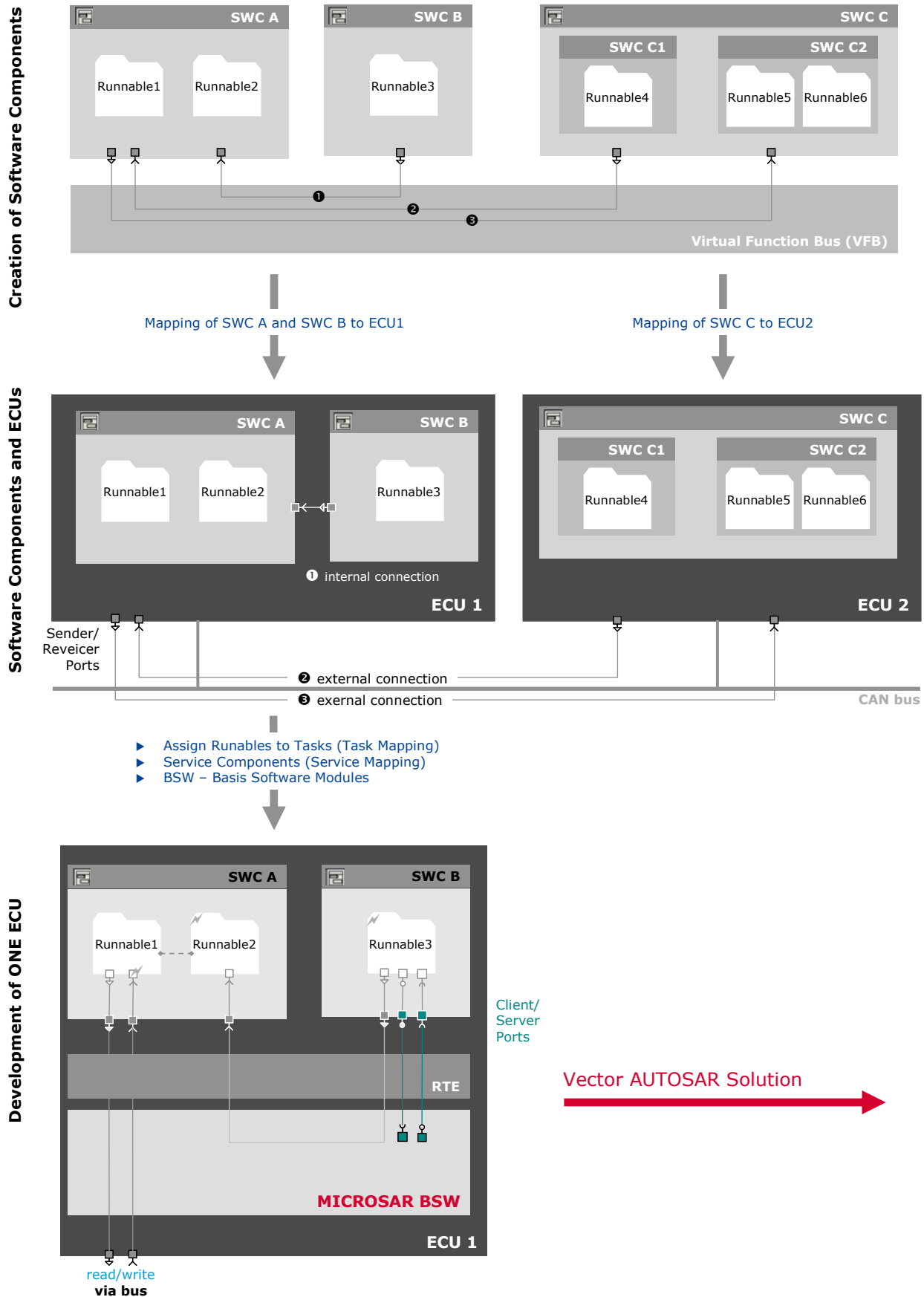
3.3.4 Complete Overview

Interaction of the shown means	<p>The following illustration shows the AUTOSAR concept graphically.</p> <p>Create Software Components</p> <p>Software components have to be designed - perhaps they contain further software components (composition).</p> <p>For the communication the components have ports and the ports can be connected with each other via the virtual function bus (VFB).</p> <p>Software Components and ECUs</p> <p>When the software components are mapped to ECUs, connections become internal when two connected software components are mapped to the same ECU ❶. In the other case when connected software components are mapped to different ECUs the connections become external ones ❷, ❸. External means that the information has to be transported via a bus system like CAN, LIN, FlexRay, MOST, IP,</p> <p>Development of ONE ECU</p> <p>When stepping towards one ECU the basis software modules (BSW) are added,</p>
---------------------------------------	---

configured and used by your software components via the RTE. A runtime environment with tasks, events and alarms is introduced. Your runnables have to be programmed and mapped to tasks. Then the complete project can be compiled, linked and used.

Vector AUTOSAR Solution

Read more about details of the Vector AUTOSAR Solution in the following chapter.



4 Vector AUTOSAR Solution Step by Step

In this chapter you find the following information:

4.1	Situation before Starting	Seite 20
4.2	Vector AUTOSAR Solution	Seite 21
	Project Assistant	
	The Tools	
	Three Tools Access one File - N:1 Scenario	
	Development Project	
4.3	ECU Project Preparations – the Project Assistant	Seite 24
4.4	To Design Software Components and more...	Seite 28
	Software Components	
	Ports, Port Init Values and Data Elements	
	Connect Ports	
	Delegation Ports	
	Define your Runnables	
	Can be Invoked Concurrently	
	Triggers for the Runnables	
	Port Access of the Runnables	
	Tasks and Task Mapping	
	The RTE and the RTE Generator	
	Data Mapping	
4.5	BSW Configuration	Seite 46
	Synchronize ECUC with DaVinci Developer	
	Configure BSW with DaVinci Configurator Pro	
	Tipps when working with DaVinci Configurator Pro	
	Default settings for OS	
	Configuration of Generated Tasks, Events and Alarms	
	The SCHM - Create Task, Event and Alarm	
	Settings for the ECUM	
	Service Ports or not?	
	Generate Configuration Files and EcuM_sw.xml	
	Usage of I/O - DIO as example	
	Settings for DIO	
	Configure ADC	
	Configure PWM	
	Settings for IOHWAB	
	Generate Configuration for DIO and IOHWAB	
	Memory via Service Ports	
	NVM and Fee Configuration	
	Configure Watchdog	
	Configure WDG	

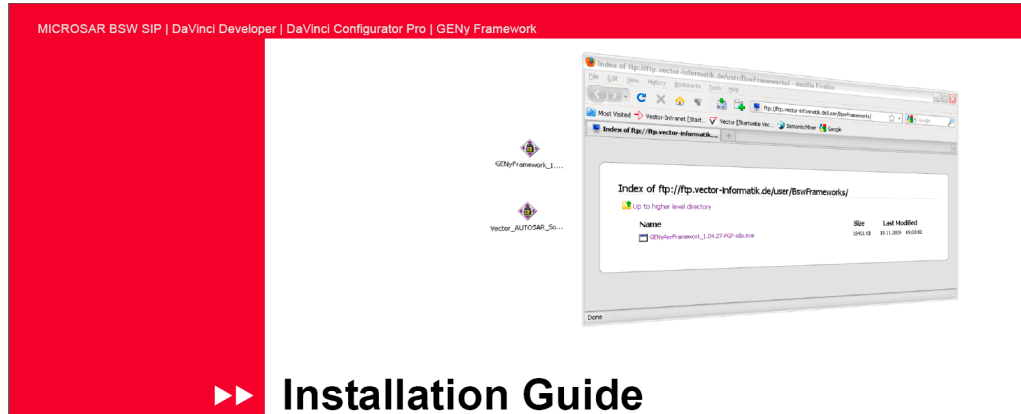
	Configure WDGIF	
	Configure WDGM	
	Switch from Configurator Pro to GENy	
	Configure BSW for Communication with GENy	
	BSWs for Communication and Diagnostics	
	Settings for CAN – Can_CanoeemuCanoe	
	Settings for COM – PDUs	
	Settings for COM – Indication Functions when working with DBC file	
	Settings for CANTP	
	Settings for CANNM	
	Setting for CANIF	
	Settings for the COMM	
	Settings for DCM	
	Settings for DEM	
	Generate Configuration Files	
4.6	Service Components	Seite 69
	Client Server Theory	
	Service Components	
	Configure and Create Service Components	
	Import a Service Component into DaVinci Developer	
	What is within a Service Component	
	How can your Software Component use a Service Component?	
	Use Service Component DEM as an example	
	Manually	
	Port Access to Service Ports	
	IOHWAB is Application Component – an Exception	
	Port Access to new IOHWAB Services	
	Inter-Runnable Variables	
	Inside IoHwAb.c	
	Store Information non-volatile – NVM Manager	
	Per-Instance Memory	
	Usage of WDGM	
4.7	Code Generation	Seite 81
	Code Generation with DaVinci Developer	
	Component Templates and Runnables	
	Skeleton for runnables	
	A Skeleton Example – Runnable MySWC_Code	
	RTE	
4.8	Generate Final Code	Seite 88
	Setup Your Build Environment	
4.9	Compile, Link and create your Executable	Seite 90

4.1 Situation before Starting

Check your installation

Before you start your first steps with Vector AUTOSAR Solution check whether you have installed all necessary tools and software.

Installation Guide from BSW module delivery on CD or via FTP



Installation Guide

- > Have you read and used the **InstallationGuide.pdf**?
- > You have received and installed all necessary tools mentioned there?
- > You have installed the BSW modules?

NO?

Then please read this document carefully and follow the installation hints given there.



Info: You find the document either on the installation CD of the BSW modules delivery or in the folder where you have unpacked the BSW module installation from Vector FTP server.

YES?

Then you are ready to go on and start with your first steps with the Vector AUTOSAR Solution.

4.2 Vector AUTOSAR Solution



Info: The illustration on the following page gives a complete overview of the Vector AUTOSAR Solution that is described in detail in the following chapters.

Short summary

The Vector AUTOSAR Solution has the focus on developing an ECU. You need the tools **DaVinci Developer**, **DaVinci Configurator Pro** and **GENy**. The ECU extract of system description is the initial file format. The ECU configuration description is derived from ECU extract of system description once and with every update. The tools work on the ECU configuration description to exchange information.

4.2.1 Project Assistant

Setting up a new project – Project Assistant

The easiest way for starting up a new project is using the **Project Assistant**. It collects all necessary administrative information like paths, licenses, etc. and helps creating the first project files very easily.

4.2.2 The Tools

DaVinci Developer Version 3.0

The **DaVinci Developer** is used to create software components, service components ..., for short, to configure the application part of the ECU. It generates code for the software components that has to be finished by the programmer.



Info: **DaVinci Developer** for Release 8 in Version 3.0

DaVinci Configurator Pro Version 4.0 SP2

With the **DaVinci Configurator Pro** you configure all non-communication BSW modules and generate configuration files for them.



Info: **DaVinci Configurator Pro** for Release 8 in Version 4.0 SP2

GENy

GENy is the well-known tool to configure the communication-specific BSW modules and to generate appropriate configuration files.

All tools access the ECU Configuration Description to store and read necessary configuration information.

4.2.3 Three Tools Access one File - N:1 Scenario

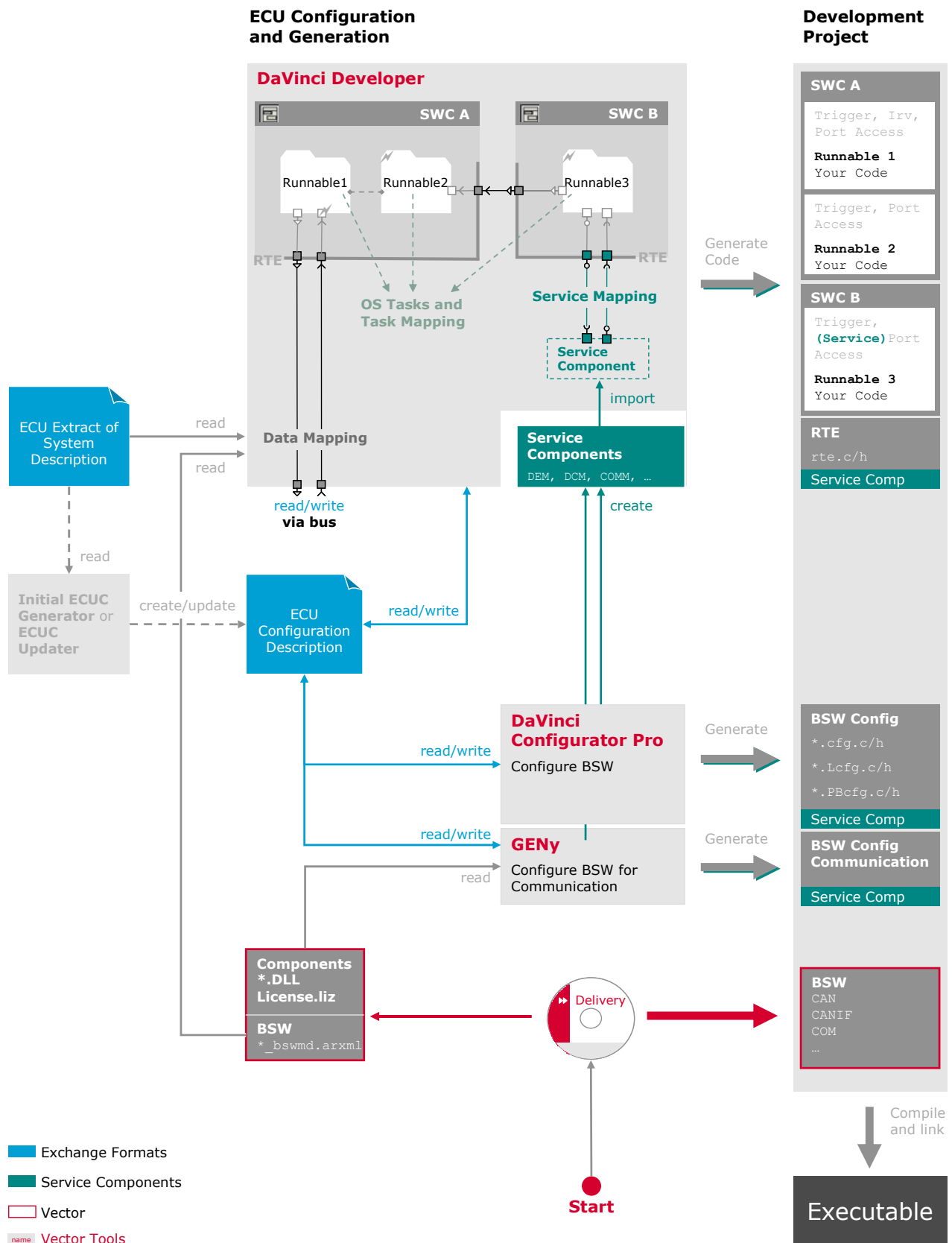
Data consistency must be kept

As three tools access one ECUC file, there must be a mechanism, that modified information of one tool is stored before another tool also modifies the ECUC file. For that reason there is a control instance implemented that checks whether you first have to store your changes before you can work with another tool. The other tools are in read-only mode for this period in time.

4.2.4 Development Project

Your Project

This is your project where all configuration files, template files, etc. are generated to and where you do your programming work to fill the empty runnables with life. Then compile and link and get your executable that you can download to your hardware.



All necessary parts
of this illustration are

In the following this overview illustration is explained in small portions. Try to understand all influences when changing different settings. The portions will help you

explained in the
following

later on working easy and well-planned on your project.

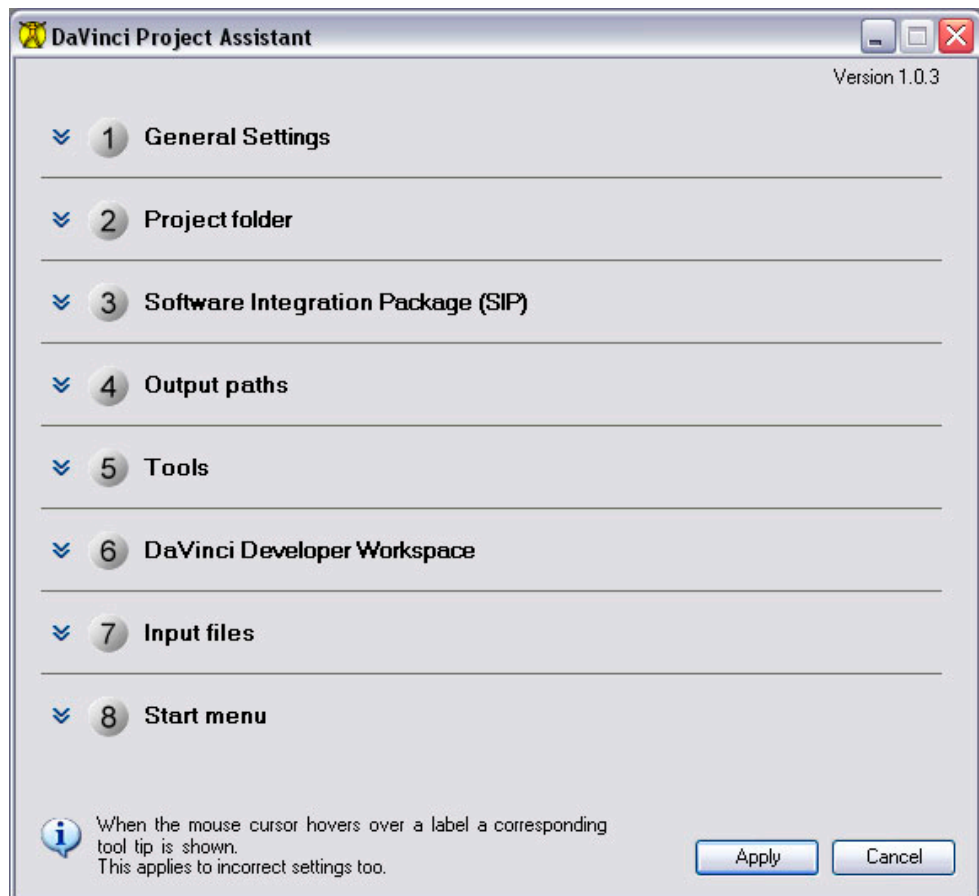
4.3 ECU Project Preparations – the Project Assistant

Start the **DaVinci Developer** and activate the **Project Assistant** via **File|Project Assistant|New ECU Project....**



Caution: Projects created with the Project Assistant cannot be moved. They have to stay at the same path. Take care when working on different PCs or working via network.

DaVinci Project
Assistant



General Settings

- > **Project Name**
The name of the project (e.g. MyECU). It will be used for the naming of the generated ECUC files and for the DaVinci Workspace.
- > **Author (optional)**
- > **Version (optional)**
- > **Description (optional)**

Project folder

This is the **root path** of your project. Any other path is given relatively and refers to this root.

Software Integration Package (SIP)

Select a registered SIP or enter a path to an unregistered SIP that should be used for the new ECU project. In the latter case select the path until the **Component** folder. Select **platform**, **compiler** and **derivative** supported by the currently selected SIP.

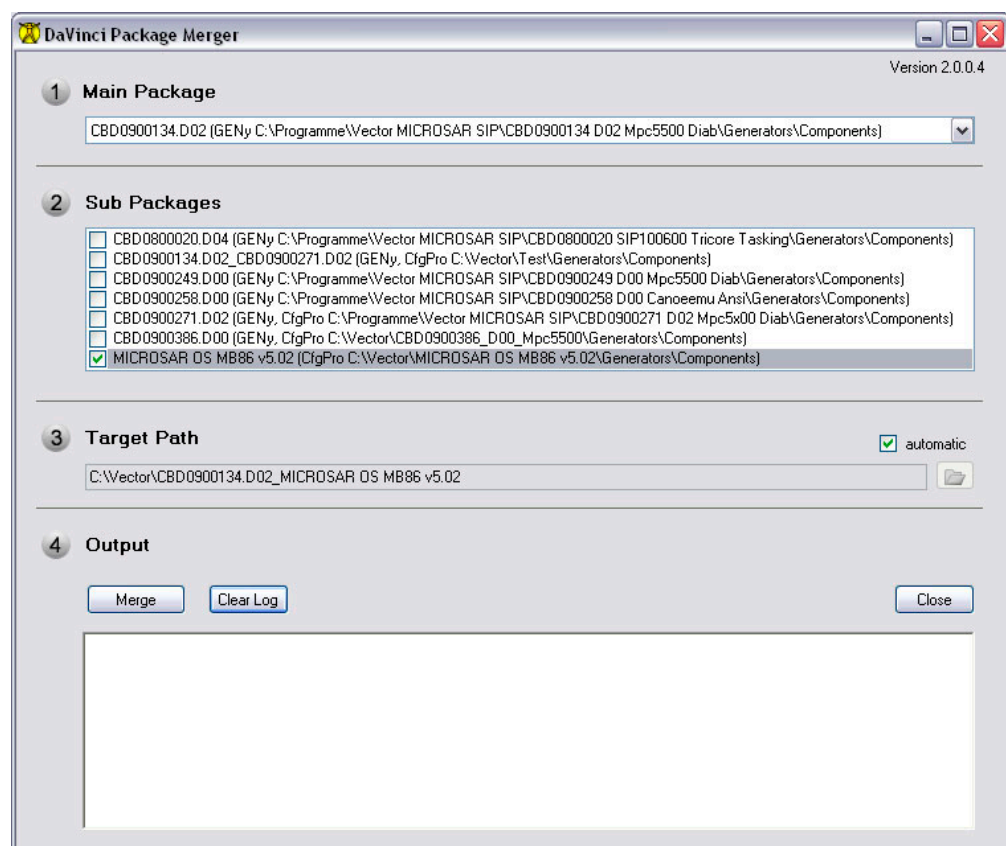
You can also create a new registered SIP based on existing packages. Therefore open the DaVinci Package Merger via **[Create new SIP]**.

DaVinci Package Merger

The DaVinci Package Merger enables the merge of different deliveries (SIPs) and creation of a new registered SIP.

The process includes the merge of BSW (source code, generators, BSWMD, DLLs), documentation, preconfiguration files and license files.

Additionally all registry entries are created for DaVinci Configurator and GENy.



Select **Main Package**, **Sub Packages** and enter **Target Path**. If the checkbox automatic is set, the package merger creates the target path automatically based on Main Package and Sub Package names.

Click **[Merge]** to start the merge process.

[Close] the package merger to return to the project assistant. Select the new registered SIP in list box.

3

Software Integration Package (SIP)

☒ Registered

CBD0900134.D02_CBD0900271.D02 [C:\Vector\CBD0900134.D02_CBD0900271.D02]

☐ Unregistered

CBD0900131_D03 Mcs12x Cosmic [C:\Programme\Vector MICRO SAR SIP\CBD0900131_D03]
CBD0900134.D02_CBD0900271.D02 [C:\Vector\CBD0900134.D02_CBD0900271.D02]
10.07.05.00.90.01.31.03.00.00 [C:\Programme\Vector MICRO SAR SIP\CBD0900134.D02_CBD0900271.D02]
CBD0700343 [C:\Programme\Vector MICRO SAR BSW\CBD0700343_R04_Mpc550]
CBD0900134.D02_CBD0900271.D02 [C:\Vector\CBD0900134.D02_CBD0900271.D02]

Platform:

Compiler:

Derivative:



Example:

A Customer possesses a SIP based on SLP 10. Additionally the customer possesses/orders a SIP for XCP and OS.

The following merge combinations are possible:

SLP 10 SIP +XCP SIP

SLP 10 SIP + OS SIP

SLP 10 SIP + XCP SIP + OS SIP

Output paths

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

Tools

Enter the path to the tool executables (if available)

- > DaVinci Configurator Pro
- > DaVinci Developer
- > GENy

DaVinci Developer Workspace

If you have a RTE and you use the **DaVinci Developer** define the path of your workspace, a new one or an already existing one.

Input files

Add your **ECU Extract of System Description** file. If you want to create a multiple ECU (e.g. an ECU for the doors, front left and front right) you have to add all necessary ECU Extract of System Description files.

Start menu

Select **Create entries in the start menu** to open your tools and the project assistant for viewing and updating the project from the **Windows** start menu.

Click **[Apply]** to start the creation process of the project assistant. Incorrect settings are marked with red.

The project assistant automatically creates multiple files in the path of the ECU Extract of System Description (using initial ECUC generator):

Files	meaning
<name>.ecuc.vdsxml	to be read in by GENy , contains references to ecuc.Initial.arxml and <name>.ecuc.arxml (marked with *)

Files	meaning
<name>.ecuc.arxml	to be read in by DaVinci Developer , DaVinci Configurator , GENy
<name>.error.txt	Log file as result of ECUC generation (If you choose the batch file solution the log file is your defined document)
<name>.ecuc.Initial.arxml	Initial ECU Configuration Description for internal tool use only
AUTOSAR_EcucParamDef.arxml	Standard file which is provided by AUTOSAR



Info: In case of error while the Project Assistant is working, refer to the log file for the error reason.



Info: The Project Assistant can be also opened via **DaVinci Configurator Pro** and **GENy**.

