SystemDesk

Tutorial

For SystemDesk 5.5

Release 2020-B – November 2020

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

Contents

About This Document 9

Introduction to SystemDesk 11

Basics on SystemDesk ... 11
How to Start SystemDesk ... 14
SystemDesk's User Interface ... 16

Introduction to this Tutorial 19

Introduction and User Profiles ... 19
Your First SystemDesk Task ... 20
Overview of Lessons ... 22
Basics on Demo Scripts ... 23
How to Extract Demo Scripts ... 24
How to Run Demo Scripts ... 26

Lesson 1: Setting up Your First Project 29

Overview of Lesson 1 ... 29
Step 1: How to Create a Project ... 30
Step 2: How to Add a Package to the Project ... 31
Step 3: How to Save, Close and Open a Project ... 32
Step 4: How to Import AUTOSAR Templates ... 33
Result of Lesson 1 ... 36

Lesson 2: Modeling and Connecting Software Components 39

Overview of Lesson 2 ... 40
Step 1: How to Create Software Components ... 41
Step 2: How to Create and Open Composition Diagrams ... 43
Step 3: How to Add Software Components to a Composition SW Component Type ... 44
Step 4: How to Customize Composition Diagrams ... 47
Step 5: How to Add Ports to Software Components ... 51
Step 6: How to Connect Software Components ... 54

Step 7: How to Connect a Composition SW Component Type to Its Inner Software Components ... 57
Result of Lesson 2 ... 59

Lesson 3: Modeling Sender Receiver Interfaces of Software Components 61

Overview of Lesson 3 ... 62
Step 1: How to Create Sender Receiver Interfaces ... 63
Step 2: How to Add Variable Data Prototypes to Sender Receiver Interfaces ... 66
Step 3: How to Define Application Data Types and Compu Methods ... 68
Step 4: How to Assign Data Types to Variable Data Prototypes ... 73
Step 5: How to Assign Interfaces to Ports ... 75
Step 6: How to Specify Queued or Nonqueued Communication and Initial Values ... 80
Result of Lesson 3 ... 88

Lesson 4: Modeling Client Server Interfaces of Software Components 91

Overview of Lesson 4 ... 92
Step 1: How to Define Data Types and Data Constr Elements ... 93
Step 2: How to Define Compu Methods, Units and Physical Dimensions ... 96
Step 3: How to Create Client Server Interfaces ... 101
Step 4: How to Add and Define Client Server Operations ... 103
Step 5: How to Assign Compu Methods to Data Types, and Data Types to Argument Data Prototypes ... 106
Step 6: How to Assign Client Server Interfaces to Ports ... 110
Result of Lesson 4 ... 112

Lesson 5: Modeling the SWC Internal Behavior of Atomic Software Components 115

Overview of Lesson 5 ... 116
Step 1: How to Add an SWC Internal Behavior ... 119
Step 2: How to Create Runnable Entities ... 120
Step 3: How to Trigger Runnable Entities with RTE Events ... 123
Step 4: How to Create Interrunnable Variables ... 127
Step 5: How to Specify Data Access by Runnable Entities ... 134
Result of Lesson 5 ... 138

Lesson 6: Modeling Measurements and Calibration Access 141

Overview of Lesson 6 141
Step 1: How to Create Measurements 142
Step 2: How to Create Calibration Parameters 143
Result of Lesson 6 151

Lesson 7: Specifying Data Type Mapping Sets and Constant Specification Mapping Sets 153

Overview of Lesson 7 153
Step 1: How to Map Application to Implementation Data Types 154
Step 2: How to Map Constants 158
Result of Lesson 7 164

Lesson 8: Exchanging SWC Containers 167

Overview of Lesson 8 167
Step 1: How to Export SWC Containers 168
Step 2: How to Integrate an SWC Implementation 171
Result of Lesson 8 175

Lesson 9: Creating and Configuring an ECU Instance 177

Overview of Lesson 9 177
Step 1: How to Create an ECU Instance 178
Step 2: How to Configure ECU Instances 179
Result of Lesson 9 182

Lesson 10: Importing Network Communication Elements 185

Overview of Lesson 10 185
Step 1: How to Import Network Communication Elements 186
Result of Lesson 10 188

Lesson 11: Modeling the System 189

Overview of Lesson 11 190
Step 1: How to Create a System 191
Step 2: How to Map Software Components to ECU Instances 195

Step 3: How to Select SWC Implementations..... 198
Step 4: How to Map SWC Communication to Network Communication..... 200
Result of Lesson 11..... 202