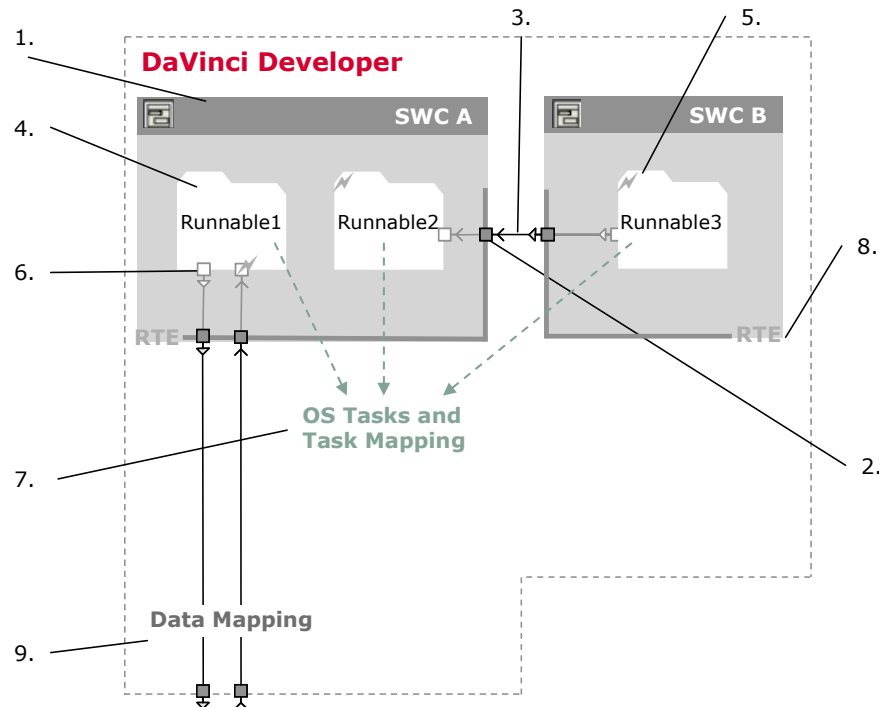
4.4 To Design Software Components and more...

In the first step you learn all about software components, ports, connection and runnables, how they work and how to exchange information between software components.



Detailed illustration
for designing
software components

Open **DaVinci Developer** if not already open. To get basic tool handling information refer to the online help of the **DaVinci Developer**.



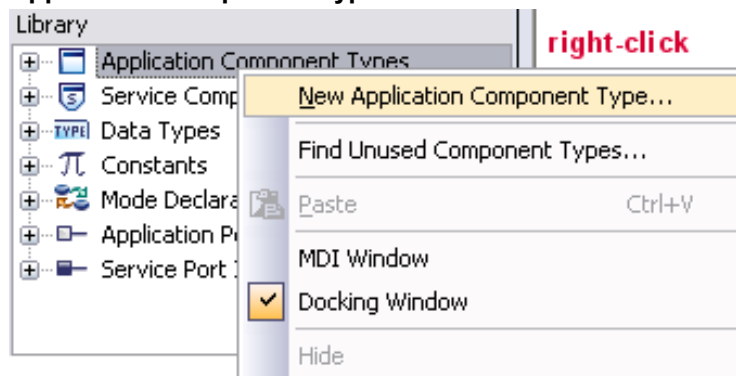
1. **Software Components**
(see section Software Components on page 29)
2. **Ports and Data Elements**
(see section Ports, Port Init Values and Data Elements on page 31)
3. **Connections**
(see section Connect Ports on page 35)
4. **Runnables**
(see section Define your Runnables on page 38)
5. **Triggers**
(see section Triggers for the Runnables on page 39)
6. **Port Access**
(see section Port Access of the Runnables on page 40)
7. **Tasks and Task Mapping**
(see section Tasks and Task Mapping on page 41)
8. **The RTE**
(see section The RTE on page 43)
9. **Data Mapping**
(see section on Data Mapping page 44)

4.4.1 Software Components

To create Software Components using **DaVinci Developer**.

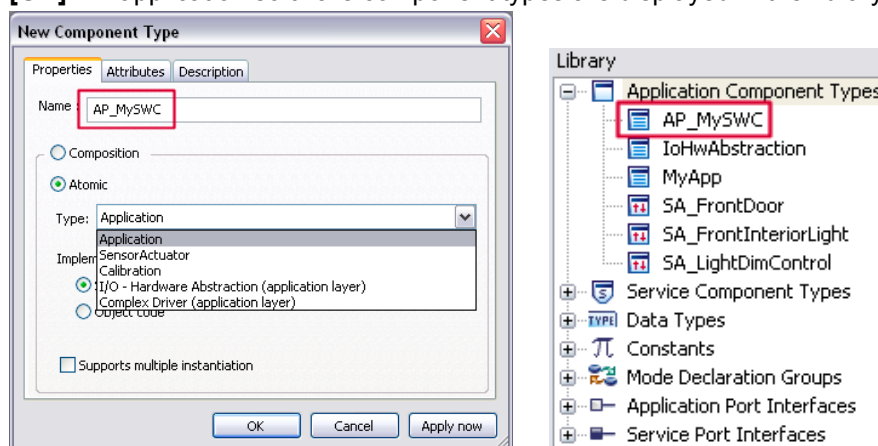
Define new
Component Types
in the **Library**

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

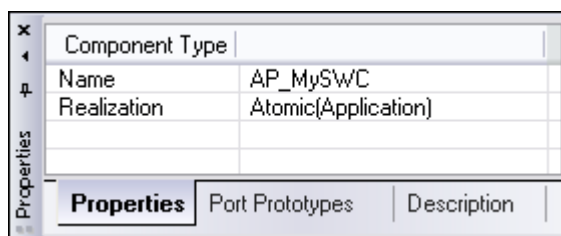


It is good to give a
speaking name to
see what kind of
component it is. e.g.
AP – Application
SA – Sensor /
Actuator

Enter a **Name** of the **Application Component Type**, select its **Type** and confirm with **[OK]**. All application software component types are displayed in the library.

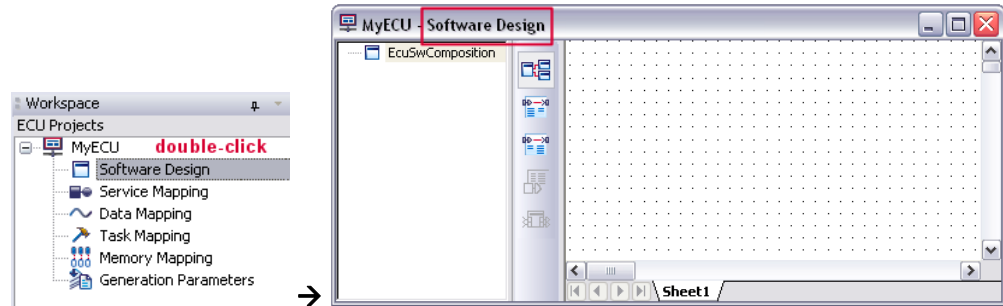


Info: To get fast information about any element in the library just select one and see its **Properties**, **Port Prototypes** (if available) and **Description** in the **Properties** view on the bottom.



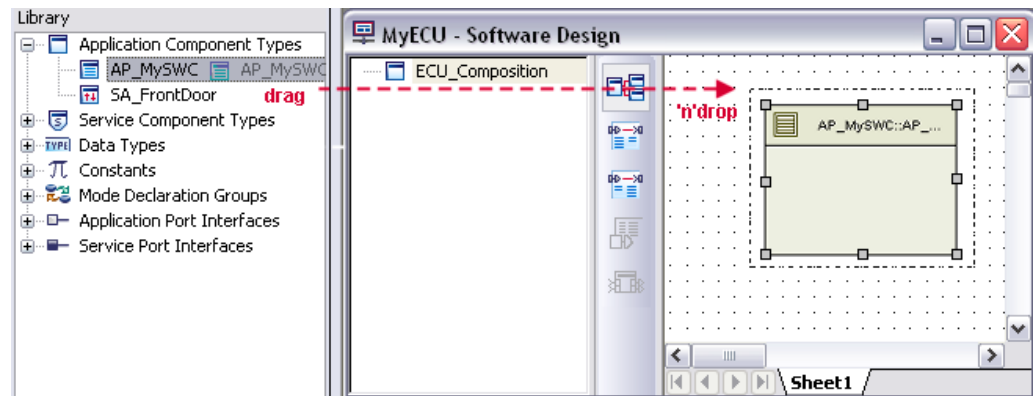
Component types
become component
prototypes by being
used

Double-click **Software Design** in the **ECU Projects** view to open the **MyECU Software Design** view.



To use the previously defined application component types (e.g. AP_MySWC) within the software design view just drag'n'drop them from library.

Drag'n'drop the application software component type to the software design view



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

Name of prototype

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.



Caution: When you use the restriction defined by AUTOSAR (32 characters) then keep the names as short as possible.





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

4.4.2 Ports, Port Init Values and Data Elements

To communicate and to exchange information the components need so-called ports (S/R ports). Ports are the gate for the data elements that carry the information.

For communication between software component ports have to be defined.

There are four different kinds of application ports,

- > **Sender Ports** to provide information 
- > **Receiver Ports** to receive information 
- > **Server Ports** to provide services (operations) 
- > **Client Ports** to use services (operations) 
- > **Calibration Ports** to hand over calibration parameters
- > **Mode Ports** to e.g. trigger or not trigger runnables within certain modes

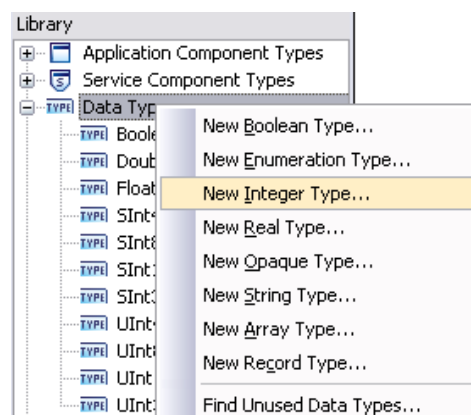


Info: You find more about service ports later in this manual (see section [Service Components](#) on page 69). The latter two port types are listed here for the sake of completeness.

Before you can use the ports you have to define **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

The library contains predefined **data types** to be used. But you can also define your own data types. Just click right and select out of the list.

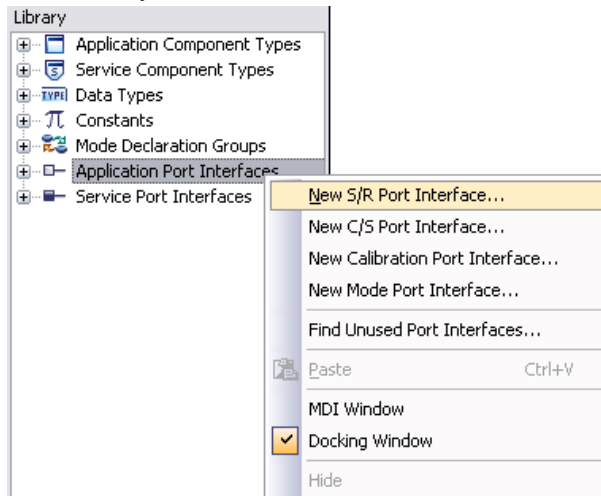


In the demo only the data type **Boolean** is used for door **open** and **closed** respectively **on** and **off** of the interior lights.



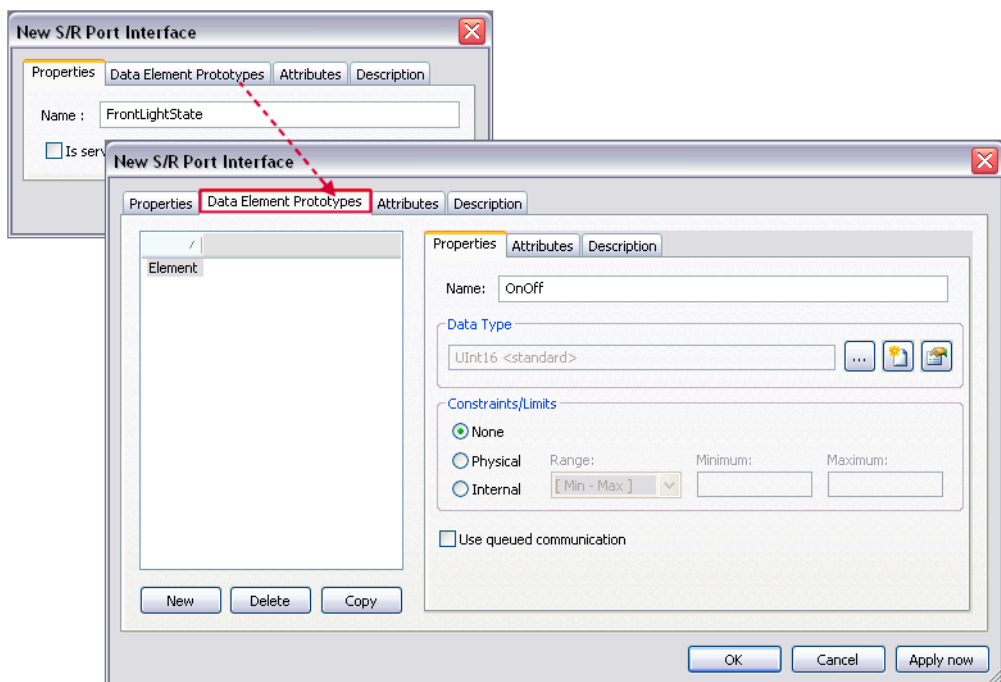
Caution: With port interfaces it is the same as for component types. You define the **port interfaces** in the library and then use them as port **prototypes** for each software component. The same applies for data types and data elements.

Create Port Interface To define new application port interfaces **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 type **Boolean**.

Data elements are the contents of ports.



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

Repeat this for all port interfaces you need.

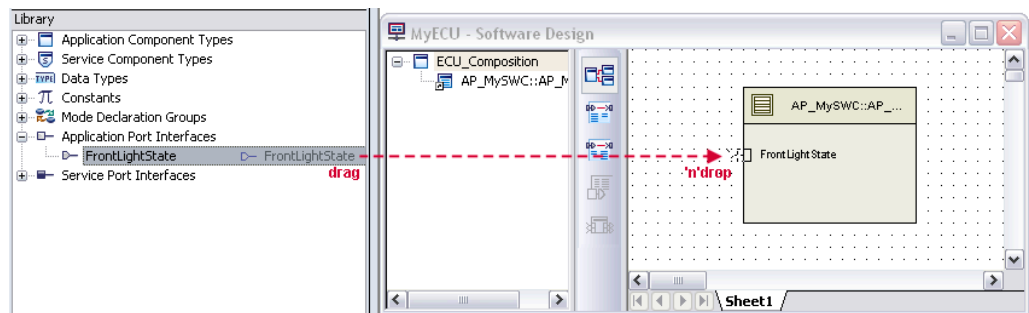
Provide software components with ports

- > 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 software component needs which port prototype.

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

Select a port interface from the library and place it via drag'n'drop onto the software component.



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

Interfaces.

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

> Sender port

> Receiver port

Do this for all necessary ports.

Ports need init values.

Port Init Values

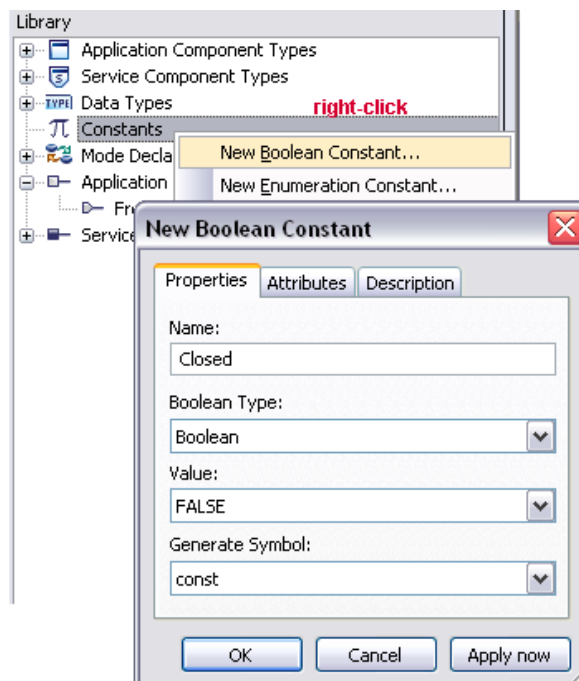
You have to assign an initial value to every used port. This value must be a constant. You can use the library to define constants first and then assign them as init value of a port or you define the constants while you assign init values.

Let's take the first way and use the library.

Constants

Right-click **Constants** in the library and select your data type from the content menu.

Port Init values are constants.

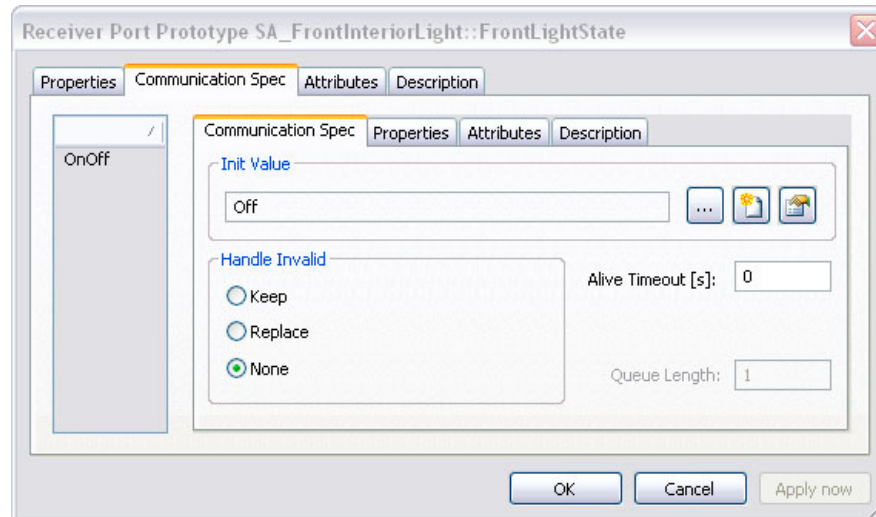


Define as many constants as you need.

Both ports of the communication need a port init value.

Double-click every port prototype and select the tab **Communication Spec**. Click [...] and search for the suitable **Init Value** from the list.

Do this for all of the ports. **On both sides of a connection!**



If there is no suitable **Init Value** because you did not define them as constant (as shown above) then click the icon **Create New** and define a new **Constant**.

4.4.3 Connect Ports

Data elements can only be transported through the ports to another software component if the ports are connected via connectors.



Draw connector

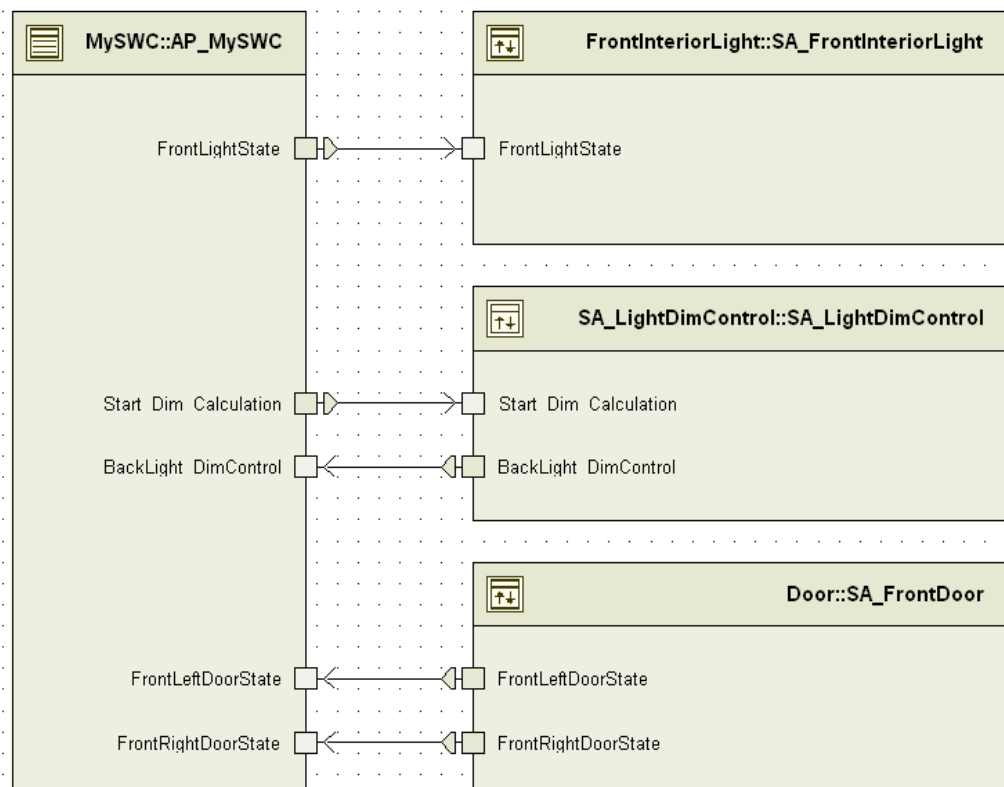
You can draw connectors manually just by connecting the appropriate ports.

Use the button **Draw Connector** to connect ports manually. Just click on the start port and then on the target port.



Automatically create connector prototypes

You can also let **DaVinci Developer** draw the connectors automatically. The **Developer** will draw those connectors where the port interfaces are compatible.



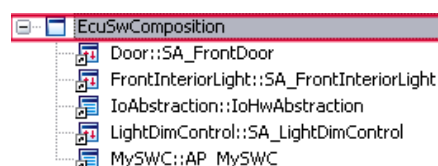
Which ports can be connected?

To connect ports, they have to be compatible. To decide whether ports are compatible or not, their content, the data elements are important and have to match.

4.4.4 Delegation Ports

All in one composition

All software components of an ECU are embedded within a composition called **EcuSwComposition** (see left side the tree root).



Design delegation ports (to communication with the outside of the composition) manually.

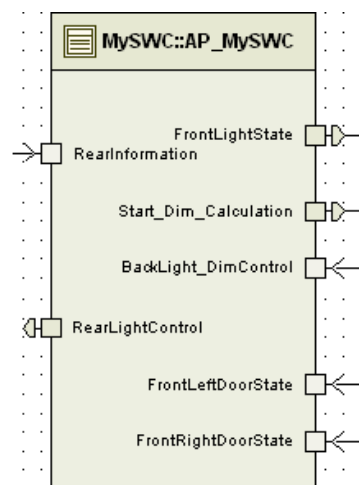
The communication between the components is designed and can be seen in the illustration above. Keep in mind, what you see is **one ECU**.

Where is the connection to send information to other ECUs or to receive information from other ECUs? This has to be defined using delegation ports.

In the demo example the connection between MyECU and the RearECU is done via CAN. The component within MyECU that has to read information from and send information to the RearECU is MySWC. So MySWC needs additional ports for this connection.

Collect the information sent via the bus into the necessary data elements.

Provide another data element for information received via the bus.



RearLightControl – Sending Data

Create a port interface for switching the rear interior light, i.e. for sending information via the bus. It needs a data element of the same type as the bus signal. In this case it is a boolean type. In the demo it is called **RearLightControl**.

Drag'n'drop this port interface to the software component.

Collect all three information units into the port. Define three data elements.

RearInformation – Receiving Data

Do the same as before for sending data now for data you receive from the bus. Create a port interface for reading the information of the RearECU and provide as much data elements with the suitable data type as you have signals. In the demo this is the state of the RearInteriorLight, the RearRightDoor and the RearLeftDoor. Therefore it needs three data elements of Boolean type. Let's call them

- > RearInteriorLight
- > RearLeftDoor
- > RearRightDoor

Drag'n'drop this port interface to the software component.

Think of port init values

Remember to set the **Init Values** for the new port prototypes and their data elements.

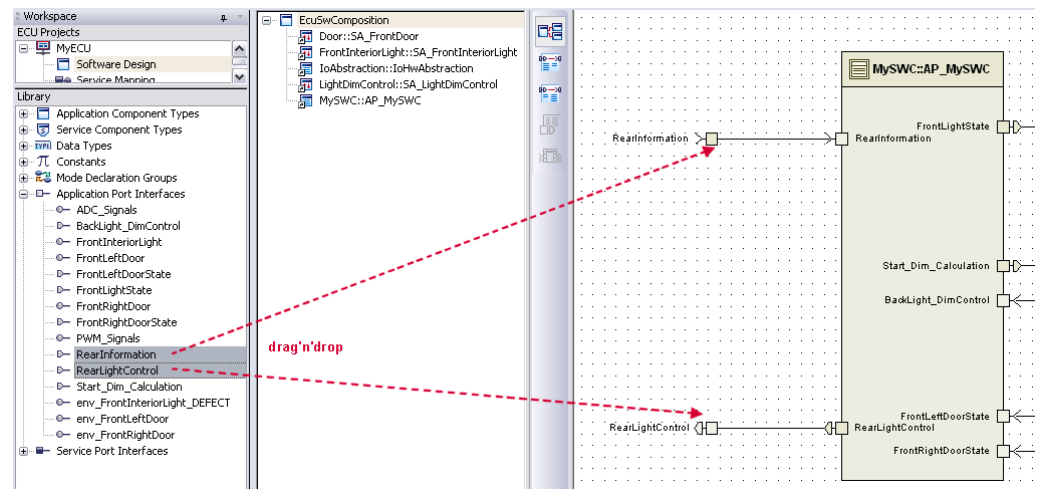


Caution: There is still something missing to communicate with the outside of the **COMPOSITION EcuSwComposition**.

Delegation ports necessary

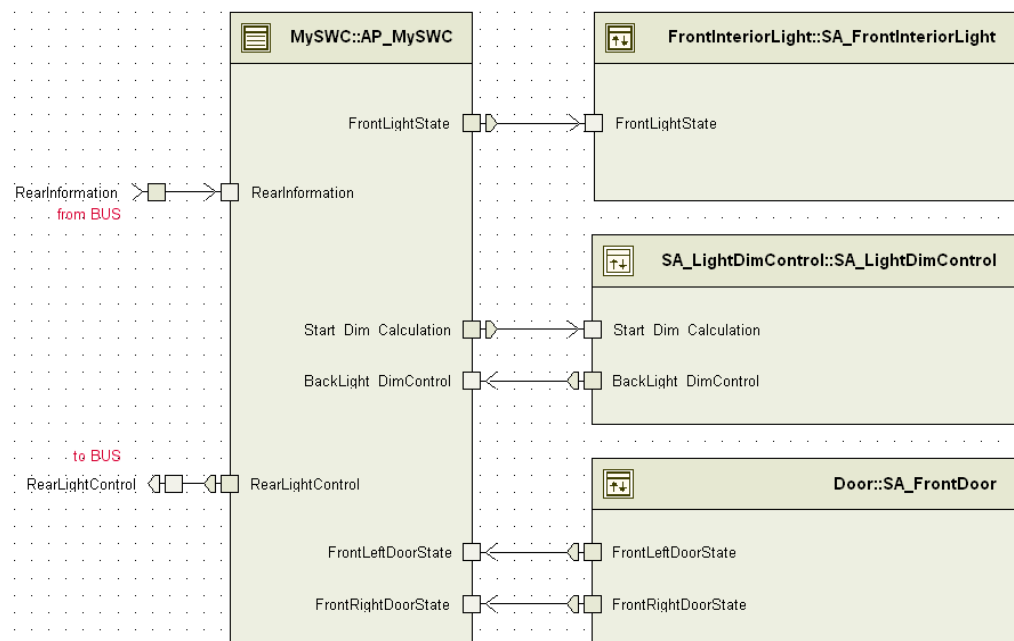
Drag'n'drop the delegation ports on the free area in the software design view

You have to create two new port prototypes at the border of the composition. These ports make the connection to the outside of your ECU, i.e. to the bus.



Draw the two necessary connectors.

The result could look like below, taking the demo as example.



Info: You can also right-click a software component and select **Complete Delegation** ports to let the **Developer** create them automatically.

4.4.5 Define your Runnables

Runnables are like functions that are called by the RTE in a configurable way and at a predefined point in time.

Name, Trigger, Port Access, Mapping

For short, there are four important topics for a runnables:

- > **SYMBOL** and **NAME** – how is the corresponding function called
- > **PORT ACCESS** – what data can be accessed
- > **TRIGGER** – when is the runnable executed
- > **MAPPING** – in which task context does the runnables work?

One or more runnables

For atomic software component you can define one or more runnables.



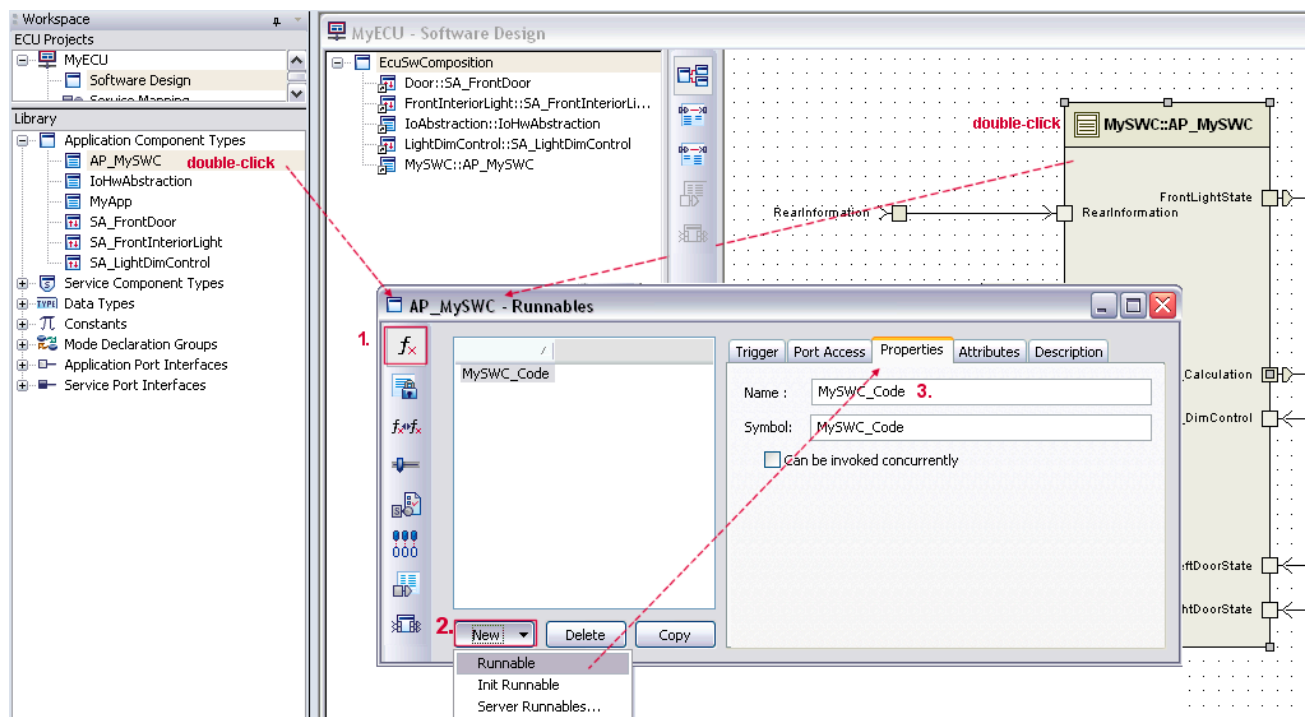
Caution: Runnables can only be defined for atomic software components.



Info: The runnable skeletons are generated by the **DaVinci Developer** (see section Code Generation with DaVinci Developer on page 82) and can then be filled with your code (see section Skeleton for runnables on page 84). More details follow in the next chapters.

f_x

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



If you leave the **Symbol** field empty, the runnables (function) is named according to the entry in the **Name** field.

4.4.6 Can be Invoked Concurrently

Criteria for re-entrance

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

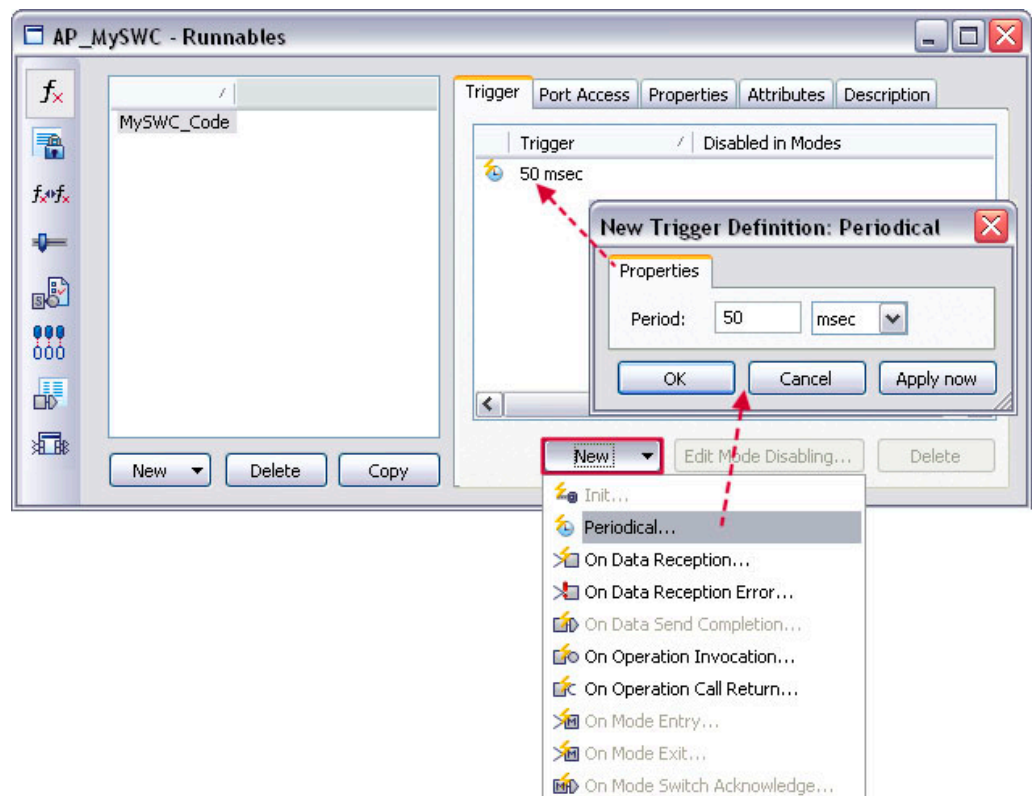
- > No static (or global) non-constant data
- > No return of address to static (or global) non-constant data
- > Only work on data provided by caller
- > No modification of own code
- > No call of non-reentrant runnables

4.4.7 Triggers for the Runnables

The trigger decides when a runnable is executed.

Define the triggers for your runnables.

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





Info: The trigger can be a periodical one or an event.

Periodically: Select the checkbox and enter the cycle time.

On Data Reception: As soon as data is received at a 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.



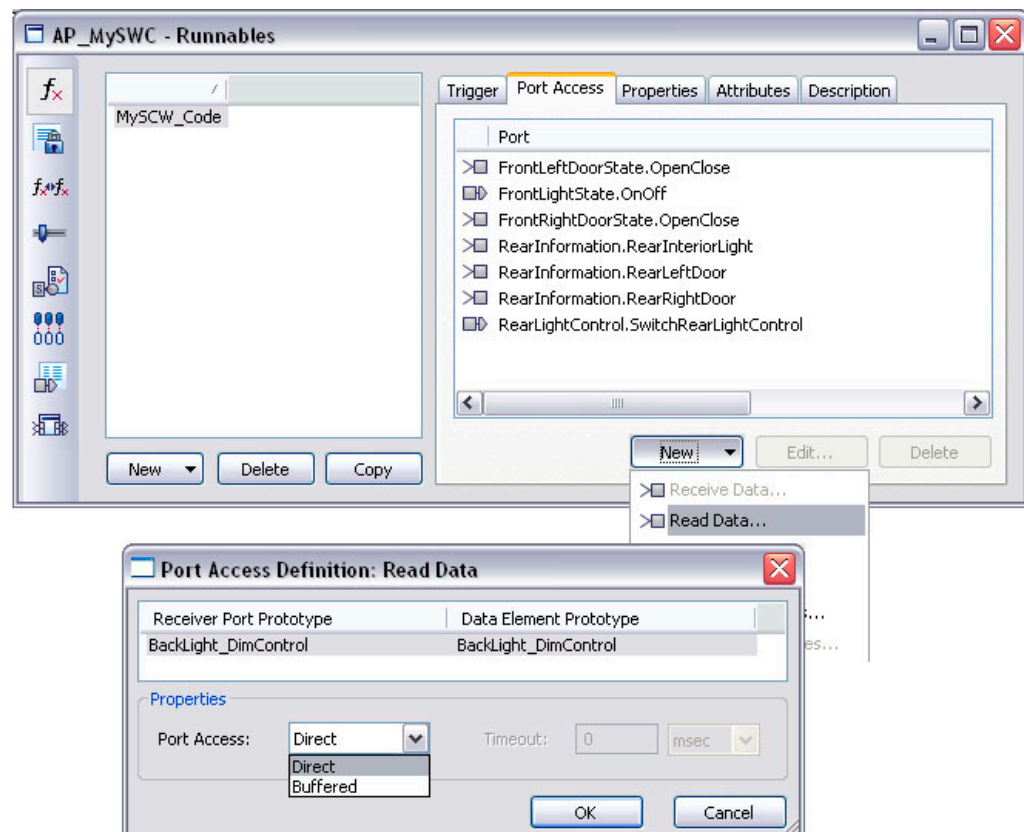
Info: When experimenting with the demo, replace the cyclic trigger with two triggers **On Data Reception...**, one for the left and one for the right front door.

4.4.8 Port Access of the Runnables

Define which port information a runnable should be able to read or write.

Select the tab **Port Access**. You only get displayed accessible ports.

PORT ACCESS –
define the interfaces
that should be used
by your runnable



You can define access to

> **Read Data...** and

- > **Write Data...**
- > and later when using service components also to operations (Invoke Operations - see section [Port Access](#) to Service Ports on page 76).

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 meanwhile. 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 is the same value until the runnable is left.



Info: 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 **Developer** and check, whether the **Port Access** is set correctly.

Summary

Summary of the settings for our runnable MySWC_Code:

- > The runnable is called MySWC_Code
- > Its functional representation is called: MySWC_Code.
- > It has to be mapped to a task (can be invoked concurrently not set) and
- > 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.

4.4.9 Tasks and Task Mapping

The Vector AUTOSAR Solution provides an intelligent RTE generator. It detects not mapped runnables that have to be mapped to task.

Very simple rule

There is a very simple rule that helps you with the task mapping.

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

A runnable that should be mapped is not mapped...

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

The RTE generator checks necessary conditions and will warn you if there are unmapped runnables that have to be mapped.

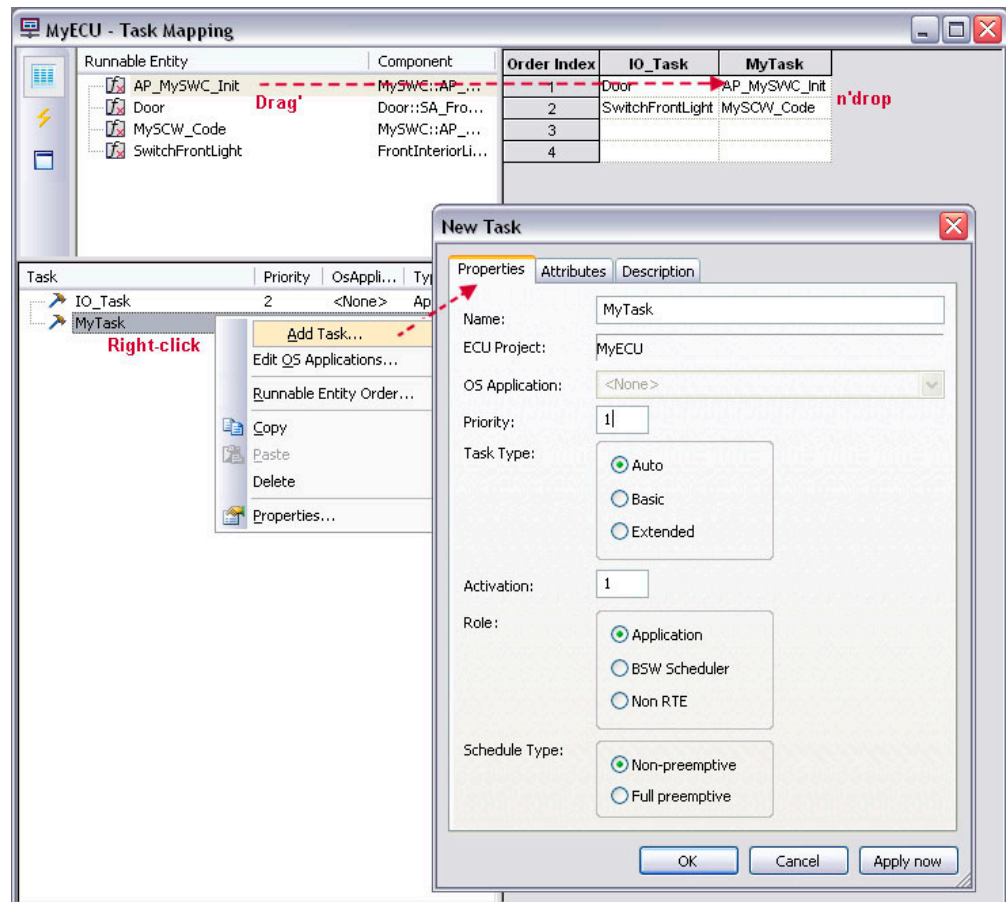
What happens if you map a runnable that has not to be mapped?

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.

Tasks and runnables To create tasks and to assign runnables to tasks manually open the **Task Mapping** view.



Create a new task Right-click in the left view below **Task** and select **Add Task...** from the pull-down menu. The **New Task** window opens. Fill-in the name of the task and its priority. You can influence the RTE generator by selection the **Task Type** to be **Auto**, **Basic** or **Extended**. The **Role** can set to **Application** (task skeleton generated by RTE generator), **BSW Scheduler** and **Non RTE** (with latter both settings, the task skeleton is not generated by the RTE generator).

Decide about **Full-preemptive** or **Non-preemptive** and confirm with **[OK]**. The new task is listed in the left view and also displayed on the right side (see arrows).

Mapping of runnables




Now assign your runnables to the tasks via Drag'n'drop.



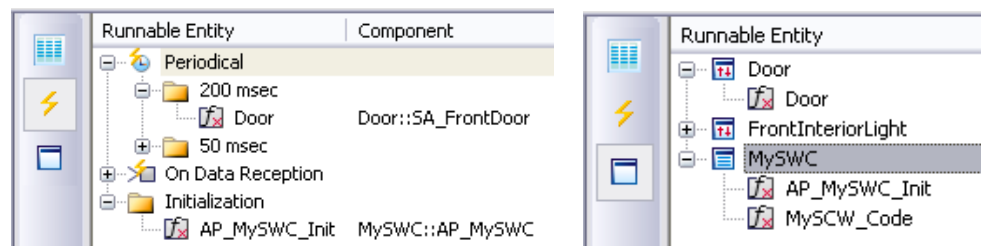
Info: The RTE generator uses the information of the tasks, the runnables that are mapped to the task and the triggers of each runnable when generating the RTE.

Different view for runnable entity:

There are three views for the runnables. The list view can be seen in the screenshot above.

-  List view
-  Trigger View
-  Component View

The **Trigger View** and the **Component View** are shown below.



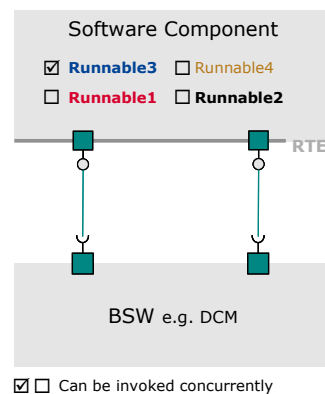
4.4.10 The RTE and the RTE Generator

Code optimization as often as possible

Background Story: Runnable, Reentrance and Task Mapping:

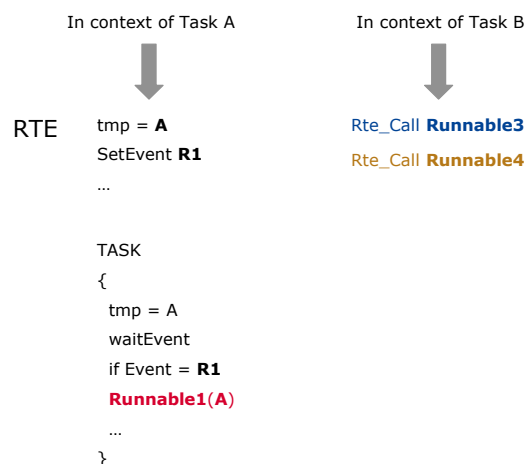
Runnables, reentrance and mapping

Software Design



Task Mapping

Task A	Task B
Runnable1	Runnable2
Runnable5	Dcm_MainFunction
Runnable9	Runnable11
	Dem_MainFunction



Your goal should be a configuration that results in a highly runtime optimized and memory consumption optimized code. Here is a little information for you to estimate what the RTE generator does.

Re-entrant and not mapped

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).



Caution: It is up to you that your runnable is really re-entrant. If not, errors can occur that are not easy to debug.

Not re-entrant and not mapped

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 is shown.

Not re-entrant and mapped

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_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 in 400 bytes!

4.4.11 Data Mapping

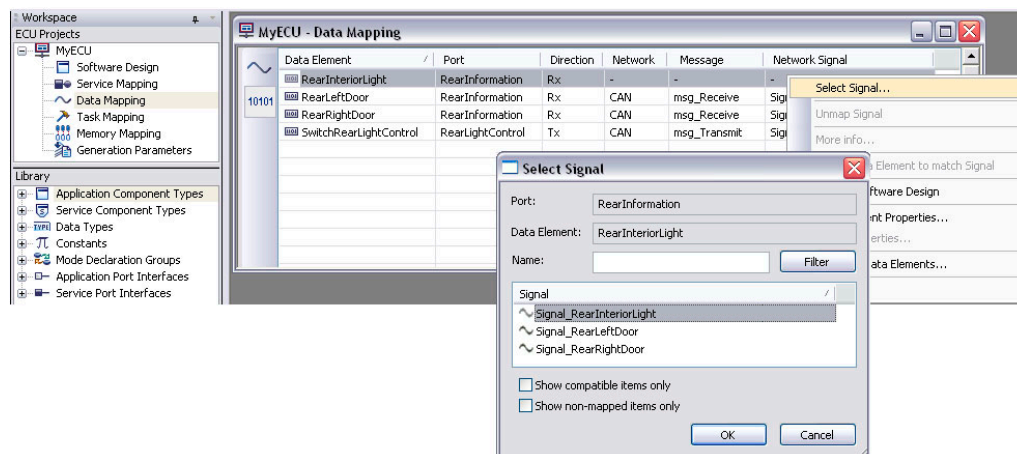
Via the data mapping you connect data elements with network signals. Network signal had been loaded via import of ECU Extract of System Description.

Assign data to signals

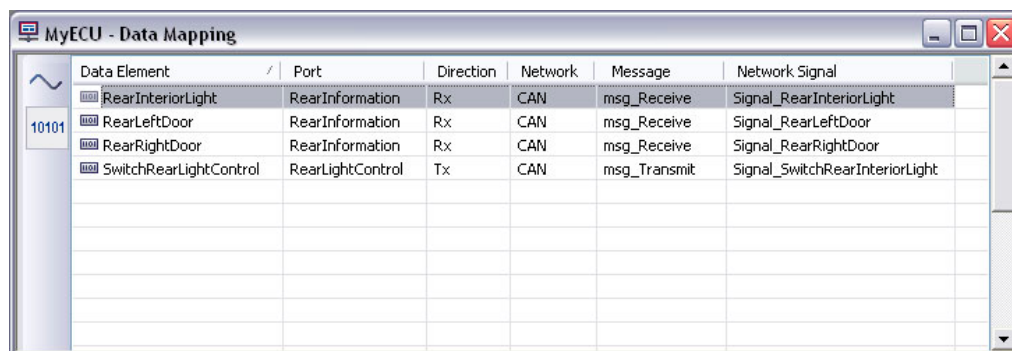
To connect the data elements with real bus signals perform the data mapping manually. Select **Data Mapping** in the ECU Projects view and select e.g. the **Data Element View Mode**. This is a list of data elements. The dash (-) on the right side below **Network**, **Messages** and **Network Signals** shows that there is no assignment between data elements and messages/signals from the ECU Extract of System Description. Do this now.

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

10101 data element view mode



The result of the data mapping should look like this.



Data Element	Port	Direction	Network	Message	Network Signal
RearInteriorLight	RearInformation	Rx	CAN	msg_Receive	Signal_RearInteriorLight
RearLeftDoor	RearInformation	Rx	CAN	msg_Receive	Signal_RearLeftDoor
RearRightDoor	RearInformation	Rx	CAN	msg_Receive	Signal_RearRightDoor
SwitchRearLightControl	RearLightControl	Tx	CAN	msg_Transmit	Signal_SwitchRearInteriorLight



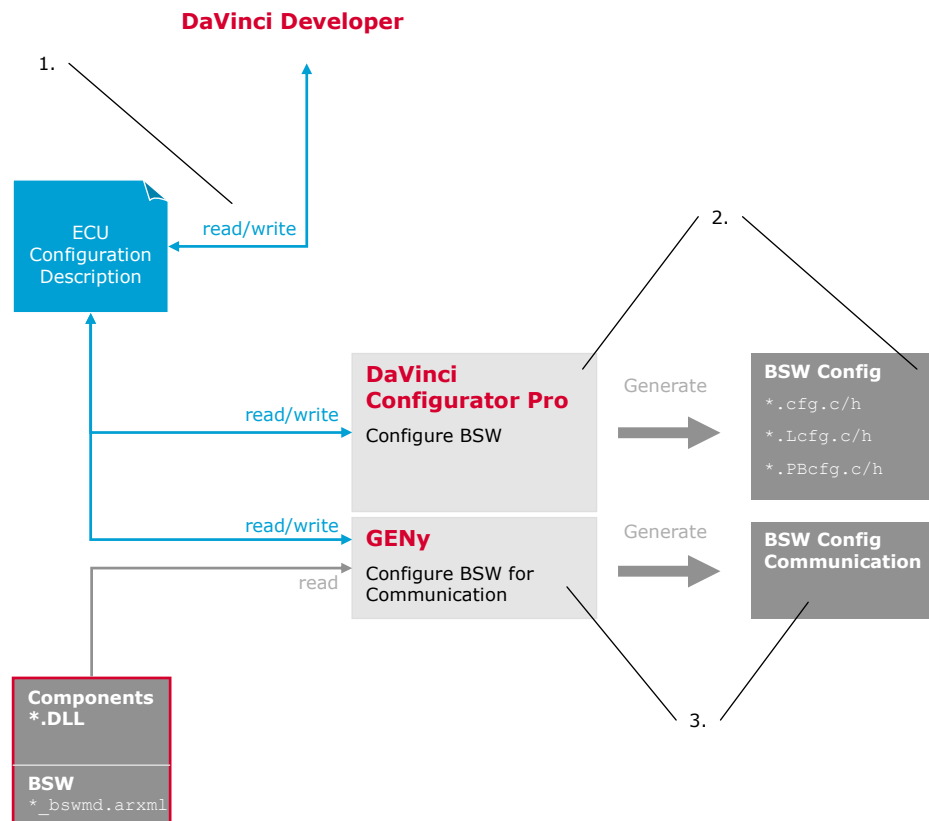
Info: You can let the Developer do this mapping automatically. Just right-click and select **Automap Data Elements**

4.5 BSW Configuration

The basis software modules are either available as library or as source code and must be configured to meet your needs and expectations.

To configure the hardware dependent basis software modules use the **DaVinci Configurator Pro**. Configure the communication modules with the well-known **GENy** and take profit from your existing knowledge about handling this tool.

Detailed illustration of the configuration tools, their source and their generator output



4.5.1 Synchronize ECUC with DaVinci Developer

Before starting to configure the BSW you should save the workspace with DaVinci Developer. DaVinci Developer automatically synchronizes the ECUC, i.e. it updates the ECUC to match the DaVinci workspace. Therefore, it adapts the relevant sections of the ECUC: RTE, Os, Com and NvM.

After that you can open the ECUC e.g. with DaVinci Configurator Pro (see following chapters), configure the BSW and save the ECUC. DaVinci Developer will automatically detect that the ECUC has changed, and perform the automatic synchronization again. This time, the DaVinci workspace is updated to reflect the modifications in the ECUC.