Lesson 12: Importing and Exporting Data 205

Overview of Lesson 12..... 205
Step 1: How to Use AUTOSAR Master Files..... 206
Step 2: How to Export System Extracts..... 212
Step 3: How to Import AUTOSAR Master Files..... 214
Result of Lesson 12..... 215

Lesson 13: Modeling ECU Configurations and Basic Software Components 217

Overview of Lesson 13..... 218
Step 1: How to Create an ECU Configuration..... 219
Step 2: How to Create the I/O Hardware Abstraction..... 223
Step 3: How to Configure the ECU State Manager and the COM Stack..... 225
Step 4: How to Map Runnables to OS Tasks..... 228
Step 5: How to Complete the OS Configuration..... 231
Step 6: How to Generate the RTE..... 234
Result of Lesson 13..... 237

Lesson 14: Generating V-ECU Implementations 239

Overview of Lesson 14..... 240
Step 1: How to Create V-ECUs..... 241
Step 2: How to Perform a V-ECU Test Build..... 244
Step 3: How to Export V-ECU Implementation Containers..... 246
Step 4: How to Build a Simulation System with VEOS..... 248
Result of Lesson 14..... 253

Summary 257

Your Working Results..... 257

Glossary 259

Index 271

About This Document

Content

This tutorial guides you through your first steps with SystemDesk. It is divided into short lessons in which you learn typical procedures in SystemDesk step-by-step. You should work through these lessons before using the more detailed *SystemDesk Guide*.

Target group

This tutorial is designed for engineers who develop distributed automotive electric/electronic (E/E) systems and subsystems.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

```
%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>
```

or

```
%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>
```

Documents folder A standard folder for user-specific documents.

```
%USERPROFILE%\Documents\dSPACE\<ProductName>\
<VersionNumber>
```

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

```
%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\
<ProductName>
```

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as Adobe® PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Introduction to SystemDesk

Where to go from here

Information in this section

Basics on SystemDesk 11
SystemDesk is a tool that lets you model software architectures according to AUTOSAR and create individual V-ECU implementation containers to be used in offline and real-time simulations for virtual validation.

How to Start SystemDesk 14
To start SystemDesk.

SystemDesk's User Interface 16
Provides an overview of the control bars and areas of SystemDesk's user interface.

Basics on SystemDesk

Introduction

SystemDesk is a tool that lets you model software architectures according to AUTOSAR and create individual V-ECU implementation containers to be used in offline and real-time simulations for virtual validation.

Classic Platform support

AUTOSAR release for modeling SystemDesk lets you model Classic Platform software and system architectures with a data model according to the AUTOSAR 19-11 Release. However, SystemDesk lets you exchange data according to other AUTOSAR releases as well.

Data exchange support SystemDesk supports AUTOSAR 19-11, 4.4.0, 4.3.1, 4.3.0, 4.2.2, 4.2.1, 4.1.3, 4.1.2, 4.1.1, 4.0.3, and 4.0.2 for data exchange.

Support of all the elements defined in the AUTOSAR XML schema SystemDesk supports all the elements defined in the AUTOSAR XML schema: When you import data to SystemDesk and export it later on, no data is lost.

Adaptive Platform support

SystemDesk supports AUTOSAR 19-11 for developing Adaptive Platform software. For exchanging data, AUTOSAR 19-11 and 19-03 are supported.

Software architecture modeling

One of SystemDesk's focal points is *modeling the automotive software architecture*, that is, the convenient design of software components with interfaces and internal behaviors.

A software architecture is the model of the software components in an electrics/electronics system, including their interconnections. Each software component (SWC) represents a functionality, a subfunction or even a set of functions. Software components can be hierarchically structured and contain an SWC internal behavior with runnable entities. When designing the software architecture, developers do not necessarily have to take the basic software of ECUs and the topology of the ECU network into account since this is modeled separately. Refer to Introduction to Modeling Software Architectures (🕮 SystemDesk Manual).

Elements of a software architecture are defined in the *AUTOSAR Software Component Template* document.

You can find it at http://www.autosar.org.

For the important elements of the *AUTOSAR software component template*, SystemDesk provides specific dialog pages to specify element properties. For all the other elements, SystemDesk provides generic dialog pages to specify element properties. Refer to Basics on Working with Elements (🕮 SystemDesk Manual).

SystemDesk provides two diagram types for the graphical modeling of compositions and software components.

- Composition diagram: for modeling software architectures by using software components.
- Component diagram: for modeling software components, and their ports and interfaces.

Refer to Basics on Working with Diagrams (🕮 SystemDesk Manual).