4.5.2 Configure BSW with DaVinci Configurator Pro

Launch DaVinci Configurator Pro from Start|Programme

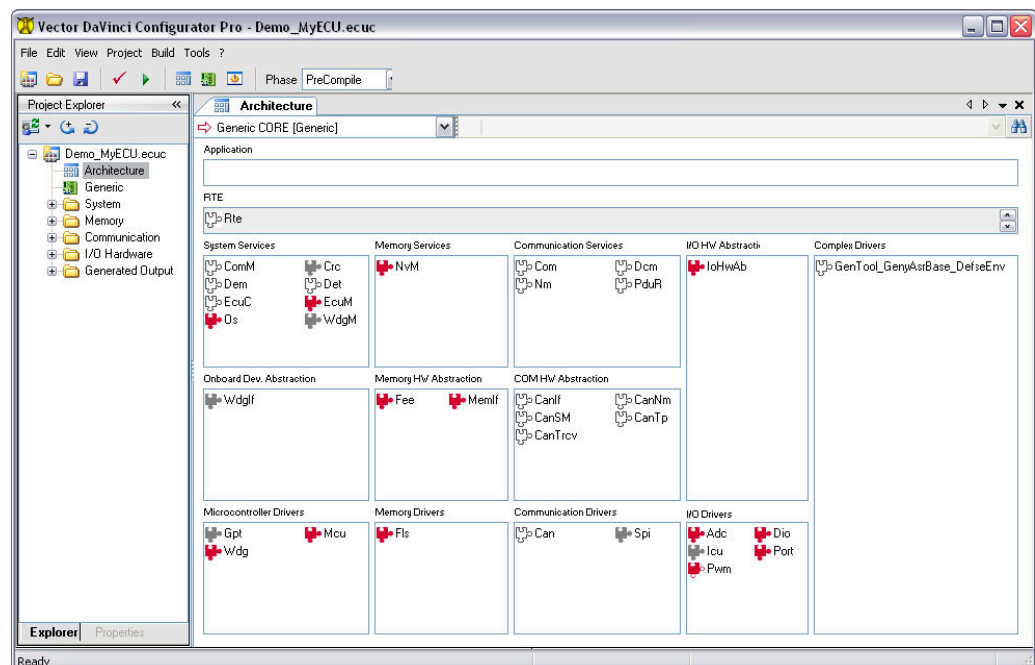
Open the **DaVinci Configurator Pro** to configure the operating system. The ECUC file is read-in by the **DaVinci Configurator Pro**.



Info: There are at least two ways to work with the **DaVinci Configurator Pro**.

The recommended one is to use **DaVinci Configurator Pro** as a stand-alone tool and start it via an appropriate link. The project assistant (see section Project Assistant on page 21) has created an according links (Configure BSW). you find this in link in the start menu and in the **.config** subfolder of the project folder.

You can also use the configurator list of the **Developer** to start **DaVinci Configurator Pro** (see more in the online help).

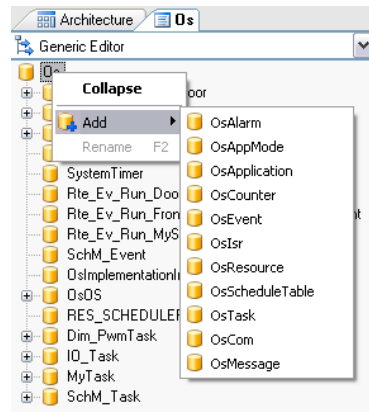


Icon	Meaning
	AUTOSAR module, activated
	AUTOSAR module, inactive Inactive modules are not stored into ECUC file.
	No licensed module This symbol is used when an ECUC file is opened that contains modules which are not licensed in current project environment.
	No AUTOSAR module A module which data definition is not based on an AUTOSAR standard module definition file.

The concept of using the **DaVinci Configurator Pro** for basis software module configuration is simple. Open each necessary module by double-click and fill in necessary configuration settings.

Using the GCE editor (**Generic Configuration Editor**):

- > Fill-in necessary information or values in already existing containers
- > User right-click and **Add** for new containers to the configuration



Info: The SchM_Task is already in the illustration. You can define it already in the **DaVinci Developer** or here as shown in the following (see **The SCHM - Create Task, Event and Alarm** on page 52).

4.5.3 Tipps when working with DaVinci Configurator Pro

Values in decimal and hex

Every value you can enter in the **DaVinci Configurator Pro** now can be given as decimal, hex value and binary (D, H, B).

Default or recommended values can be reset

Using the <Shift>+<F1> keys the values are reset to default or recommended value. Then a info window will open showing the default value, if available.

4.5.4 Default settings for OS

Defaults

The following entries are mandatory and should be already part of the predefined OS configuration.

- > OsImplementationInformation no further settings possible
- > OSDEFAULTAPPMODE no further settings possible
- > OsOs
- > SystemTimer no further settings possible

OsOs – OSEK OS properties

The settings for the OSEK OS configuration could look like below.

OsOS	
OsScalabilityClass	SC1
OsStackMonitoring	True
OsStatus	EXTENDED
OsUseGetServiceId	False
OsUseParameterAccess	False
OsUseResScheduler	True
OsOSTypeHeaderInclude	true
OsOSTickTime	1000
OsOSCompiler	MicrosoftVisualC

Set the standard OS information like scalability class, stack monitoring (if needed), stack size (not for CANoe Emu), etc.

OsOs | OsHooks

Define all OS Hook functions you need for your project.

OsHooks	
OsErrorHook	False
OsPostTaskHook	False
OsPreTaskHook	False
OsProtectionHook	<Empty>
OsShutdownHook	False
OsStartupHook	False

In the demo nothing is selected.

4.5.5 Configuration of Generated Tasks, Events and Alarms

Tasks

In the **DaVinci Developer** you have defined some tasks, e.g the ones from the demo.

- > MyTask
- > IO_Task
- > Dim_PwmTask

Events and Alarms

Derived from the settings of the runnables and their activation there are created OS events and OS alarms. Here are some examples listed:

- > Rte_**AI**_TE_Door_Door (**Alarm**)
- > Rte_**Ev**_Run_Door_Door (**Event**)
- > Rte_**Ev**_Run_FrontInteriorLight_SwitchFrontLight (**Event**)



Info: The names are generated by the **Developer**

Task configuration first

The tasks have to be configured now.

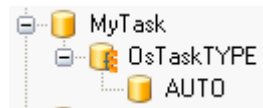


Info: The configuration of the tasks can look different from it is done for this example.

MyTask

MyTask configuration

Add OsTaskTYPE and set it to AUTO.



MyTask settings

Then click on MyTask and fill-in configuration information in the view on the right side of the **DaVinci Configurator Pro**.

MyTask	
OsTaskActivation	1
OsTaskPriority	1
OsTaskSchedule	NON
OsTaskStackSize	<Empty>
OsTaskAccessingApplication	<Empty>
OsTaskEventRef	/Os/Rte_Ev_Run_MySWC_MySCW_Code
OsTaskResourceRef	<Empty>
OsTaskMESSAGE	<Empty>

Your settings can vary.

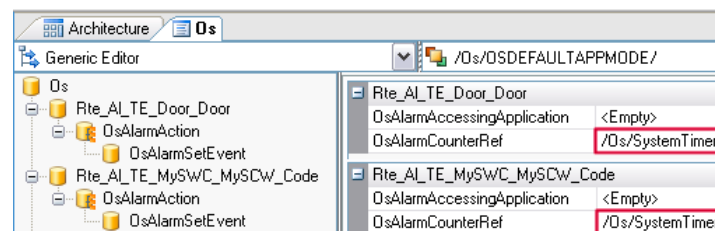
Make sure that you enter a stack size (**OsTaskStackSize is only available if real hardware is used**) and the **OsTaskActivation** is set.

No further settings for events necessary

For the generated events there are no further settings necessary.

Settings for generated alarms

Settings for the alarms (select SystemTimer) for e.g. **Rte_AI_TE_Door_Door** and **Rte_AI_TE_MySWC_MySCW_Code**



4.5.6 The SCHM - Create Task, Event and Alarm

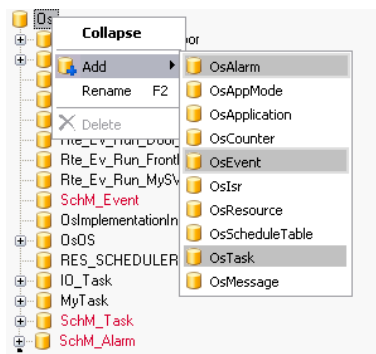
SchM_Task with Events and Alarms,

The **SchM_Task**, the task for the Schedule Manager (SCHM), has to be created manually. As well as its events and alarms. You can also enter the task in the **DaVinci Developer**.

The initial name always is **OsAlarm**, **OsEvent** and **OsTask**. Create it and then rename it.

Right-click the OS and **Add**

- > OsAlarm
- > OsEvent
- > OsTask



Rename

To make a **SchM_Task**, **SchM_Event** and **SchM_Alarm** out of the default task, alarm and event, you have to rename the added entries (right-click and **Rename**).



Caution: Make sure you write the **SchM_Task**, **SchM_Event** and the **SchM_Alarm** like shown here. The delivered SchM BSW expects exactly that naming.

Autostart for SchM_Task

The **SchM_Task** must be a task that starts automatically.

Add **OsTaskAutostart** to the SchM_Task (right-click & Add) and set the **OsTaskAppModeRef** to **/Os/OSDEFAULTAPPMODE**.

SchM_Task settings

These are the settings of the **SchM_Task**. Depending on your system, the setting will vary.

SchM_Task	
OsTaskActivation	1
OsTaskPriority	5
OsTaskSchedule	NON
OsTaskComputationTime	<Empty>
OsTaskPeriod	<Empty>
OsTaskDeadline	<Empty>
OsTaskAccessingApplication	<Empty>
OsTaskEventRef	/Os/SchM_Event
OsTaskResourceRef	<Empty>
OsTaskMESSAGE	<Empty>

Reference of SchM_Event

Remember to set the **SchM_Event** as **OsTaskEventRef**.

SchM_Event settings

No additional settings are necessary.

SchM_Alarm settings

An Alarm needs a counter or a time base. Add the **SystemTimer** as

OsAlarmCounterRef reference.

SchM_Alarm	
OsAlarmAccessingApplication	<Empty>
OsAlarmCounterRef	/Os/SystemTimer

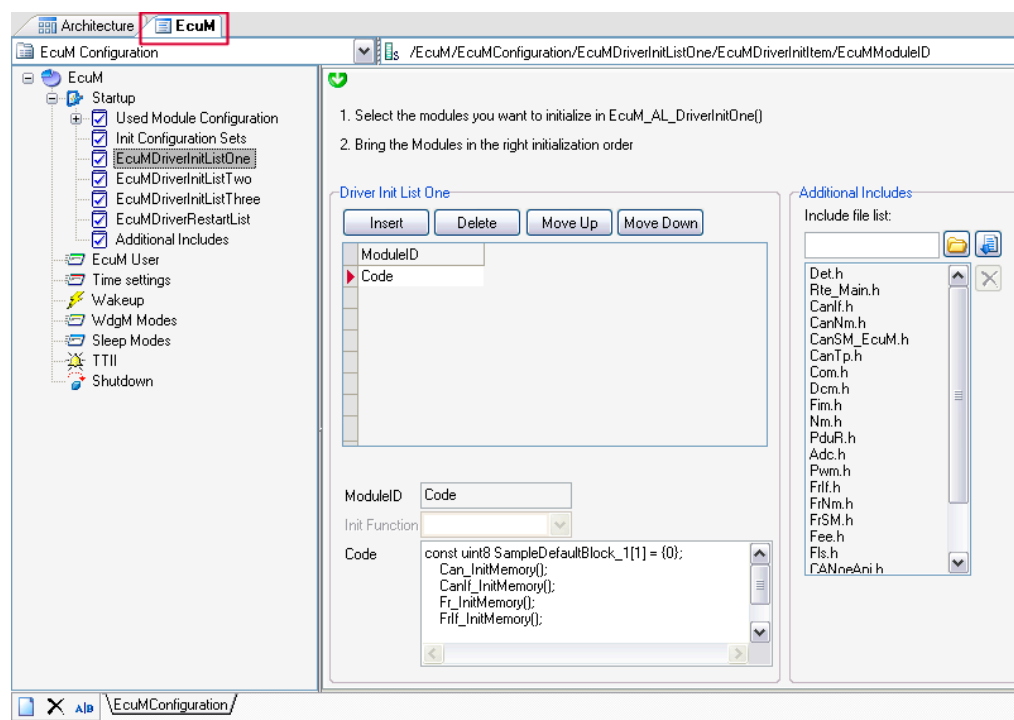
The **SchM_Alarm** triggers the **SchM_Event**

An occurring Alarm of the **SchM_Task** triggers the **SchM_Event**. This is configured in the Container **OsAlarmSetEvent**. Select the **SchM_Event** that belongs to the **SchM_Task**.

OsAlarmSetEvent	
OsAlarmSetEventRef	/Os/SchM_Event
OsAlarmSetEventTaskRef	/Os/SchM_Task

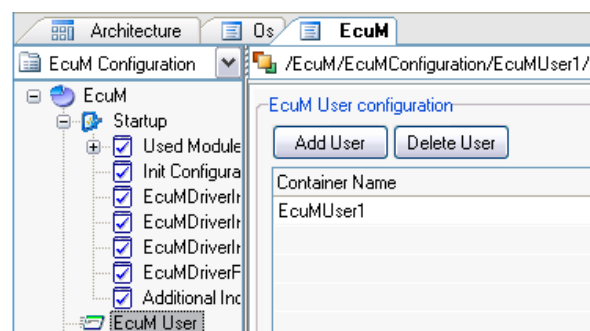
4.5.7 Settings for the ECUM

ECUM Configuration in DaVinci Configurator Pro



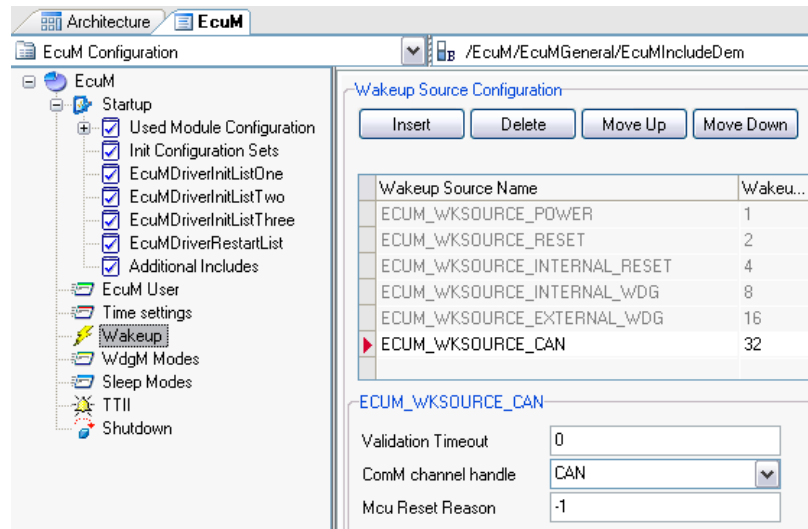
Info: Make sure only to use those module init functions in the driver init lists that are used.

- > One ECUM User is a must and is predefined. You can change its name if necessary.



- > Define the Wakeup via Wakeup Source Name and Wakeup Source Id (**ID = 2ⁿ**). The number depends on the already existing wake-up sources.

Wake-up source for CAN wake-up.



Info: This **Wakeup Source ID** should be the same as you will set in **GENy** later (see section Settings for CAN – Can_CanoeemuCanoe on page 65). Otherwise the ECUM will not recognize the wake-up reason as CAN wake-up and the ECU will not be woken up by a CAN event.



Cross reference: You find detailed information about wake-up and sleep with AUTOSAR in the document **TechnicalReference_Wake-up_and_Sleep_with_AUTOSAR**.

List one, two, three,...

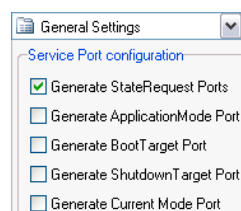
The ECUM controls the calls of the init functions of the BSW modules and uses the EcuMDriverInitLists. Fill the lists with the necessary initialization calls like

- > CanIf_InitMemory();
- > Can_Init(&CanConfig);
- > CanIf_Init(&CanIf_Config);
- > ComM_Init();
- > ...

4.5.8 Service Ports or not?

Creation of Service Ports must be set

Open the **General Settings** tab from the pull-down menu at the top of the tabs and select your necessary service ports via **Service Port configuration**.



4.5.9 Generate Configuration Files and EcuM_swc.arxml

Start generation process by clicking the green play button...

After you have finished the configuration start the **Generation Process** via the green triangle. There is also created a file named **EcuM_swc.arxml**. See later in the chapter dealing with service components what to do with this XML file.

4.5.10 Usage of I/O - DIO as example

Digital Input / Output

Access environment variables of CANoe

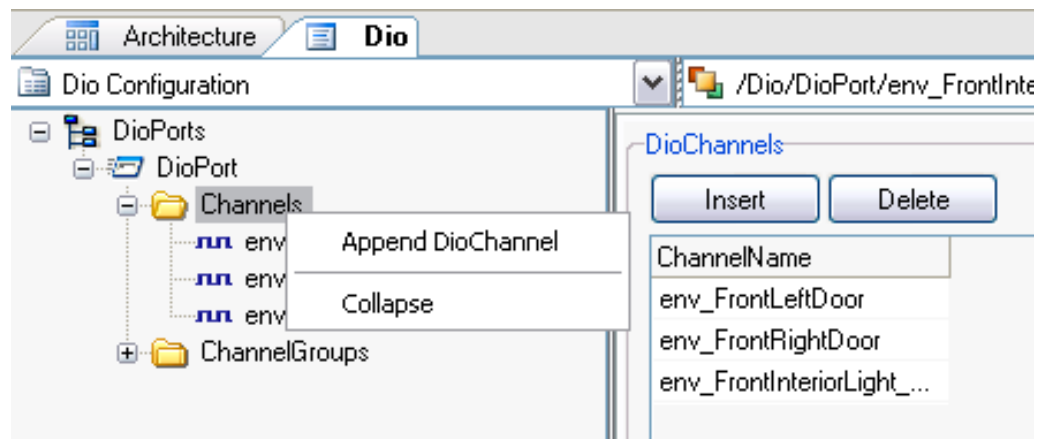
In this example we access environment variables of the CANoe panel using I/O. This will explain the basic concept of using I/O.

4.5.11 Settings for DIO

Select the **DIO** and define **Channels**, e.g. channels for the two doors and the interior light.

- > env_FrontLeftDoor
- > env_FrontRightDoor
- > env_FrontInteriorLight_DEFECT

Append channels and name them.



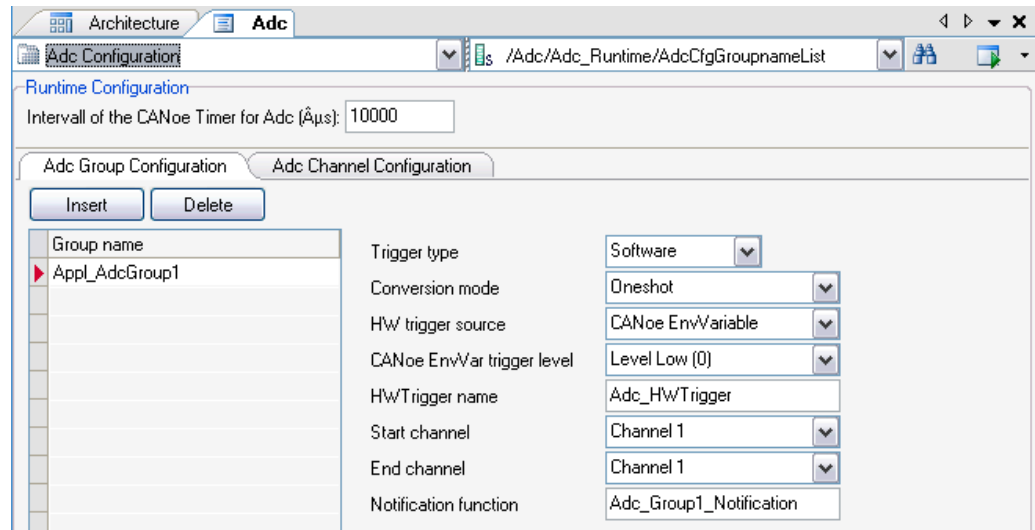
Info: These names are used later as names of the **environment variables** of the CANoe panels.

4.5.12 Configure ADC

Activate the ADC in the **Architecture** view of the **DaVinci Configurator Pro**

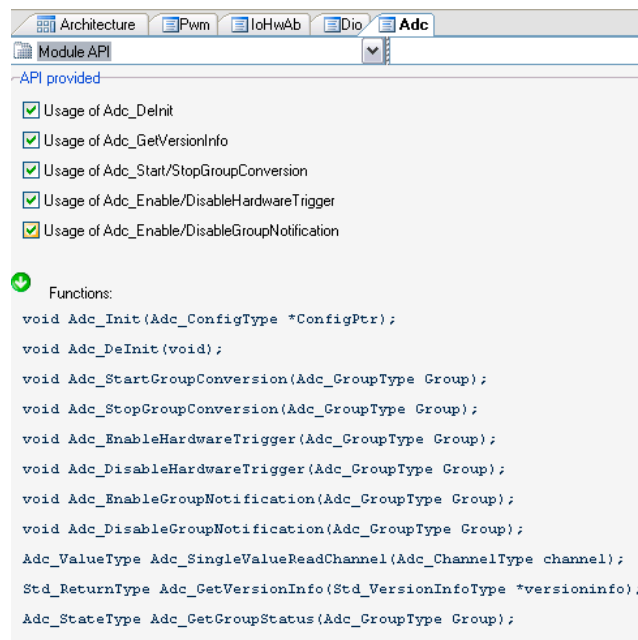


and configure an ADC channel.



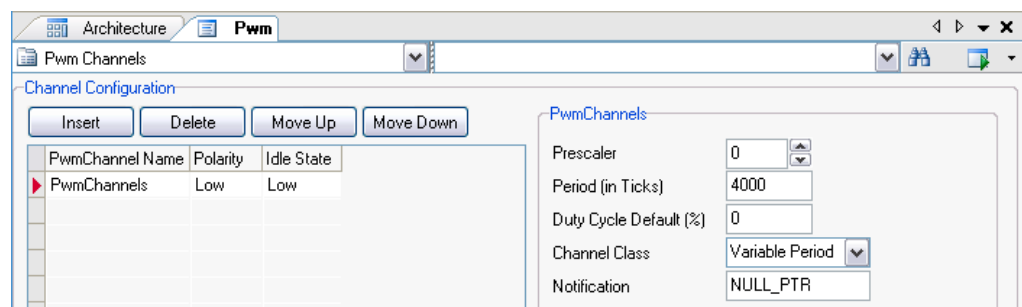
Available API

The available API for realizing the ADC could look like shown.



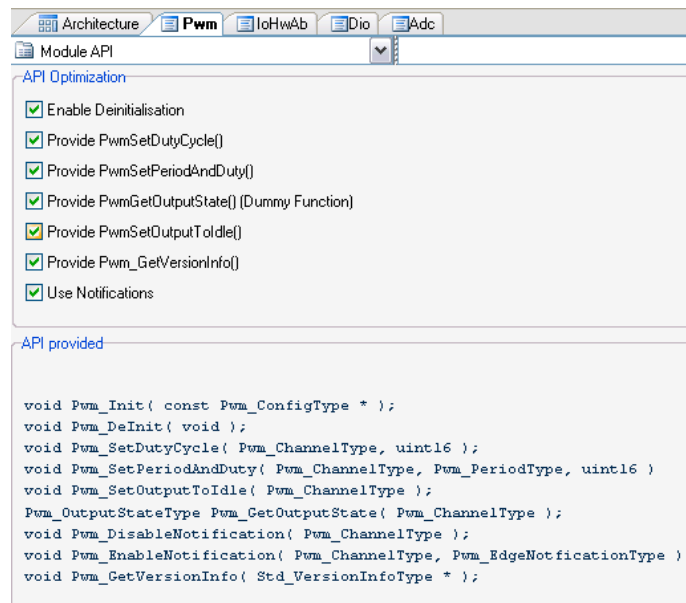
4.5.13 Configure PWM

Activate the PWM in the **Architecture** view of the **Configurator Pro** and configure a PWM channel.



Available API

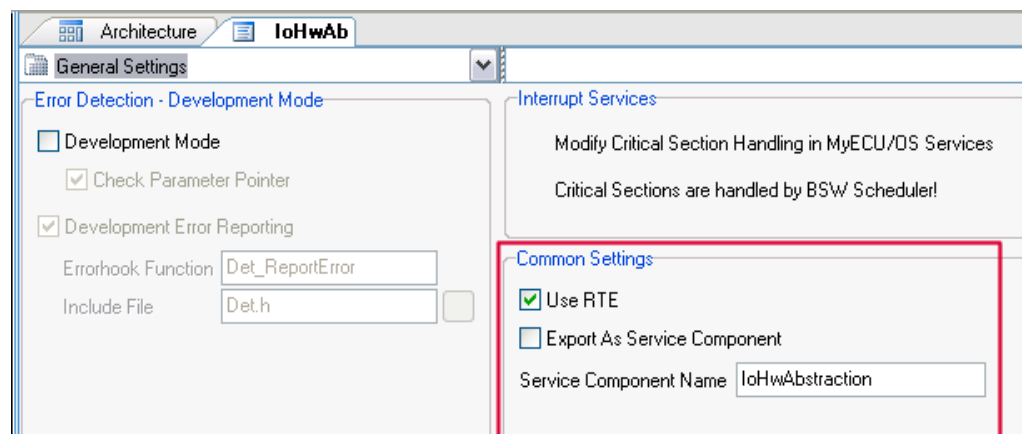
The available API for realizing the PWM could look like shown.



4.5.14 Settings for IOHWAB

Application software component or service software component?

The IOHWAB can be used as application software component or as service software component. You can configure this in the **DaVinci Configurator Pro** in the **General Settings** of the **IOHWAB**.



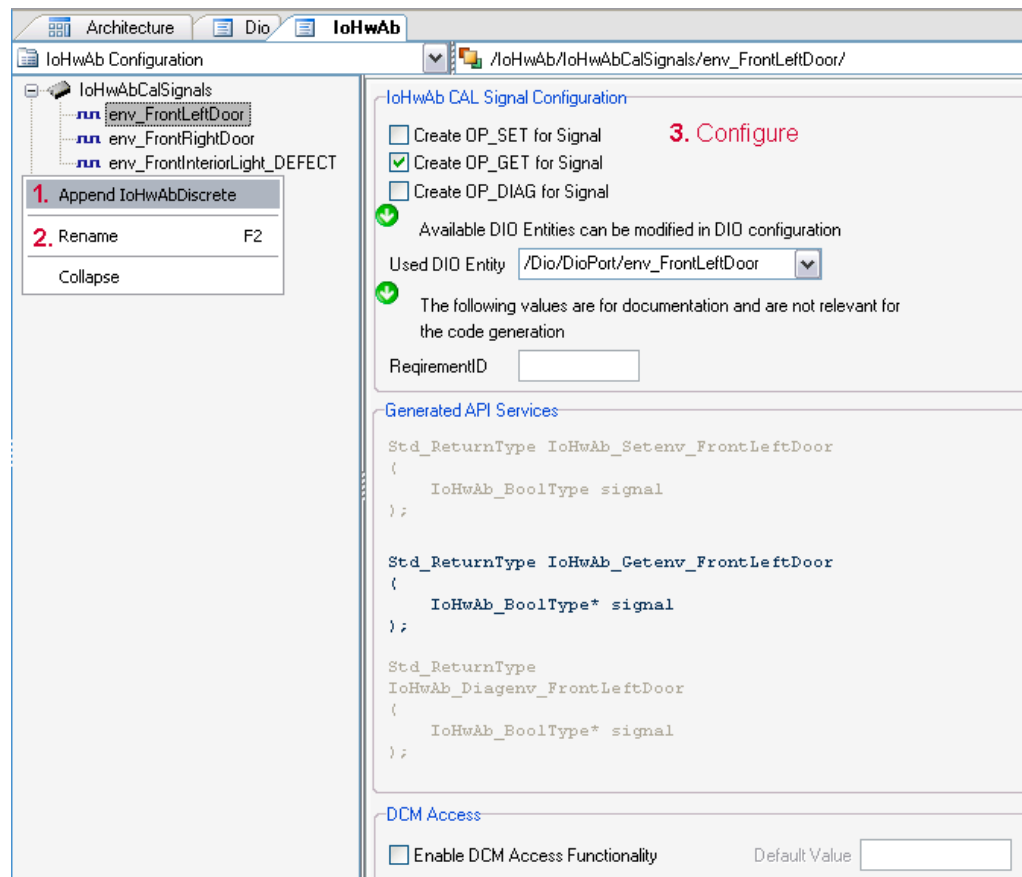
The demo uses it as application software component. Its usage then differs slightly from the usage of service components. For more see **IOHWAB is Application Component** – an Exception on page 77.

IOHWAB for DIO
GET or SET? Or
Both?

Now select the configuration view of the **IOHWAB** (I/O Hardware Abstraction).

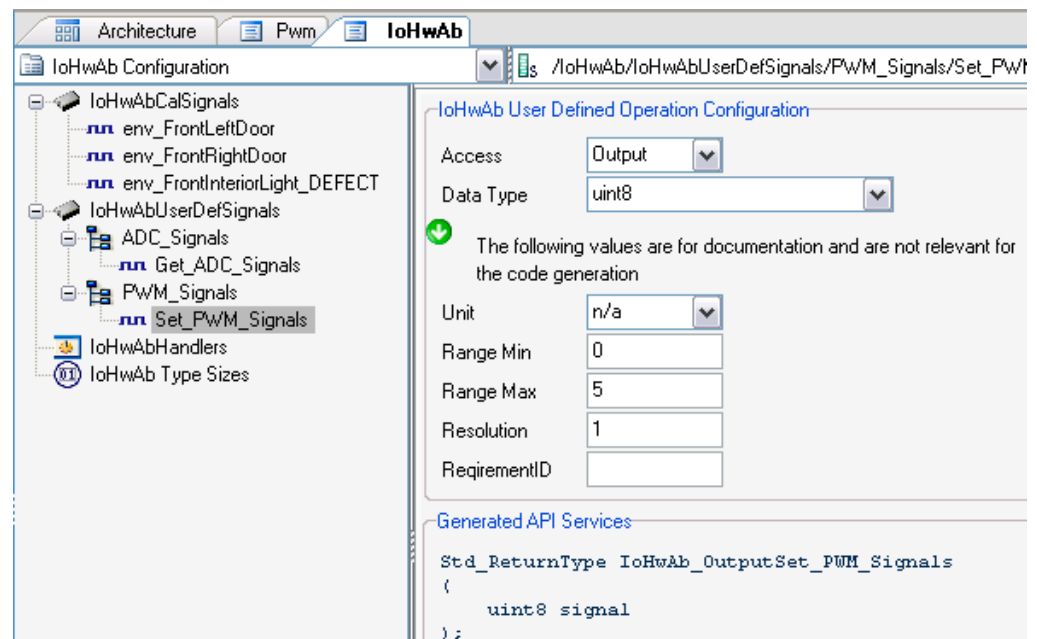
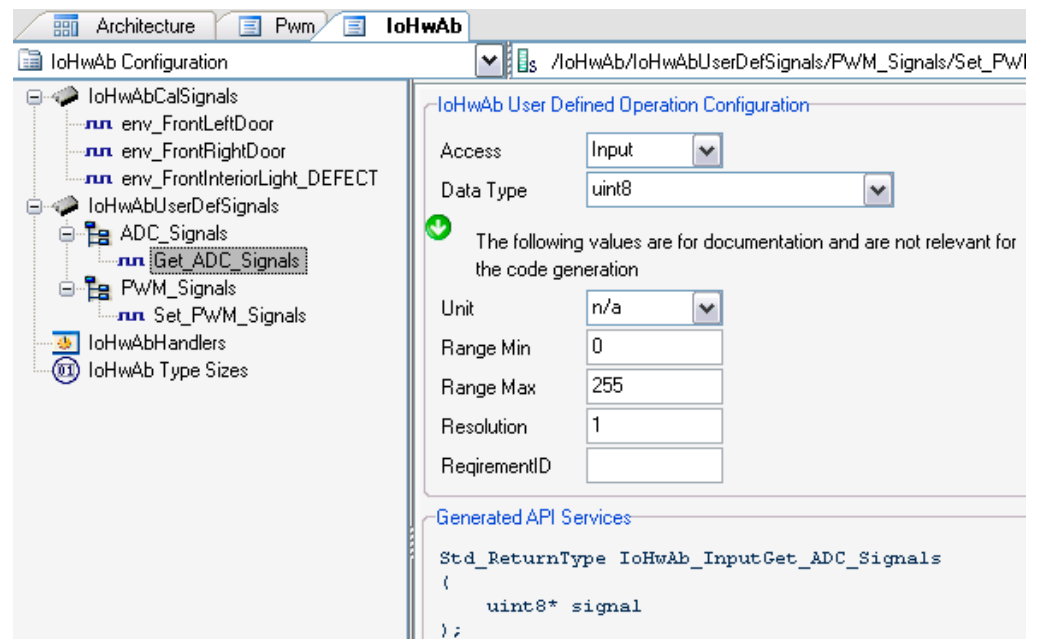
1. Create appropriate new CalSignals via right click and **Append IoHwAbDiscrete**.
2. Select the appropriate DioEntity.

To rename the **IoHwAbDiscrete** select the one you want to rename and press **[F2]** or use right-click and select **Rename**.



IOHWAB for ADC and PWM

Add signals for **ADC** and **PWM** below **IoHwAbUserDefSignals**. Configure the signals like shown in the following screenshots.



Cross reference: Find more about usage of IOHWAB software component in the chapter dealing with service components (see section Service Components on page 71).

4.5.15 Generate Configuration for DIO and IOHWAB

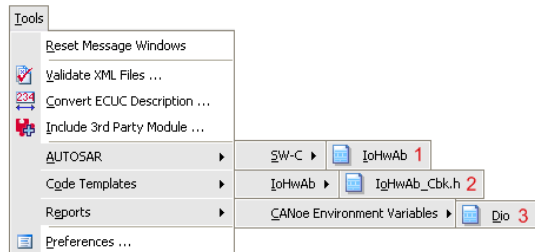


Save your setting via clicking the button **Start Generation Process**.

You have to generate more...

Additionally you have to generate the following three files.

1. IoHwAb **arxml** file
2. **Callback functions** for the IoHwAb
3. **DBC** file that contains the environment variables for the CANoe panel.

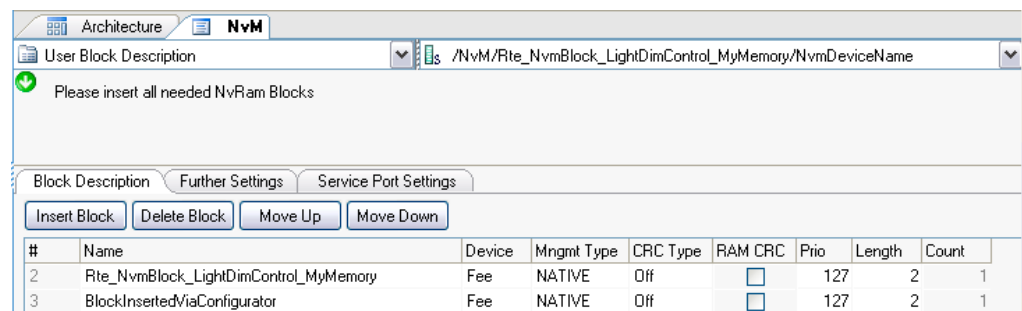


Hint:

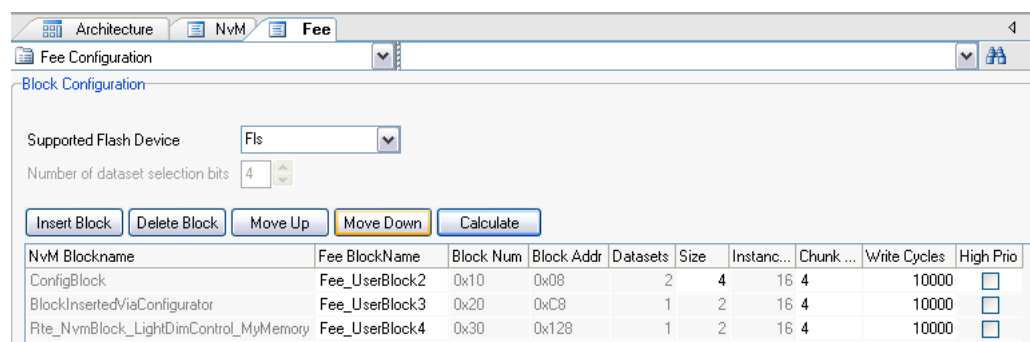
Make sure to generate the files (1-3) to the appropriate folder.

4.5.16 Memory via Service Ports

Select NVM



Select e.g. Fee



Info: Just have a look at the demo example to get all the settings for NVM and FEE.

Generate service component for NVM

Generate the service component for the NVM via:

Tools|AUTOSAR|SW-C|NVM.

The file `NvM_sw.c.arxml` is generated.

**Already known
actions**

You already know how to handle service components. Import them into the **Developer** and connect you SWC via service ports with the NVM.



Info: Use the automatic method to save time.

4.5.17 NVM and Fee Configuration

Read more in the
Technical
References

The configuration of Fee and NVM need much knowledge about memory and its functions. It would be too much to be explained in this sort of manual.



Cross reference: Please refer to the Technical Reference for Fee and NVM to get detailed information.

4.5.18 Configure Watchdog

The watchdog is designed for functional supervision of an ECU and has to be triggered in a predefined period of time. Otherwise the watchdog will perform a reset.

WDG, WDGIF and
WDGM

The watchdog functionality comprises three BSW modules

- > WDG
- > WDGIF
- > WDGM

4.5.19 Configure WDG

Watchdog Variants

Set the Default Mode to Fast Mode, Slow Mode or Off and define the appropriate timeout values.

Architecture Wdg

Watchdog Variants /Wdg/Wdg_Runtime/WdgDefaultmode

Mode

Default Mode Fast Mode

Settings

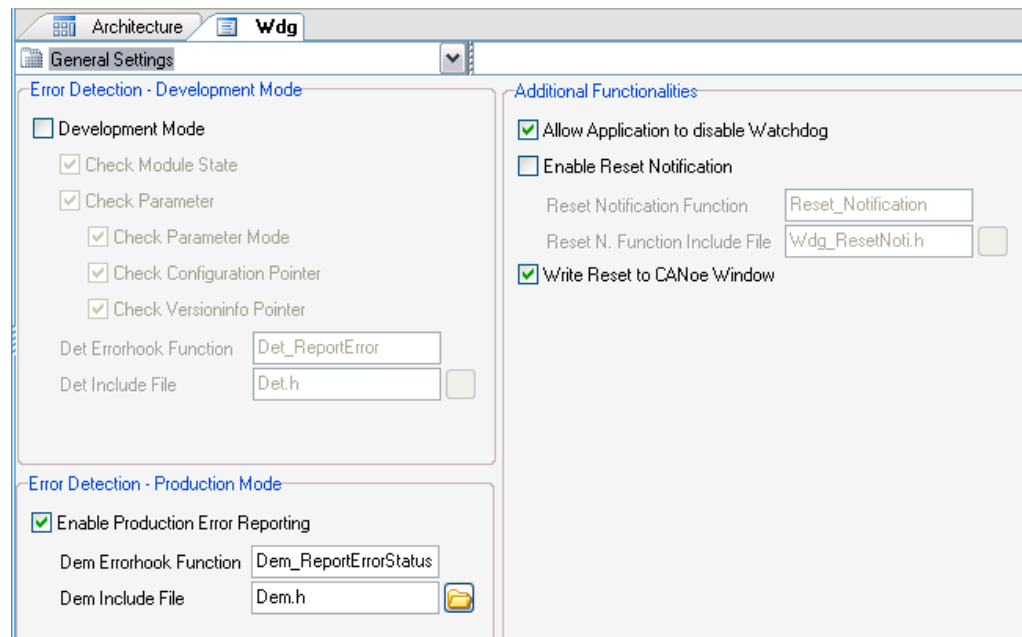
Fastmode Timeout (µs): 15000

Slowmode Timeout (µs): 20000

Off Mode: No Configuration Options for Off Mode

General Settings

Decide whether to use DET or not and whether to do DEM settings in error case. Decide about notifications and what application is able to do.



4.5.20 Configure WDGIF

Chose your device In the WDGIF select you **Device Name** and decide whether to use DET or not.

4.5.21 Configure WDM

Supervised entities Define as much supervised entities as you need for the application. There are more settings for the WDM. Read more in the corresponding technical reference.



Cross reference: The WDM is a service component. For more information about its usage see section [Service Components](#) on page 69.

4.5.22 Switch from Configurator Pro to GENy

After saving the ECUC with the DaVinci Configurator Pro you can open GENy to configure the BSW modules for communication. Use the generated link from the project assistant.

4.5.23 Configure BSW for Communication with GENy

GENy is the well-know and comfortable tool to configure communication modules and diagnostics.

Open GENy via DaVinci Developer

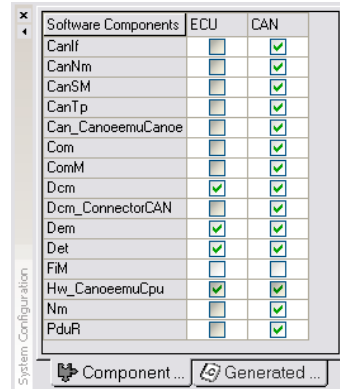
The following information is necessary, regardless using **Developer** or just a *.bat file.

- > Path to **GENy.exe**
- > Path to component folder of your BSW delivery (DLLs, BSWMD)
- > Path to project file (ECUC)

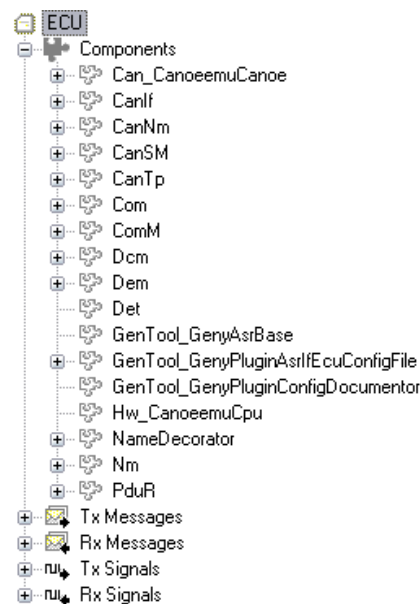
Open **GENy**.

Component selection All basis software modules that are defined inside the ECUC file are selected in the component selection. Perhaps you have to select further modules you need for your project.

The result could look like shown in the following illustration.



Component list in the tree view of GENy Every selected component is immediately entered in the component list of the navigation view of **GENy**.



Set generation output path **GENy** generates configuration files for the components that have to be compiled and linked together with your project and the cores of the BSWs, the basis software components. Therefore it is very important to set the **Generation Paths...** correctly.

Via **Generation|GenerationPaths...** you open the **Generation Directories** window where you can set the paths for the **GENy** output files. More about how to deal with this window is written in the Online Help of **GENy**.

Necessary settings of the BSW For a basically running AUTOSAR system some default settings of the BSW have to be changed like shown in the following.

4.5.24 BSWs for Communication and Diagnostics

COM, CANTP,
CANNM, CANIF,
CANSN, NM,
CANTRCV_Generic

The first settings in BSW are dealing with communication. This is more than just the BSW COM. In detail this affects the listed BSWs.

- > CAN_CanoeemuCanoe
- > COM (manual settings necessary)
- > CANIF
- > CANTP
- > CANNM
- > NM
- > CANSN
- > CANTRCV_Generic1 (not for CANoe Emu)
- > DCM
- > DEM
- > COMM

4.5.25 Settings for CAN – Can_CanoeemuCanoe

Wake-up source to
identify the reason
for a wake-up.

In case of a wake-up the wake-up source is necessary. For this reason, the CAN wake-up source must be set to the same value as it will be set in the **Configurator Pro** (see there). Open the channel path of the CAN Driver (in this case it is CAN_CanoeemuCANoe as **CANoe** is the CAN Driver).

Set the **Wakeup Source ID** to the same value as set before for the ECUM (e.g. **32**).



Cross reference: See more about wake-up and sleep with AUTOSAR concept in the **TechnicalReference_Wake-up_and_Sleep_with_AUTOSAR**.

4.5.26 Settings for COM – PDUs

Names of IPDU
Groups

Just check the **IPdu Groups** and whether receive and transmit IPDUs are assigned correctly.

4.5.27 Settings for COM – Indication Functions when working with DBC file

COM

There is nothing to do here.

4.5.28 Settings for CANTP

Transport Protocol

There is nothing to do here.

4.5.29 Settings for CANNM

Network
Management

There is nothing to do here.

4.5.30 Setting for CANIF

CAN Interface

Probably the ECU must be able to wake up. Check these settings.

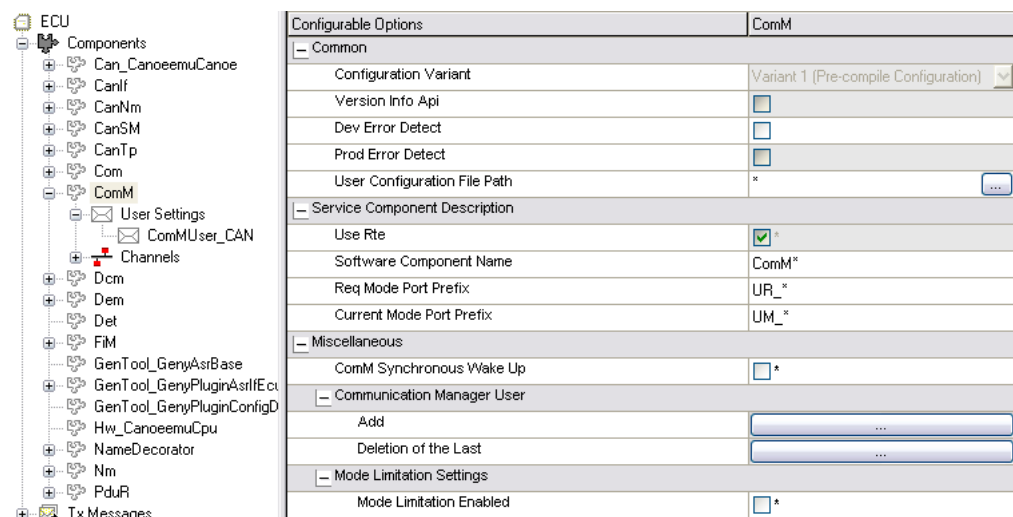
Callback Functions	
Wakeup Notification Function	EcuM_SetWakeupEvent*
Wakeup Validation Notification Function	
BusOff Notification Function	CanSM_ControllerBusOff*

4.5.31 Settings for the COMM

Communication Manager

To make the Communication Manager basically work you must define at least one User per channel. This one user is defined per default. You can change its name.

To define further users just click on the Add [...] button below **Communication Manager User** and **GENy** creates a new user that is displayed below **User Settings**.



The screenshot shows the ECU Components tree on the left, with 'ComM' selected under 'User Settings'. The 'Configurable Options' dialog for 'ComM' is open on the right. The 'Common' tab is active, showing settings like 'Configuration Variant' (Variant 1), 'Version Info Api', 'Dev Error Detect', 'Prod Error Detect', and 'User Configuration File Path'. The 'Service Component Description' tab is also visible, showing 'Use Rte' (checked), 'Software Component Name' (ComM*), 'Req Mode Port Prefix' (UR_*), and 'Current Mode Port Prefix' (UM_*). The 'Miscellaneous' tab shows 'ComM Synchronous Wake Up' (unchecked). The 'Communication Manager User' tab is at the bottom, showing 'Add' and 'Deletion of the Last' buttons.



Info: The name of the User is predefined when you **[Add]** a new one, but can be changed.

Now select the new user (UR000) and assign it to the corresponding **Communication Channel**.

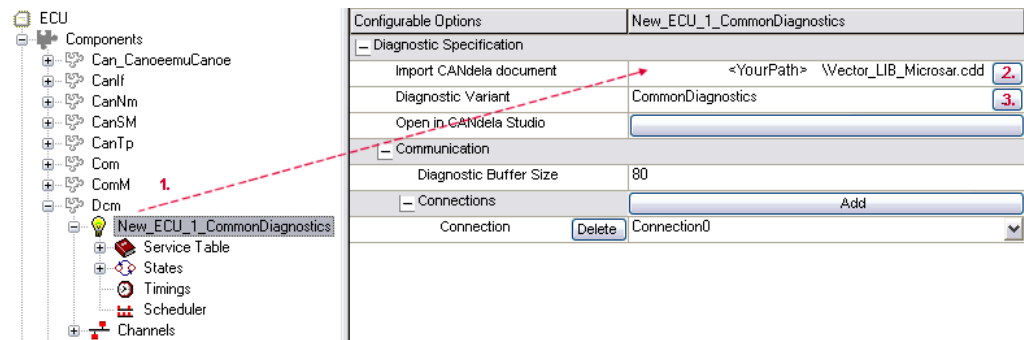
Configurable Options	
ComMUser_CAN	
Communication Channel: CAN (ComMUserChannel)	
	<input checked="" type="checkbox"/>

4.5.32 Settings for DCM

Import CDD file
containing diagnostic
information

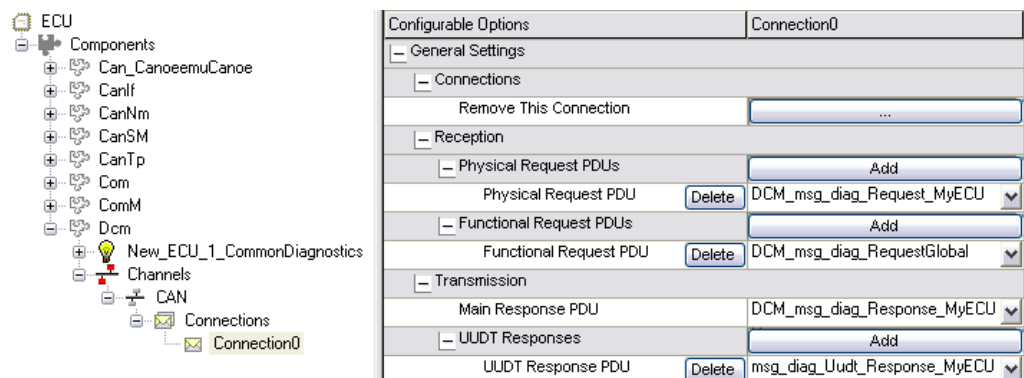
To configure the DCM you have to import the CDD file.

1. Select the entry below Dcm
2. **Import Candela Document [...]** by browsing for the CDD file
3. Select a suitable **Diagnostic Variant** from the pull down menu and confirm with **[OK]**. The pull down menu is only available if the CDD file contains more than one **Diagnostic Variant**.



Diagnostic
Communication
Manager

The SDUs that are automatically set for the Transport Protocol are also automatically assigned to the diagnostic connection as shown in the screenshot.



Now the tree view for the Diag_AsrDcm is extended by the entries for **Service Table**, **States** and **Timings** containing the default settings for the diagnostic.

4.5.33 Settings for DEM

Events

The **DEM** needs at least one event that can be accessed by the application, a so-called external event.

Here the external events are DTC_0x000002 and DTC_0x000003.

4.5.34 Generate Configuration Files

Generate System



After configuration work is finished, you have to generate the configuration files. Click on the icon (on the left side) and **GENy** starts generating all necessary files to the folder you have defined before.



Info: Have a close look at the message view to get informed about configuration errors. If errors occur, follow the error text and try to do the settings correctly. Then start the generation process again.



Info: It is important to generate at this point in time as **GENy** generates the **Dcm_swc.arxml** file, the **Dem_swc.arxml** and the **ComM_swc.arxml** that have to be imported into the **DaVinci Developer**.