Modeling systems

A system is a combination of software components that represent a software architecture, a number of ECUs, and a network topology. It also contains mapping information such as the distribution of software components to ECUs.

SystemDesk provides support for modeling systems and allows you to export system extracts for exchange with basic software configuration and generation tools. Refer to Modeling Systems (🕮 SystemDesk Manual).

Configuring ECUs

Configuring an ECU means configuring the run-time environment (RTE) and the ECU's basic software. An ECU configuration contains all the required information for RTE and basic software code generation and for building the ECU software in an executable HEX file.

SystemDesk provides an ECU configuration framework that you can use to configure ECUs with RTE and basic software modules of any vendor.

You can auto configure selected basic software components that are supported for virtual validation, such as the operating system and the communication stack. During auto configuration, SystemDesk takes the service needs of application software and the dependencies between basic software into account.

You can then generate code for the RTE and the basic software components that are supported for virtual validation, such as the operating system and the communication stack.

Refer to Basics on SystemDesk's ECU Configuration Framework (🕮 SystemDesk Manual).

Focus on creating V-ECUs for virtual validation

Virtual validation lets you verify and test ECU functions and ECUs by means of virtual ECU models and models for their environment. You can use virtual ECUs (V-ECUs) in simulation scenarios throughout the entire ECU development process. A V-ECU implementation comprises the source code files, variable descriptions, etc. that are required to build a V-ECU.

SystemDesk supports two kinds of V-ECUs according to the AUTOSAR Classic Platform:

- *Model-based V-ECUs* based on ECUs in an AUTOSAR system that are modeled in or imported to SystemDesk and their related ECU configurations.
- *Code-based V-ECUs* based on external AUTOSAR or non-AUTOSAR code files.

With SystemDesk, you can export V-ECU implementation containers for integration into simulations for virtual validation. Refer to Basics on Virtual Validation (🕮 SystemDesk Manual).

V-ECUs according to the Adaptive Platform SystemDesk also supports V-ECUs for validating dynamically configured ECUs according to the Adaptive Platform. You can export V-ECU implementations for these adaptive V-ECUs for simulation with VEOS. Refer to Adaptive Platform Support (🕮 SystemDesk Manual).

Interoperability with other AUTOSAR-compliant tools

SystemDesk can be integrated into an AUTOSAR tool chain:

- You can export software components from SystemDesk for behavior modeling in TargetLink.
- You can export a system extract from SystemDesk for further use in basic software configuration tools.

Refer to Integration of SystemDesk into a Tool Chain (🕮 SystemDesk Manual).

Validation of elements

SystemDesk provides validation rules to check whether the elements in a project comply with specified criteria. You can also specify your own validation rules, for example, to check element-naming conventions.

In addition SystemDesk provides validation rule configurations for use cases such as exchanging data with dSPACE TargetLink or EB tresos Studio. You can also save and edit rule configurations of your own for use cases such as modeling software components. Refer to Validating SystemDesk Elements (🕮 SystemDesk Manual).

Automation interface

SystemDesk lets you automate most of its features via its automation interface. Refer to Automating SystemDesk (🕮 SystemDesk Manual).

SystemDesk modules

SystemDesk is intended for users who model ECU software, i.e., software architects and users who validate and test V-ECUs.

SystemDesk's modules reflect the two user groups by offering two different sets of SystemDesk features. Each module will let you use one set of features, while features only available in the other set are disabled. Some core features such as importing and exporting AUTOSAR files or validating model consistency are provided with both feature sets. You can use all of SystemDesk's features only if both modules are available.

Module	Enabled Set of Features[1)]
Modeling	▪ Modeling ECU software according to the AUTOSAR Classic Platform, e.g., modeling a software architecture and a system using SystemDesk's modeling features ▪ Specifying ECU configurations
V-ECU Generation	Preparing ECUs to generate V-ECU implementations for virtual validation.

1) The other set of features is disabled unless both modules are available.

How to Start SystemDesk

Objective

To start SystemDesk.

Preconditions

SystemDesk is installed.

Method

To start SystemDesk

1 On the Start menu, select Programs – dSPACE SystemDesk 5.5, and click SystemDesk.

SystemDesk opens.

Result

You started SystemDesk.

Tip

The installation process automatically creates a shortcut on the desktop. You can double-click  to start SystemDesk.