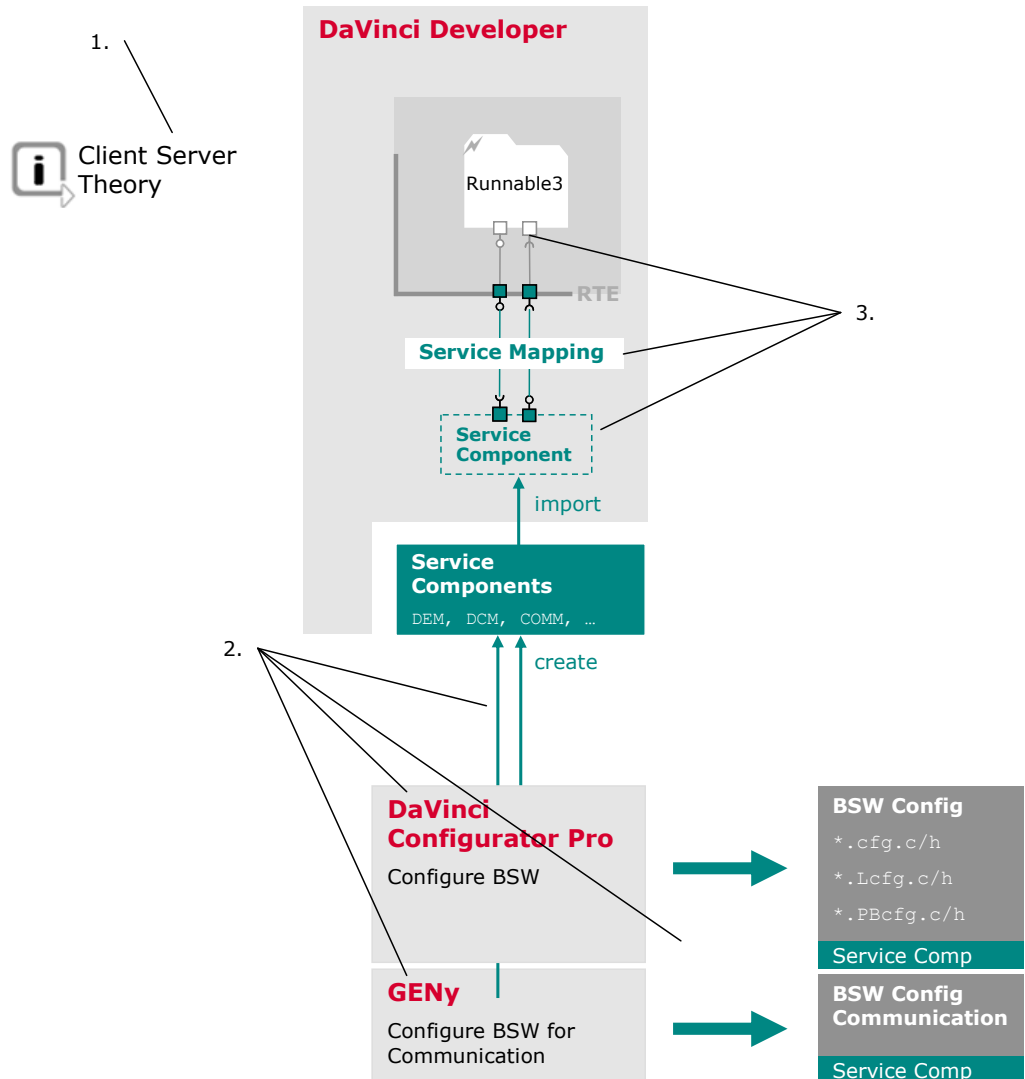
4.6 Service Components

Service Components have to be treated in a different way. How they work is shown theoretically in this chapter.



Info: If you know the theory you can go on with the usage directly (see section Service Components on page 71).

Detailed illustration
of service
components



Content

1. Client Server Theory
(see section [Client Server Theory](#) on page 70)
2. Configure and create Service Components
(see section [Service Components](#) on page 71)
3. Import and use Service Components – Service Mapping
(see section [Import a Service Component](#) on page 72)

4.6.1 Client Server Theory

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

Client server concept basically knowledge necessary to work with service components successfully

The client uses a service of the service. The server itself provides the operation.

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.

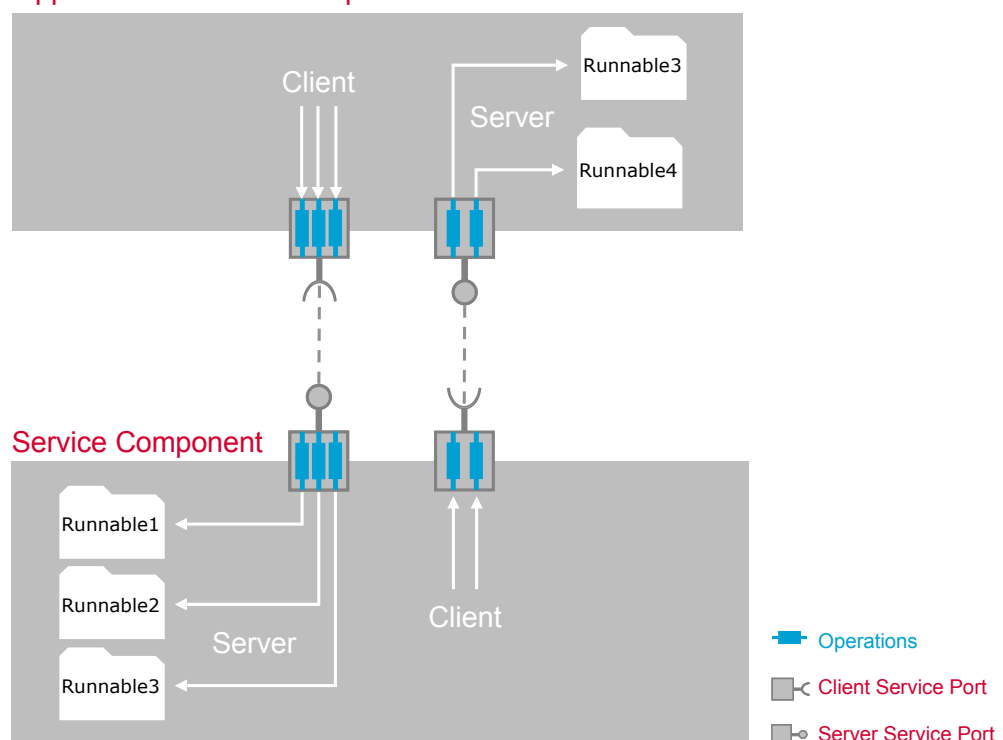
> **Service port of the service component is a server port**

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 with the request of the service.

Application Software Component



The server has to provide the runnables (1,2,3 of service component, 3 and 4 of application software component) that contain the code. Within one service port there could be n operations. Every single operation has to be assigned to a runnable.

4.6.2 Service Components

Service ports and more...

Handling **service components**, **service ports** and **operations** is similar to handling **application software components**, **application ports** and **data elements** – but there is no graphical pendant.



Caution: Service ports and the connections between the ports are **not shown graphically!** This would be too much connections to be displayed.

BSW represented as service components

There are BSW modules that are provided as service components (type and prototype as already known from application software components). These BSW modules are:

- > FIM
- > IOHWAB (can also be application component as used in the demo))
- > ECUM
- > DCM
- > DEM
- > COMM
- > WDGM

4.6.3 Configure and Create Service Components

Configured in GENy or DaVinci Configurator Pro...

You configure service components in the appropriate tools like **DaVinci Configurator Pro** or **GENy** as you configure other BSW modules (see before).

The difference occurs when generating the code. For service components the tools additionally generate an **<ModuleName>_swc.ARXML** file.

...and used within Developer

To use the service component you have to import **<ModuleName>_swc.ARXML** them into the **DaVinci Developer**. Open the **Developer** if it is not already open.

FIM, DEM, DCM and COMM with GENy

- > FIM
- > DEM
- > DCM
- > COMM

You configure the service components as quite normal BSW modules. With the generation process **GENy** automatically creates the ARXML files for those BSW modules, which are service components.

IOHWAB, ECUM, WDGM with DaVinci Configurator Pro

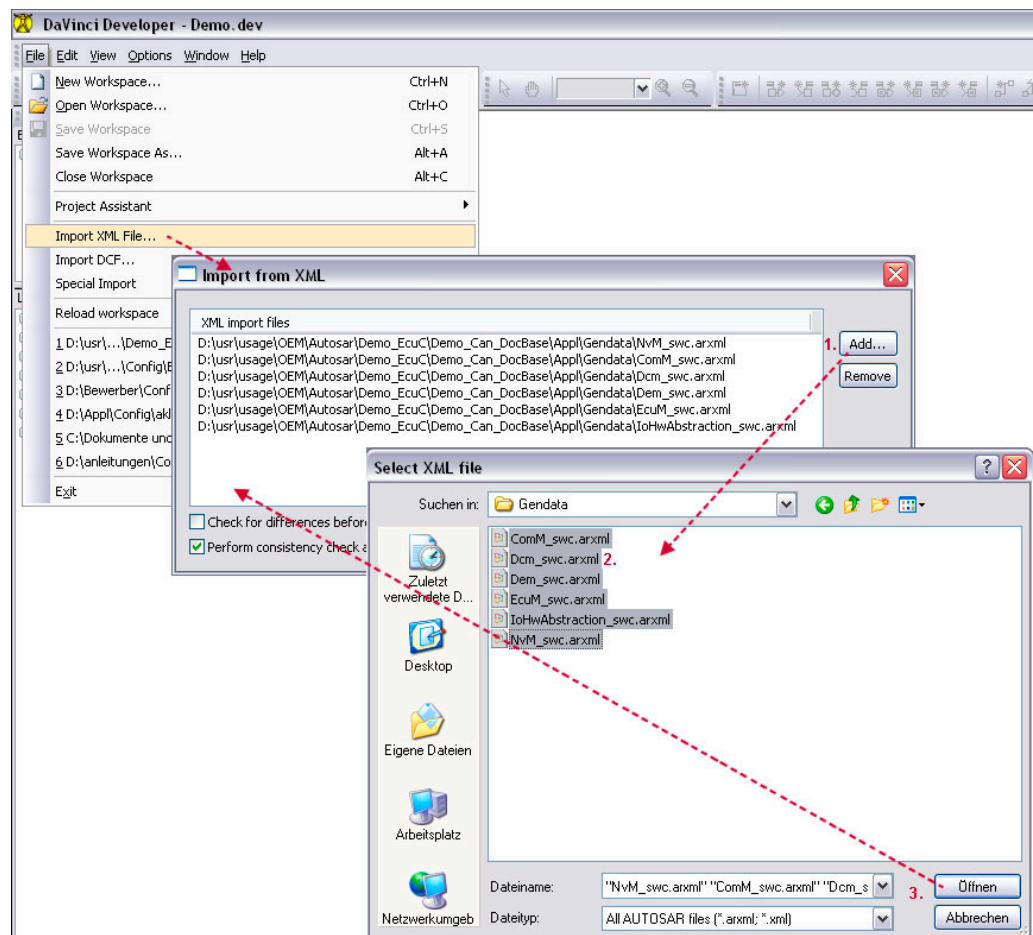
- > IOHWAB
- > ECUM
- > WDGM

Using the **DaVinci Configurator Pro** you have to generate the service component ARXML file manually. See more in the chapters above, dealing with the listed BSW modules.

4.6.4 Import a Service Component into DaVinci Developer

Import XML File...

Import the XML file of the service component (into **DaVinci Developer**) via **File|Import XML File....** Click **[Add...]**, browse to the folder where you have exported your service components at (Gendata) and select the service components ARXML files. Confirm the window and components will be imported.



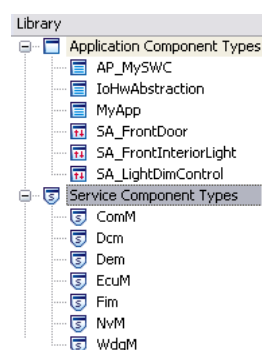
Result of import

The component type can be seen in the **library** and can be used.

4.6.5 What is within a Service Component

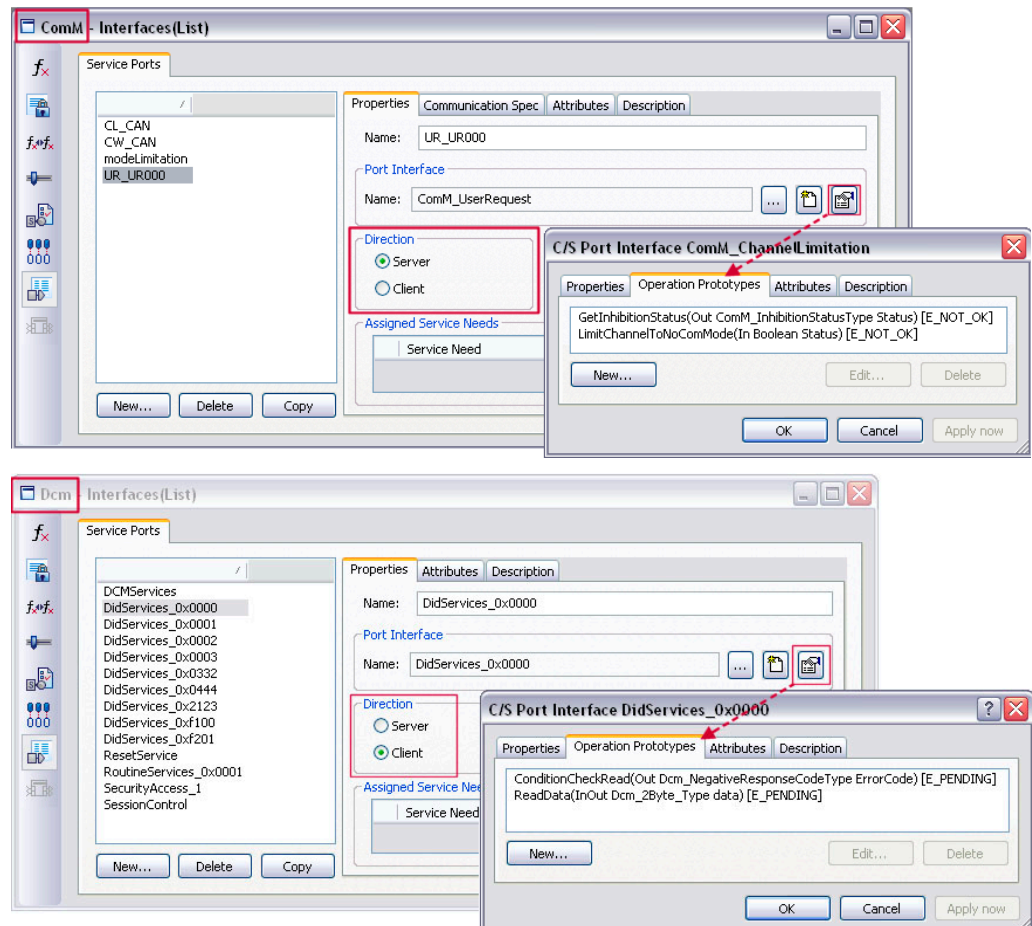
Open a service component type

Double-click a service component in the library and look at the **Service Ports** tab.



To see the operations click the **Properties** icon and select the tab **Operation Prototypes**. In these examples the service port UR_UR000 is a server and the DIDService_0x0000 is a client. In the first case the two operations are provided by the COMM. In the latter case there must be provided two operations by the server, i.e your application.

Dependent on the selected service port the **Direction** could be set to **Server** or **Client**



Look at it...

- > A service component contains **Service Port Prototypes** of certain **Service Port Interfaces**.
- > A **Service Port Prototype** is either **Client** or **Server**.
- > A **Service Port Interface** contains from 1 to n **Operations**.

4.6.6 How can your Software Component use a Service Component?

...and use it.

- > You have to define a service port prototype at your software component that has the same service port interface as that one from the service component
- > If the service port prototype of the service component is server, the service port prototype at your software component must be client
- > If the service port prototype of the service component is client, the service port prototype at your software component must be server. You have to provide runnables that contain the operations
- > With the service mapping you connect the service port prototype of your software component with the service port prototype of the service component

This can be done **by hand** or **automatically**.

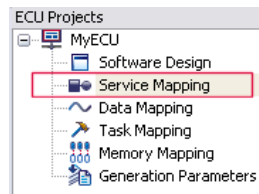
4.6.7 Use Service Component DEM as an example

Client and server port interfaces

The DEM is a service component that normally has service ports that are client and ones that are server. So it is an ideal example to learn how to use service components.

Import ARXML file for service component

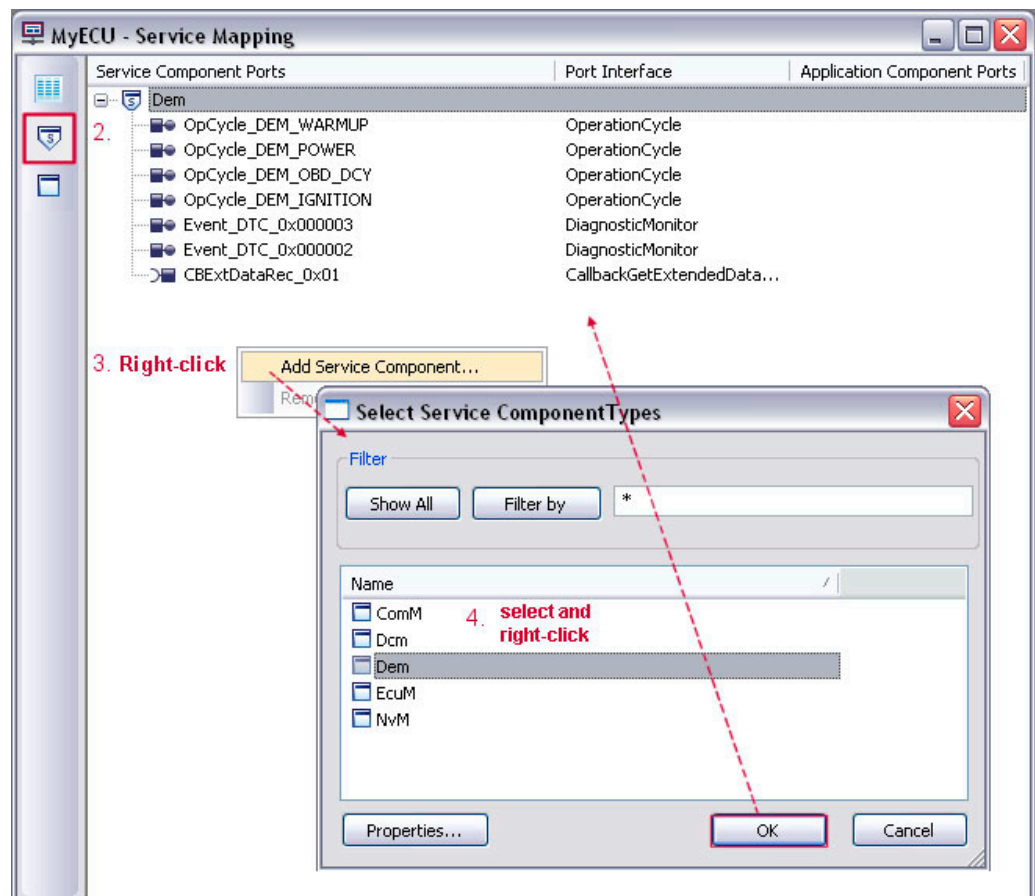
Make sure that you have imported the DEM_sw.c.xml file as shown in section **Import** a Service Component on page 72.



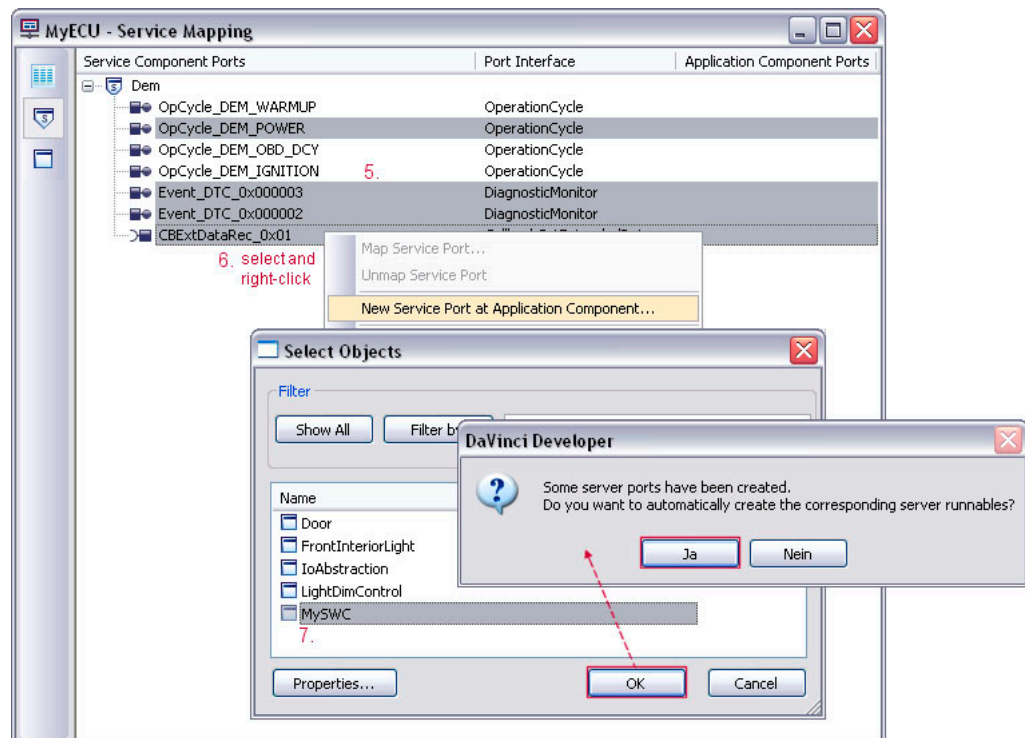
1. Open the **Service Mapping**.
2. Select the **Service Component View**
3. Right-click and select **Add Service Component...** and select DEM
4. Open DEM and see its service ports.



Client
Server



5. Select **ALL! clients** and those **servers you want to use**.
6. Right-click and select **New Service Port at Application Component...**
7. Select your application software component (e.g. MySWC)



The application is server – runnables are necessary

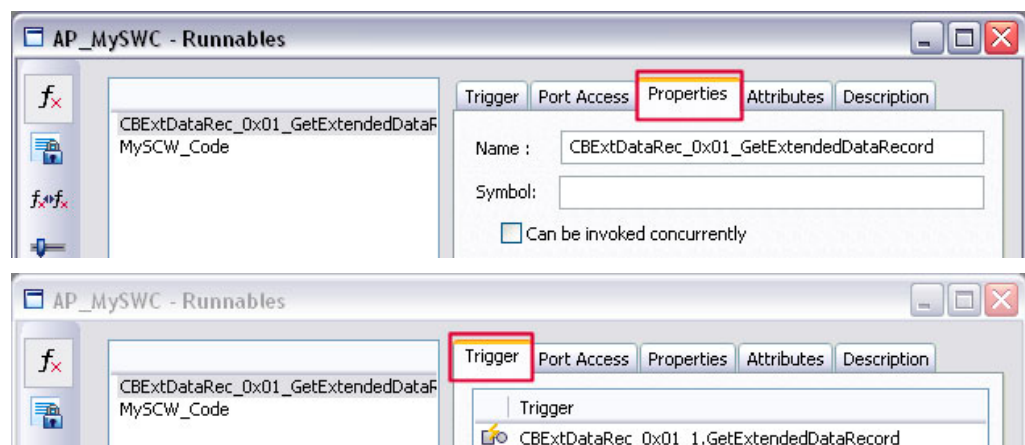
As a port interface of the DEM is a **client**, the application has to be **server** and therefore appropriate runnables have to be provided by the application. The **Developer** can generate these runnables and their triggers automatically, if you want it to do that. Confirm settings.

Where can I find the automatically created runnable?



Now select MySWC, chose the **Runnable Entity List** view and you will find the generated runnable **CBExtDataRec_0x01_GetExtendedDataRecord** with all necessary settings:

- > Name
- > You have decide about **Can be invoked concurrently** is set or not (see section Can be Invoked Concurrently on page 39)
- > Trigger is set

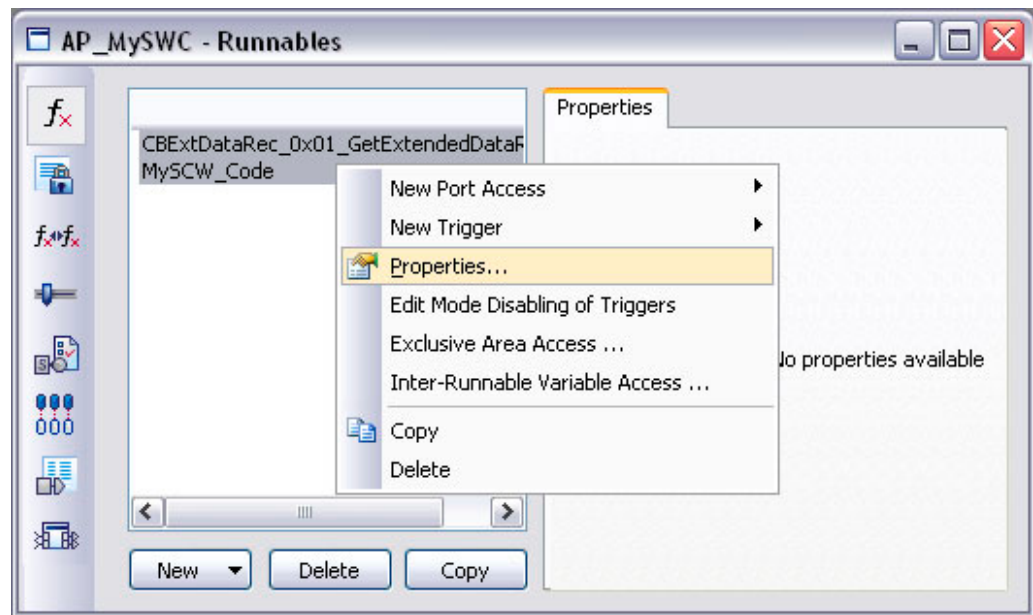


Multiple selection possible

To modify the same settings of more than one runnable, make a multiple selection using <Shift> key and open the context menu via right-click.

Select from the context menu what you want to change for all runnables

simultaneously.



4.6.8 Manually

It is also possible to create all service ports and necessary runnables manually

If you want to create the service ports for DEM and DCM manually, you first have to

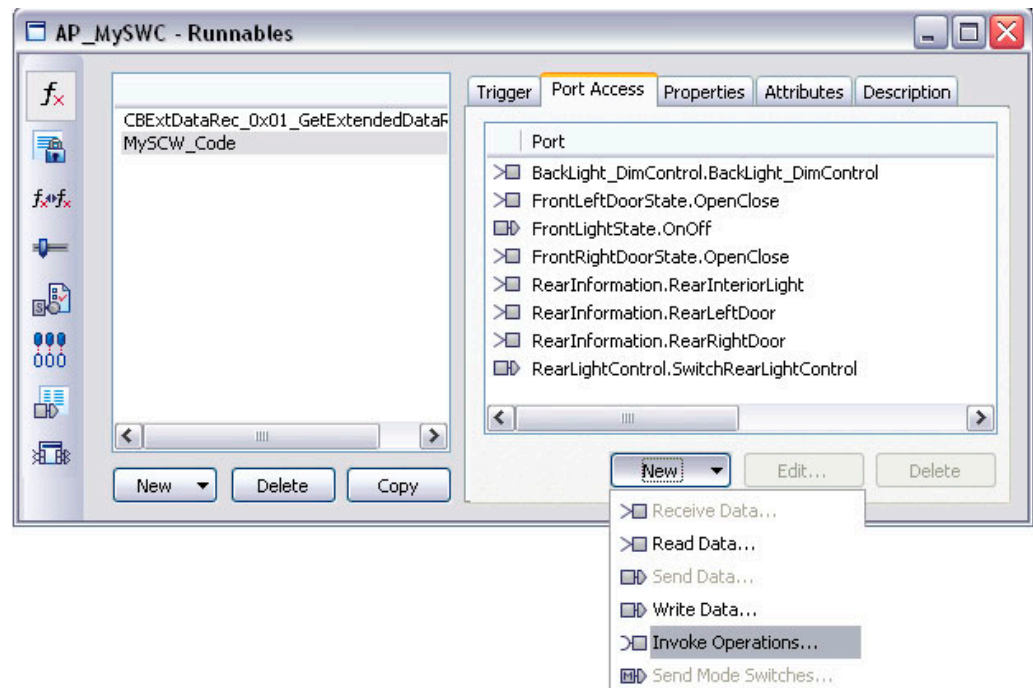
- > Create service ports at your SWC. They must base on the same port interface as the service port of the service component you want to connect with.
- > Create the service port to be a server if the service component is client and vice versa
- > If your SWC is server, create as much runnables in your SWC as operations in the service port. You can get this information in the illustration above
- > Set the triggers correctly. Each operation triggers **ONE** runnable
- > Decide about checkbox **Can be invoked concurrently** (see section **Can be Invoked Concurrently** on page 39)

4.6.9 Port Access to Service Ports

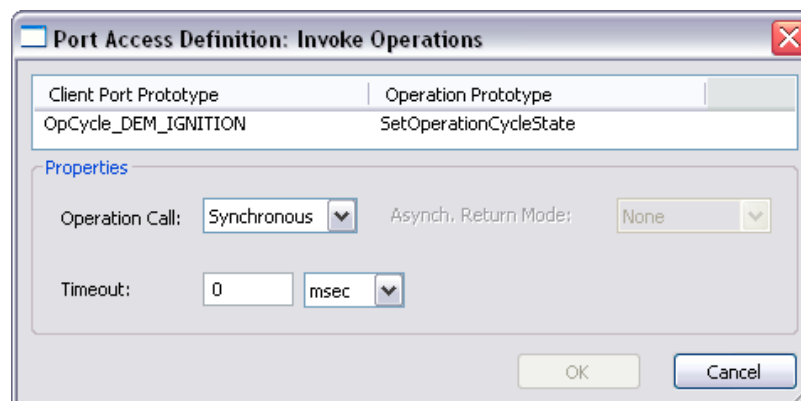
Accessing a service port is basically the same as accessing a data port.



Click **[fx]** to see the runnable entity list of a certain software component. Select the tab **Port Access**. Click **New** and select **Invoke Operations...** from the context menu.



A list opens with all available operation prototypes. Select the one you want to use and confirm with **[OK]**.



The added service port access is also displayed in the **Port** list.



Info: You can distinguish client server ports from sender receiver ports via their different icons using a rounded bracket  or circles .

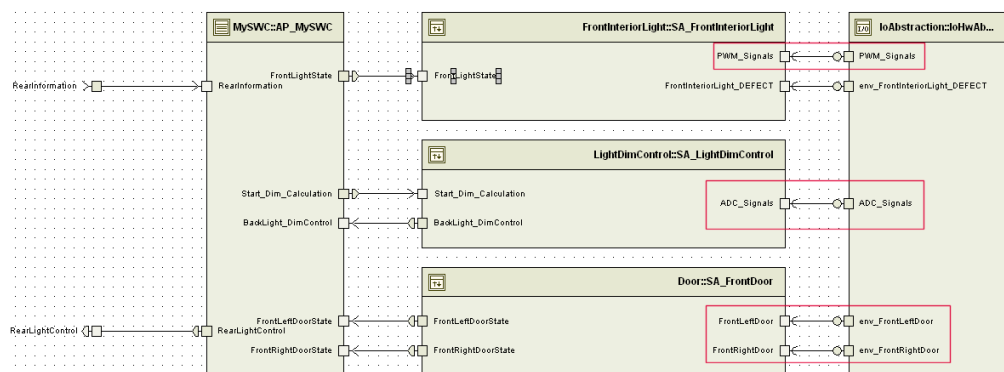
4.6.10 IOHWAB is Application Component – an Exception

IOHWAB is
application
component in demo

As application software component the IOHWAB is also imported like this is done with other application software components. Drag'n'drop it from the library to the software design view.

But then the client port interfaces of the using application software components have to be created by hand and connected to the server ports of the IOHWAB.

Create appropriate client ports and draw the connectors.



4.6.11 Port Access to new IOHWAB Services

Port access

To access the value from ADC (GET) and PWM (SET) define the appropriate port access of the runnables FrontInteriorLight and Dim_SignalCalculation.

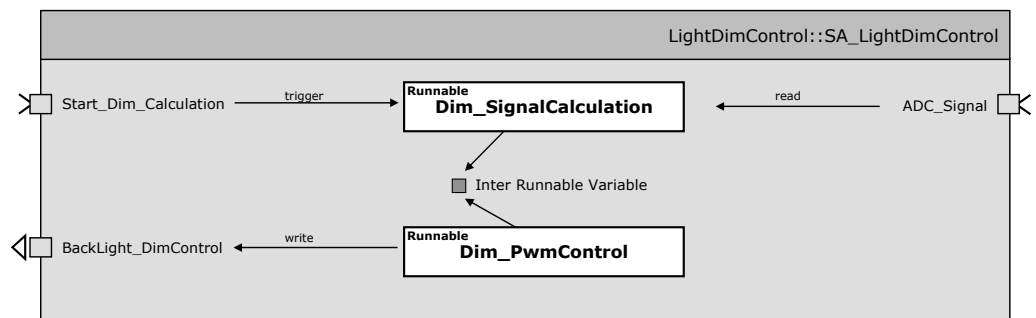
4.6.12 Inter-Runnable Variables

New runnable for PWM

In the demo the calculation of the PWM value is done via a runnable called Dim_PwmControl within the software component LightDimControl. It has port access to BackLight_DimControl.

As you see from the defined port accesses and in the illustration below the Dim_PwmControl has no access to the ADC value. But it needs that value to calculate the values for the PWM module.

Exchange information between runnables of one SWC via so-called inter-runnable variables.



This introduces the so-called **Inter-runnable Variables** to exchange information between runnables of one SWC.

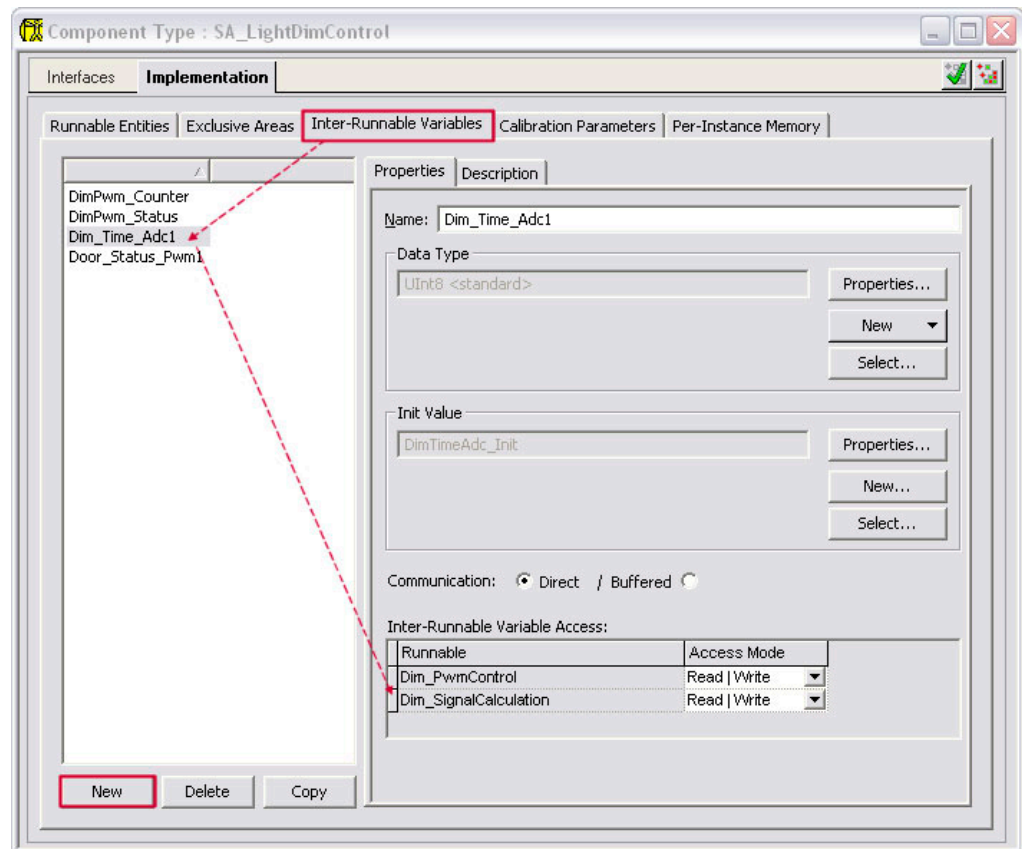
You have to:

- > Define the variables via **[New]**.
- > Provide a **Data Type** and an **Init Value**.
- > Define the **Inter-Runnable Variable Access** and select which **runnable** is allowed to access which **Inter-runnable Variable**.

The access can be

- > No Access
- > Read
- > Write

> Read | Write



Now the values for the light can be calculated, in the MySWC_Code it is determined whether the light should be switched to on or off and the dim speed is also available. The information is handed over to the SWC FrontInteriorLight. This SWC has access to the set operation of the PWM.

4.6.13 Inside IoHwAb.c

Use the shown API Now the functionality of the IOHWAB for ADC and PWM must be programmed using the API from section [Configure ADC](#) on page 55 and section [Configure PWM](#) on page 56.

To access the operations of the IOHWAB use the description about port access of a runnable. There is no further difference.

4.6.14 Store Information none-volatile – NVM Manager

To store information in non-volatile memory, there are different ways. One is using the Per-Instance memory or to use the NVM directly via service ports.

4.6.15 Per-Instance Memory

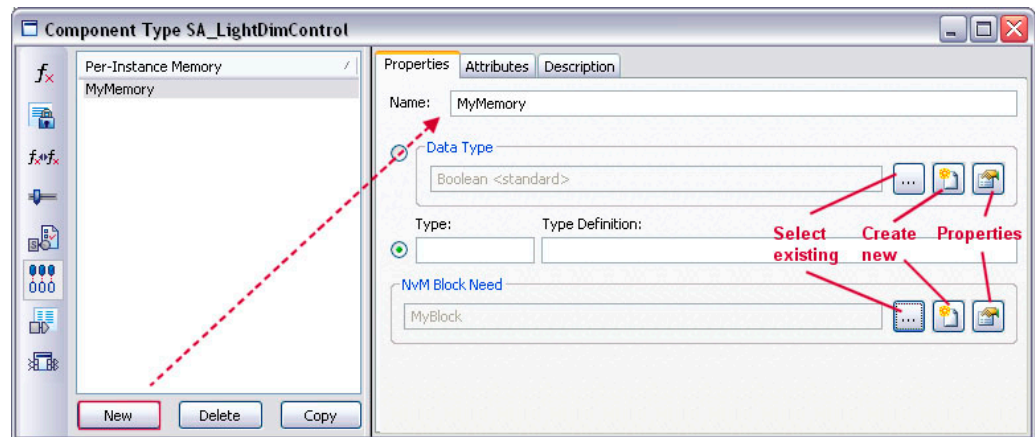
Per-Instance 1. Open the SWC **LightDimControl**,

memory tab in the Implementation tab



2. Select **Per-Instance Memory** list.
3. Click **[New]**,
4. Enter a **Name** and
5. **[Select...]** the **Data Type**.
6. Create a new NVM Block Need object using **Create New**. See [DaVinci Developer](#) online help for more details.

Define Per-Instance Memory



Memory Mapping

Close the window and select the **Memory Mapping** view. Right-click and **Create NV Memory Block**.

Open Configurator Pro

Then save workspace and open the [Configurator Pro](#). Chose the NVM and you will see your NVM block and you can configure it. Do not forget setting the **Device**.

4.6.16 Usage of WDGM

Basically using the WDGM

The WDGM is a service component and has to be handled as described above.

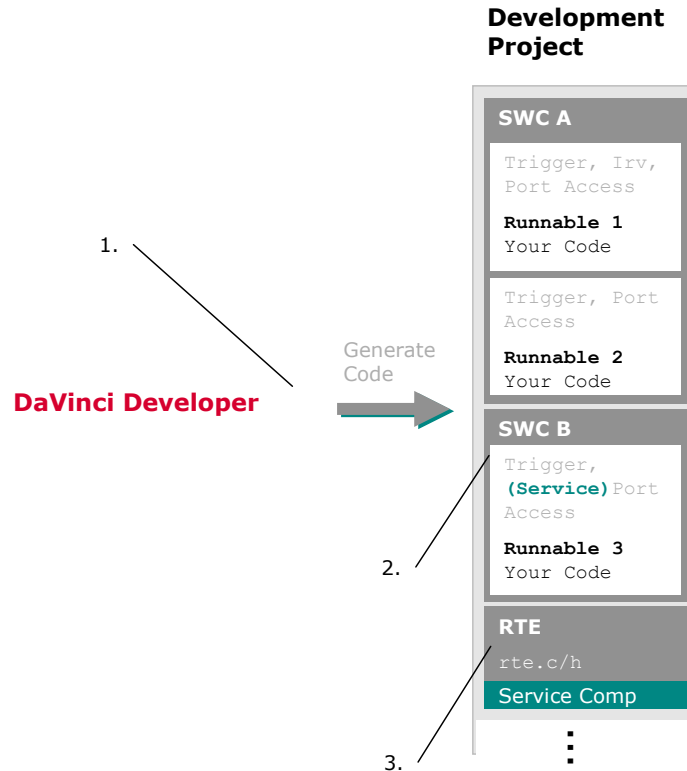
To use the functionality basically, follow this list. Configure WDGM in [DaVinci Configurator Pro](#), create XML file and then:

- > Import XML file of WDGM (like other service components)
- > Add the service component WDGM in the service mapping view
- > Define a cyclic runnable for every software component you want to realize a supervision via the watchdog
- > These runnables need a service port to the WDGM
- > Perform service mapping
- > Code of runnable: call `Rte_Call_Wdgm_Trigger_Update1_UpdateAliveCounter()` with every call of the runnable to prevent the watchdog from triggering an ECU reset
- > The application can also enable and disable (if configured) the watchdog supervision for a supervised entity if e.g. runnable is not called in a certain operation mode

4.7 Code Generation

Generate Code when your configuration work is done.

Generate code with
DaVinci Developer



1. Code Generation with DaVinci Developer
(see section [Code Generation with DaVinci Developer](#) on page 82)
2. Components Templates and Runables
(see section [Component Templates and Runables](#) on page 84)
3. RTE
(see section [RTE](#) on page 87)

4.7.1 Code Generation with DaVinci Developer



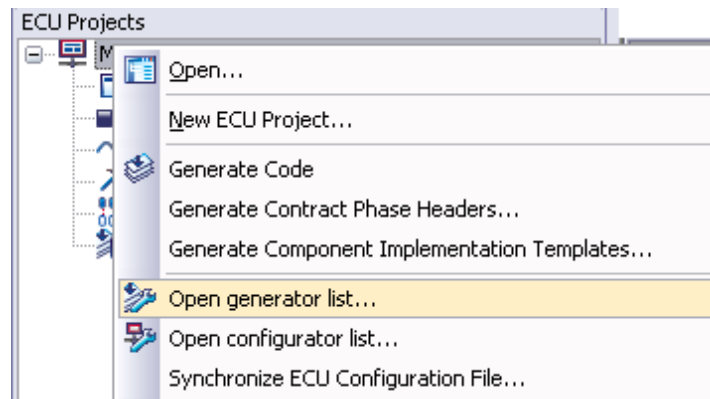
Before generation code, use the **Check** button that checks your DaVinci Developer configuration for error and missing information. To be able to use the check button you have to select the root of your ECU Project (MyECU in the demo example).



When the check is ok (see messages in output window), let the **Developer** generate the necessary code.

List of generators

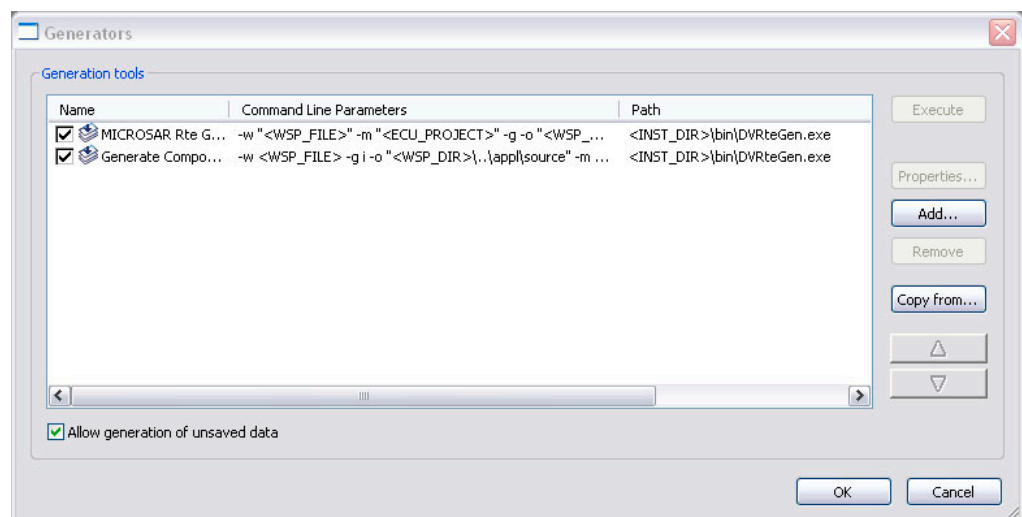
To introduce generators select **Open generator list...** from the pull down menu.



Necessary generators

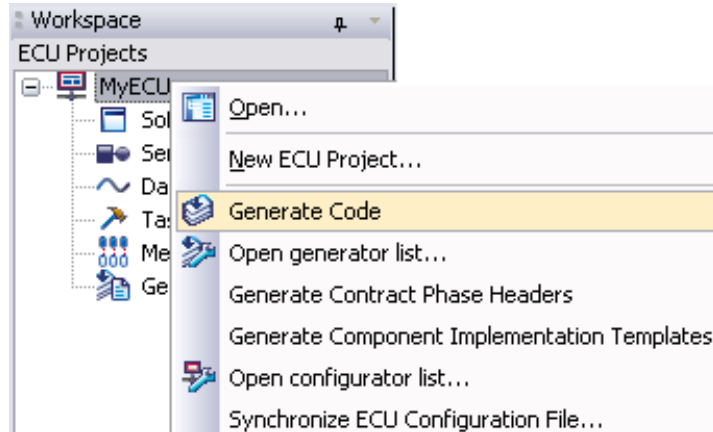
See the list of the generators below.

- > MICROSAR RTE generator – generates the RTE
- > Component Template generator – generates a C template for every software component. That is the place for you to enter your code later. With a new generation action, you code still stays unchanged.



Info: There is a exception. If a runnable is deleted in the Developer, its code will be also deleted in the generated template – but in the first generation after the deletion this runnable will be added at the end of the template file. And there is generated the *.bak file – the backup file.

Generate Code



What happens?

The **Developer** works off the generator list and generates:

- > The configuration files for the BSW
- > The operating system
- > The RTE
- > The component templates files without deleting your software parts within, if already exist

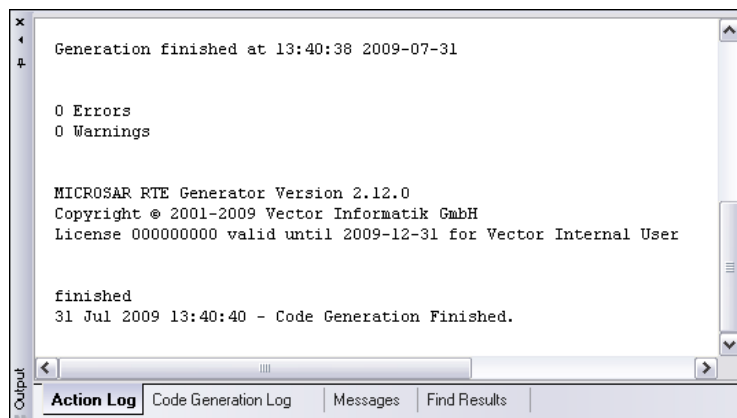
Where are the files written to?

Where the different generators write the created files depend on your settings for the target path.

Watch the Log

Watch the **Log views (Action Log, Code Generation Log)** at the bottom of the **Developer** and see this result.

- > 0 Errors?
- > 0 Warnings?
- > Finished?



Then go on!



Info: This view also informs you in case of errors that occur during the generation process. Watch it carefully.

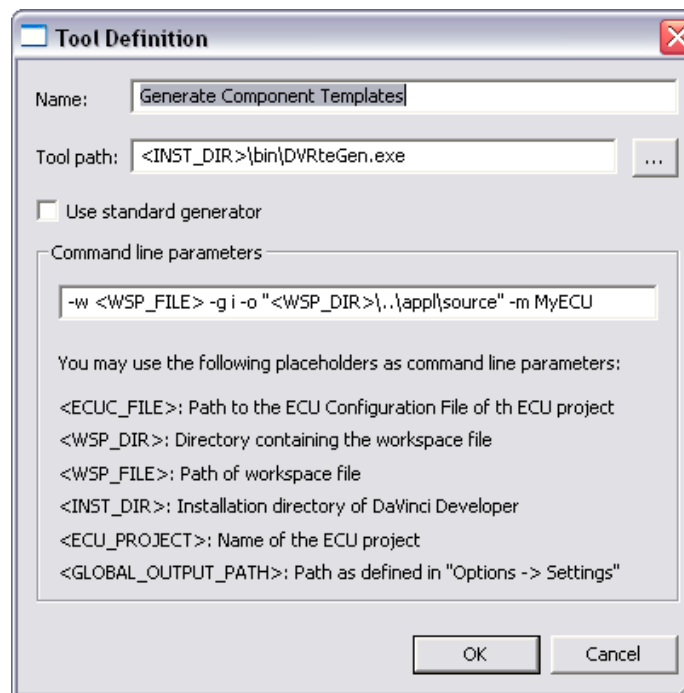
4.7.2 Component Templates and Runnables

Your code has its place in runnables. A runnable has its place within its component template file.

Until now you only have **configured** your ECU. Now it is time for coding.

The generator generates component template files containing all runnables.

When you enter the name of the ECU project template files for every software component will be generated.



With these settings shown above the **Developer** will generate component template files for every software component that is used in your software design. This is because of the name of the ECU Project (**MyECU**) is given behind the **-m** parameter.



Info: You can also generate the template for only one software component. Put the name of the component behind **-m**.

Regard target path

Also the location of the code generation is given in this command window. In this case it is path **...appl\source**.

Here you find all generated template files that are called like the **Component Type**.

4.7.3 Skeleton for runnables

Component templates and runnables

The component templates contain all runnables of the according software components. The generator generates skeletons for the runnables. The structure of those skeletons looks like shown in the illustration below.

**Example:**

Generated Skeleton for a Runnable

Name of runnable

When is runnable executed

Interfaces

- ☐ Input Interfaces
- ☐ Output Interfaces
- ☐ Service calls
- ☐ Available application errors
- ☐ Inter Runnable Variables

Start Comment

Your code

End Comment

Summary of environment conditions:
Contains Name and information when this runnable will be executed

Available Interfaces:
Lists all available interfaces of this runnable. All those listed interfaces can be used by your code.

Start Comment:
Only add code after this comment to prevent the code from being overwritten with the next generation process.

Your Code:
Enter the code for your runnable here. This code will not be overwritten with the next generation process.

End Comment:
Only add code before this comment to prevent the code from being overwritten with the next generation process.

You code stays untouched and is saved by an additional backup

Make sure to enter your code between the start and end command. Only this section is protected against modification by a generation process.

All other sections like execution information or interfaces can be changed depending on your settings in the **Developer**.



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 in the **Developer**.

4.7.4 A Skeleton Example – Runnable MySWC_Code



Example: This is an example showing the runnable **MySWC_Code** with the function name (Symbol) **MySWC_Code**

First example for
MySWC_Code:
Interface

```

/*****
 *
 * Runnable Entity Name: MySWC_Code
 *
 *****/
-----
 * Executed if at least one of the following trigger conditions occurred:
 * - triggered on DataReceivedEvent for DataElementPrototype <RearInteriorLight> of PortPrototype <RearInformation>
 * - triggered on DataReceivedEvent for DataElementPrototype <RearLeftDoor> of PortPrototype <RearInformation>
 * - triggered on DataReceivedEvent for DataElementPrototype <RearRightDoor> of PortPrototype <RearInformation>
 * - triggered on TimingEvent every 50ms
 *
 *****/
-----
 * Input Interfaces:
 * =====
 * Explicit S/R API:
 * -----
 * Std_ReturnType Rte_Read_BackLight_DimControl_BackLight_DimControl(UInt8 *data)
 * Std_ReturnType Rte_Read_FrontLeftDoorState_OpenClose(Boolean *data)
 * Std_ReturnType Rte_Read_FrontRightDoorState_OpenClose(Boolean *data)
 * Std_ReturnType Rte_Read_RearInformation_RearInteriorLight(DT_Signal_RearInteriorLight *data)
 * Std_ReturnType Rte_Read_RearInformation_RearLeftDoor(DT_Signal_RearLeftDoor *data)
 * Std_ReturnType Rte_Read_RearInformation_RearRightDoor(DT_Signal_RearRightDoor *data)
 *
 * Output Interfaces:
 * =====
 * Explicit S/R API:
 * -----
 * Std_ReturnType Rte_Write_FrontLightState_OnOff(Boolean data)
 * Std_ReturnType Rte_Write_RearLightControl_SwitchRearLightControl(DT_Signal_SwitchRearIntel6128EC8 data)
 * Std_ReturnType Rte_Write_Start_Dim_Calculation_Start_Dim_Calculation(UInt8 data)
 *
 * Service Calls:
 * =====
 * Service Invocation:
 * -----
 * Std_ReturnType Rte_Call_UR_ComMUser_CAN_GetCurrentComMode(ComM_ModeType *ComMode)
 * Synchronous Service Invocation. Timeout: None
 * Returned Application Errors: RTE_E_ComM_UserRequest_E_NOT_OK
 * Std_ReturnType Rte_Call_UR_ComMUser_CAN_GetRequestedComMode(ComM_ModeType *ComMode)
 * Synchronous Service Invocation. Timeout: None
 * Returned Application Errors: RTE_E_ComM_UserRequest_E_NOT_OK
 * Std_ReturnType Rte_Call_UR_ComMUser_CAN_RequestComMode(ComM_ModeType ComMode)
 * Synchronous Service Invocation. Timeout: None
 * Returned Application Errors: RTE_E_ComM_UserRequest_CONM_E_MODE_LIMITATION, RTE_E_ComM_UserRequest_E_NOT_OK
 *
 *****/

```

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



Caution: If some access is missing, go back to the **Developer**, add the necessary port access for your runnable and generate the component templates again.

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

First example for MySWC_Code: Code

```

FUNC(void, RTE_AP_MYSWC_APPL_CODE) MySWC_Code(void)
{
    /****** << Start of runnable implementation >> ***** DO NOT CHANGE THIS COMMENT!
    * DO NOT CHANGE THIS COMMENT!
    * Symbol: MySWC_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);
        }
    }
    /****** << End of runnable implementation >> ***** DO NOT CHANGE THIS COMMENT!
    * DO NOT CHANGE THIS COMMENT!
    *****/
}

```

Communication and mode management

The doors are checked. If any door is open, the FrontInteriorLight and the RearLight should be switched to on.

Request COM mode, get requested COM mode

To switch the communication mode of the COMM there are a few services (operations) available. Switch the communication mode to desired state. Remember that such a state switch takes time.



Info: Before you switch a state you can use the operation GetRequestedComMode to find out which mode is already requested.

4.7.5 RTE

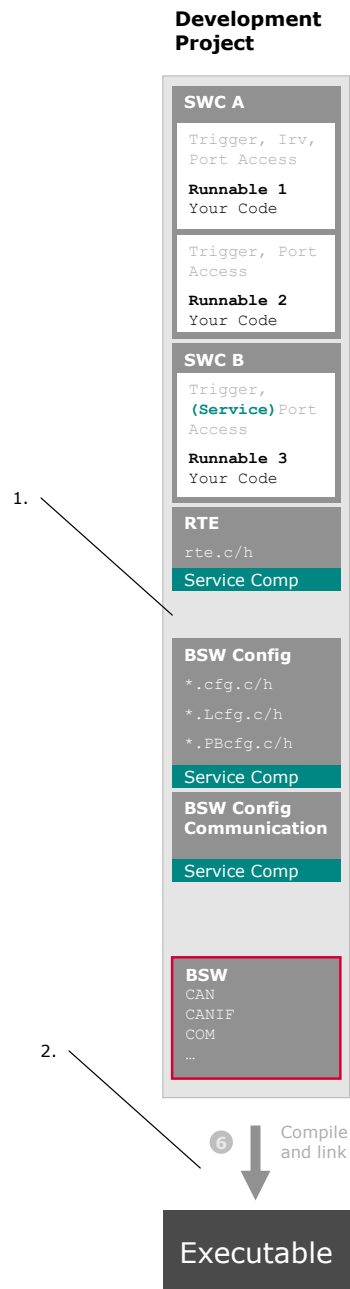
MICROSAR Rte Gen

The RTE is generated by the MICROSAR Rte Gen. Its generation depends on many settings in the configuration tools and how runnables are mapped to tasks.

The files start with **Rte_**.

4.8 Generate Final Code

This step is highly dependent on your project management, compiler, linker, etc.



1. Setup Your Build Environment
(see section [Setup Your Build Environment](#) on page 89)
2. Compile, Link and create your Executable
(see section [Compile, Link and create your Executable](#) on page 90)

4.8.1 Setup Your Build Environment

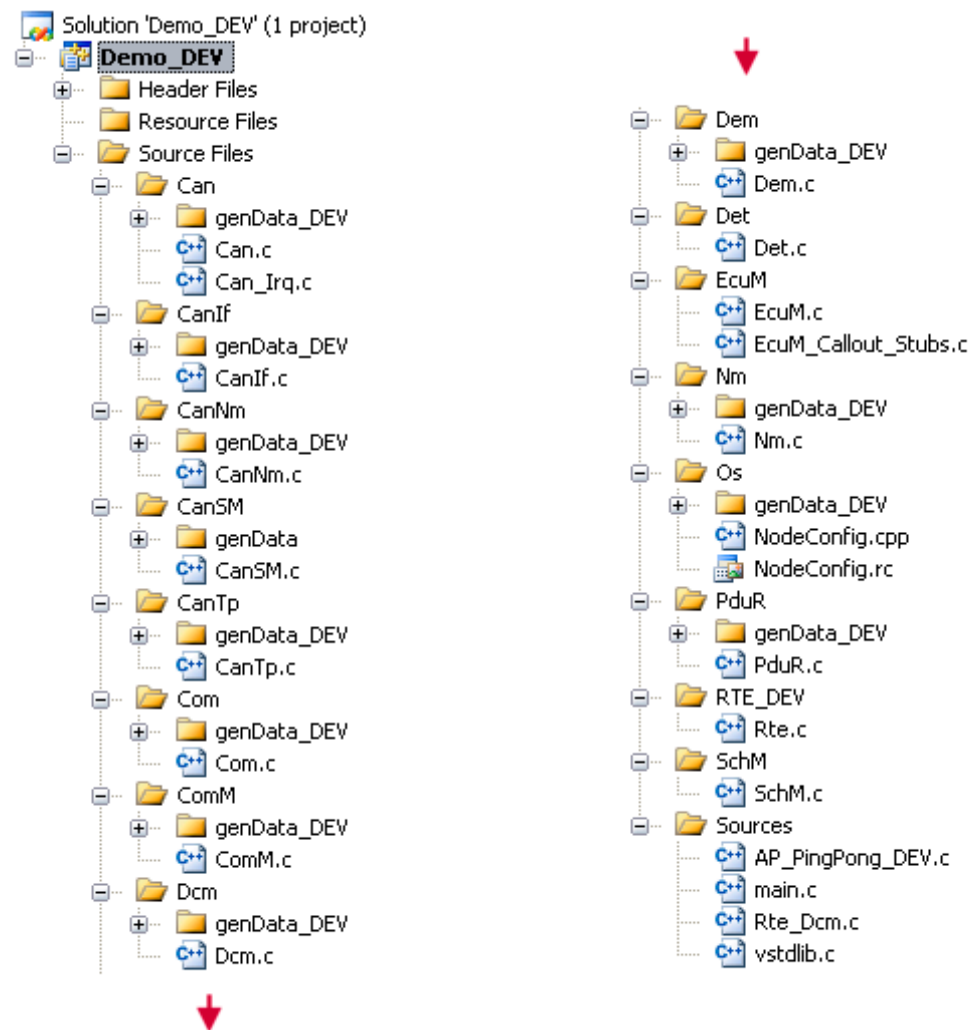


What to do with the generated configuration files?

Now the code is generated. But what is generated? What to do now to get this to a running system?

Just an example

This screenshot from the **Microsoft Visual Studio™** is merely an example how a project could look like.



What should be in your project?

Make sure your project includes all files from:

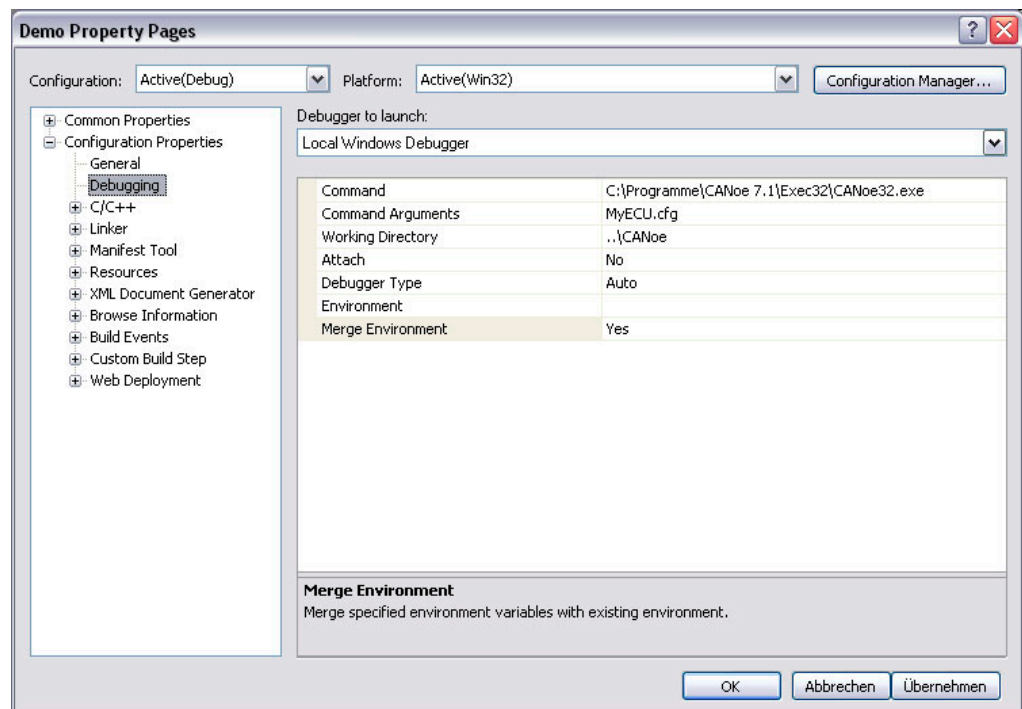
- > the BSW delivery (CAN, CANIF, CANNM, OS...)
- > The configuration files of **GENy**
- > The configuration files of **DaVinci Configurator Pro**
- > The generated RTE (RTE_DEV)
- > And the folder **Sources** with the component templates filled with your code.

Using CANoe as

All generated code has to be compiled and linked.

target

CANoe as debugger Enter **CANoe** as debugger like shown in the screenshot below.



Comfortable
debugging using
breakpoints

Now you can run the ECU together with other ECUs, programmed like this one and represented as DLL or as remaining bus simulation. You can debug the system using breakpoints in **Microsoft Visual Studio™**.

DLL

When working with **CANoe** as target you use **Microsoft Visual Studio™** to build the project. The result is a DLL which represents your first AUTOSAR compliant ECU. To run the DLL you may now start **CANoe** and load the configuration file **MyECU.cfg**.

4.9 Compile, Link and create your Executable



Compile, Link and test your code now.

5 Additional Topics

In this chapter you find the following information:

5.1	SIP Check, Version Check, Generator Check SIP Check Generator Check Error Function in EcuM_CalloutStubs.c	Seite 92
5.2	Flash Bootloader Topics Valid Application or Flash Bootloader and Invalid Application	Seite 93
5.3	DCM Template for Transition into Bootloader	Seite 93
5.4	Update ECUC via Project Assistant	Seite 94

5.1 SIP Check, Version Check, Generator Check

5.1.1 SIP Check

SIP check is a compile check

There is a C file called **SIP_Check.c**. When you add this file to your compiler project it checks whether all corresponding C and H files have the correct and identical version.

The SIP check also checks versions between libraries and their H files.



Info: When you compile your project **WITH** SIP_Check.c a **step-wise integration** is not possible. Do not use the SIP check during this sort of integration work.



Caution: But it is highly recommended to use the SIP check at the end of your integration.

5.1.2 Generator Check

Compares generator versions

Generator version checks are implemented in case of:

- > Delivery of libraries
- > Postbuild

5.1.2.1 Libraries are delivered

Same generator for libraries and configuration files

The generator that builds the libraries must be the same as used by the TIER1 to generate the configuration files for the libraries.

ECUM

This is checked for every used BSW module and in case of an error a function within the file **EcuM_CalloutStubs.c** is called.

5.1.2.2 Postbuild is used

ECUM

In case of postbuild concept the generator for the post build configuration must have the same version as the generator for the library and for the configuration files for the library.

If not, the error function within the file **EcuM_CalloutStubs.c** is called.

5.1.3 Error Function in EcuM_CalloutStubs.c

ECUM collects all negative check results

```
FUNC(void, ECUM_CODE) EcuM_GeneratorCompatibilityError(uint16
ModuleId, uint8 InstanceId)
```

5.2 Flash Bootloader Topics

5.2.1 Valid Application or Flash Bootloader and Invalid Application

Only valid applications are allowed to be started. If an application is marked as invalid, the execution should stay in the bootloader.

One possible
behaviour

Within this function decide whether to start the application or to define the application invalid and jump into the bootloader.

5.3 DCM Template for Transition into Bootloader

Application and
bootloader

Application and bootloader never run simultaneously. After a reset, the bootloader is started first. Within the bootloader there is met the decision whether to

- > stay in the bootloader because application is not valid or a new application has to be flashed
- > jump into the valid application

From application to
bootloader

The transition from application to bootloader is necessary when there is received a request that the application has to be flashed now. After some preparations a reset is provoked and after that the bootloader is started. The previously made preparations tell the bootloader not to jump into the application but to start the flash process.

Template file

There is a template file you have to fill in your code to perform the transition from your application to the bootloader. This is mainly used for communication with the bootloader via variables stored in non-volatile memory.

DcmFbIS.c

The file **DcmFbIS.c** contains several functions that are empty as they are defaults to be filled by you.



Cross reference: Refer to the TechnicalReference_DCM for detailed information.

5.4 Update ECUC via Project Assistant

OEM modifies ECU Extract of System Description.

Sometimes the OEM changes the ECU Extract of System Description because of changes e.g. in signals, PDUs, frames, ...

These changes must find their way into the ECUC file you are currently working with.



How?

This is described in the following.

You can start the Project Assistant for updating your ECUC file from different places.

- > **Start | Programs | Vector AUTOSAR Projects | <Your Project name> | Update Project** (Only if **Create entries in the start menu** is selected)
- > You can start the project assistant from each AUTOSAR tool (**GENy**, **DaVinci Developer**, **DaVinci Configurator Pro**)
- > You can start the project assistant from **<your project folder>/Config/Update Project**



1. Start the project assistant with **Update Project**.
2. Open **7 Input files**
3. Delete  the ECU Extract
4. Add  the new ECU Extract.
5. Apply to start the ECU Update process

What happens while updating

The Project Assistant has to know the new ECU Extract of System Description.

Before the Project Assistant updates the ECUC file and the project workspace (inclusive all project data), the current versions will be saved in a backup folder (backup_dvpa_hh-mm-ss_yyyy-mm-dd).

The updater generates a new initial ECUC file based on the new ECU Extract and overwrites the current version on the working folder.



Info: The backup folder is generated while the update process.

Result of update process

The result is an updated project and a backup folder with the previous project data, saved in your project folder.

6 What's new, what's changed

In this chapter you find the following information:

6.1	Version 3.7	Seite 96
	What's new	
	What's changed	
6.2	Version 3.8	Seite 97
	What's new	
	What's changed	
6.3	Version 3.9.0	Seite 98
	What's new	
	What's changed	
6.4	Version 4.0.0	Seite 99
	What's new	
	What's changed	
6.5	Version 4.1.	Seite 100
	What's new	
	What's changed	

6.1 Version 3.7

What's new and what's changed This explains the changes within this document from the previous Version to the one mentioned in this headline.

6.1.1 What's new

New chapter Persistors setup and update There is a new chapter for additional information about Persistors setup and update at chapter additional information.

Create an Initial ECUC The creation of an Initial ECU is described here

Update ECUC There is a new chapter how to update you ECUC file in case of modifications .

6.1.2 What's changed

Document name changed The document name has changed from UserManual_MICROSAR to Startup_Vector_AUTOSAR_Solution.
The new name better describes the content of the document and the startup prefix shows that this is an introduction manual that should be read first.

Demo example slightly modified The described demo example is slightly changed. The light now has also dim functionality which has been controlled via ADC and PWM. This causes different software design and module configuration.

Difference in delivery The delivery place of GENy Framework and DataServer has changed. Now the setup.exe files are available on the FTP server.

Overview Tools and Files The overview of tools and files has changed.

Range of Basic Software The range of Basic Software has been enlarged.

ECU Extract The document now concentrated completely on the ECU Extract of System Configuration, the usage of DBC is mentioned only.

6.2 Version 3.8

What's new and what's changed

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

6.2.1 What's new

There is nothing new, please regard the changes.

6.2.2 What's changed

Update ECUC Workflow figure

Workflow figures in section Generate ECUC and Update ECUC have been updated.

6.3 Version 3.9.0

What's new and what's changed This explains the changes within this document from the previous Version to the one mentioned in this headline.

6.3.1 What's new

New versioning From now on the version number has three digits to avoid increasing version numbers because of minor changes or typos.

Hint for reading order There is a hint added to read the Startup_OEM.pdf first, if available in your delivery.

6.3.2 What's changed

Currently no changes.

6.4 Version 4.0.0

What's new and what's changed

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

6.4.1 What's new

Completely new

The version 4.0.0 is completely reworked. It is designed to fit the Release 8 of Vector AUTOSAR Solution.

6.4.2 What's changed

No introduction to build up the demo example

Now this manual is not a guide to rebuild the delivered demo example. It explains necessary step when configuring your ECU (like usage of service components as an example) and the demo example is for you to follow the steps explained here.



Caution: This is the reason why changes, described in previous versions are only mentioned, but cannot be found anymore. Version 4.0.0 is very different from the previous versions.

6.5 Version 4.1.x

What's new and what's changed

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

6.5.1 What's new

DaVinci Package Merger

The DaVinci Package Merger was added to the DaVinci Project Assistant chapter.

Update ECUC

Add Update ECUC via Project Assistant in chapter Additional Topics

6.5.2 What's changed

Project Assistant

Regard new caution (see section **ECU Project Preparations – the Project Assistant** on page 24).

7 Address table

Vector Informatik GmbH	Vector Informatik GmbH Ingersheimer Str. 24 D-70499 Stuttgart Phone: +49 (711) 80670-0 Fax: +49 (711) 80670-111 mailto:info@de.vector.com http://www.vector.com
Vector CANtech, Inc.	Vector CANtech, Inc. Suite 550 39500 Orchard Hill Place USA- Novi, Mi 48375 Phone: +1 (248) 449 9290 Fax: +1 (248) 449 9704 mailto:info@us.vector.com http://www.vector.com
Vector France SAS	Vector France SAS 168, Boulevard Camélinat F-92240 Malakoff Phone: +33 (1) 4231 4000 Fax: +33 (1) 4231 4009 mailto:information@fr.vector.com http://www.vector.com
Vector GB Ltd.	Vector GB Ltd. Rhodium Central Boulevard Blythe Valley Park Solihull, Birmingham West Midlands B90 8AS United Kingdom mailto:info@uk.vector.com http://www.vector.com

Vector Japan Co., Ltd.
Vector Japan Co., Ltd.
Seafort Square Center Bld. 18F
2-3-12, Higashi-shinagawa, Shinagawa-ku
J-140-0002 Tokyo
Phone: +81 3 (5769) 7800
Fax: +81 3 (5769) 6975
<mailto:info@jp.vector.com>
<http://www.vector.com>

Vector Korea IT Inc.
Vector Korea IT Inc.
#1406 Mario Tower
222-12 Guro-dong, Guro-gu
Seoul, 152-848
Republic of Korea
Phone: +82 2 8070600
Fax: +82 2 8070601
<mailto:info@kr.vector.com>
<http://www.vector.com>

VecScan AB
VecScan AB
Theres Svenssons Gata 9
SE-417 55 Göteborg
Phone: +46 (31) 76476-00
Fax: +46 (31) 76476-19
<mailto:info@se.vector.com>
<http://www.vector.com>

8 Glossary

BAM	Broadcast Announce Message
CMDT	Connection Mode Data Transfer
COM	AUTOSAR Communication layer
ECUC	ECU Configuration Description
ECUEX	Extract of System Description
PCI	Protocol Control Information
PDU	Protocol Data Unit
PGN	Parameter Group Number
RTE	AUTOSAR Run Time Environment
S/R Port	Sender/Receiver Port of the application component
SDU	Service Data Unit
SPN	Suspect Parameter Number
VFB	Virtual Function Bus, abstracted view of the communication between application software components

9 Index

A

Action Log	82
ADC	54
Add Service Component	73
Alarms	49
Application Component	18, 56, 76
application component prototype	29
Application Component Type	28
Application Port Interface	31
application ports	70
application software components	70
ARXML	70

B

BSW	14
BSW Configuration	45
Buffered	40
Build Environment	88

C

CAN	18, 53, 64
Can be invoked concurrently	42
Can be Invoked Concurrently	38
CANIF	65
CANNM	64
CanoeemuCanoe	64
CANTP	64
Client	72
client port	69
Client Server Theory	69
Code Generation	18, 37, 80, 81
Code Generation Log	82
COM	18, 64
COMM	65, 70
Communication	18, 45, 62

Component Templates	18, 80, 83
Configure BSW	18, 45, 47, 62
Connect Ports	34
Connections	14, 27
Constants	32

D

Data Element Prototypes	31
Data Element View Mode	43
data elements	70
Data Elements	14, 30
Data Exchange Formats	14
Data Mapping	27, 43
data types	30
DaVinci Configurator Pro	21
DaVinci Developer	21
DCM	66, 70
DCM Template	90, 92
Delegation Ports	34
DEM	66, 70
Design Software Components	18, 27
Development Project	22
Diagnostics	64
DIO	18, 54
Direct	40
drag'n'drop	32

E

ECU	13
ECU Configuration Description	12, 15
ECU Extract of System Description	12, 15
ECUM	52, 70
EcuM_sw.c.arxml	54
Events	18, 49
Executable	89

F

Fee.....	18, 61
FIM.....	70
Flash Bootloader.....	90, 92

G

Generate Configuration	18, 58
Generator Check	91
GENy	21, 62

I

Import.....	18, 68, 71, 73
InstallationGuide.pdf.....	20
Inter-runnable Variables	77
IOHWAB	56, 70
IOHWAB Services	77

L

Log views	82
-----------------	----

M

Mapping	37
Memory.....	18, 59
Microsar	6

N

Network Signal.....	43
New Task.....	41
none-volatile	18, 78
NVM.....	18, 61
NVM Manage.....	18, 78

O

Operation	72
operations	70
OS.....	48
OSDEFAULTAPPMODE	51
OsTaskAppModeRef	51
OsTaskAutostart.....	51

P

PDU	64
-----------	----

Per-Instance Memory.....	78
Port Access.....	18, 27, 37, 39, 75
Port Init Values.....	18, 27, 30, 32
Ports.....	14, 30
Ports and Data Elements.....	27
Project Assistant	21, 25
PWM	55

R

Receiving Data.....	35
RTE.....	14, 18, 42, 86
RTE generator	41
RTE Generator.....	42
Runnables.....	13, 27, 37

S

SCHM	18, 48, 51
SchM_Alarm	51
SchM_Event.....	51
SchM_Task.....	51
Sending Data	35
Server	72
server port.....	69
service components.....	70
Service Components.....	68, 70
Service Mapping	68, 73
service ports.....	70
SIP Check	90, 91
Skeleton	18, 37, 83
Skeleton Example	18, 85
Software Components	13, 27
Software Design.....	28
Step by Step	18
Synchronize ECUC.....	18, 45, 46
System Description.....	15

T

Task Mapping	41
Tasks and Task Mapping.....	27
The RTE.....	27

Index

tools	20
Transition	90, 92
Trigger	37
Triggers.....	18, 27, 38

V

Version 3.7.....	94
Version 3.8.....	95
Version 3.9.0.....	96
Version 4.0.0.....	97

Version Check.....	90, 91
--------------------	--------

W

Wakeup Source	52
Watchdog.....	61
WDG	61
WDGIF	62
WDGM	62, 70, 79
What's new, what's changed	93

Get more Information!

Visit our Website for:

- > News
- > Products
- > Demo Software
- > Support
- > Training Classes
- > Addresses

www.vector-worldwide.com