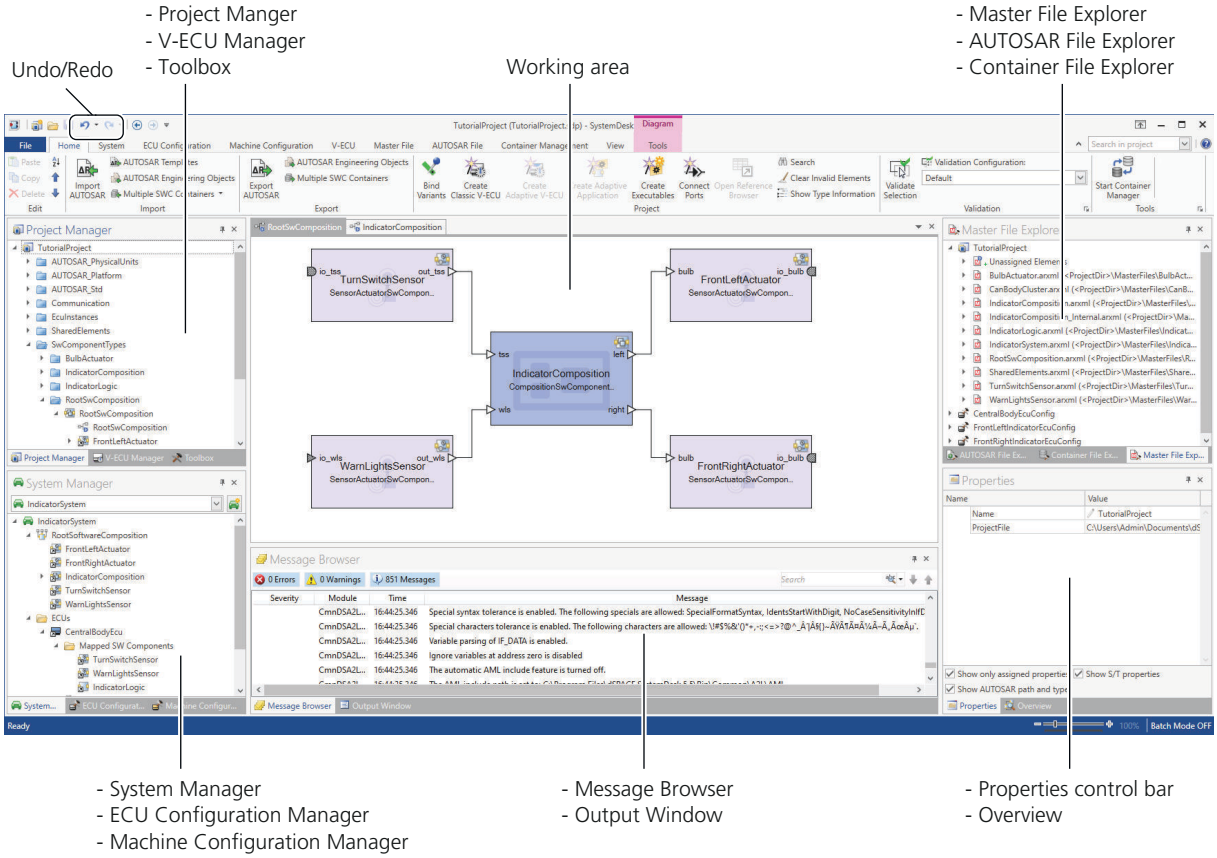
SystemDesk's User Interface

Purpose

Provides an overview of the control bars and areas of SystemDesk's user interface.

User interface

The following illustration shows SystemDesk's user interface.



Control bars and areas

SystemDesk provides the following control bars and areas (listed in alphabetical order):

AUTOSAR File Explorer A SystemDesk control bar that lets you configure import and export of AUTOSAR files for data exchange.

Container File Explorer A SystemDesk control bar that lets you configure SWC containers for data exchange with TargetLink.

ECU Configuration Manager A SystemDesk control bar to configure the run-time environment and the basic software of an ECU.

Machine Configuration Manager A SystemDesk control bar to configure an AUTOSAR model for an adaptive V-ECU.

Message Browser A SystemDesk control bar that provides a history of all the error and warning messages that occur during work with SystemDesk. The Message Browser provides quick access to the files, folders and elements related to a message.

Master File Explorer A SystemDesk control bar that lets you assign the AUTOSAR elements of a SystemDesk project to AUTOSAR master files and read/write the master files.

Output Window A SystemDesk control bar that can be used to display the standard outputs of custom tools that are integrated in SystemDesk.

Overview A SystemDesk control bar that provides a reduced-size view of the currently active diagram.

Project Manager A SystemDesk control bar that provides access to a SystemDesk project and all the elements belonging to it. It displays all the elements of the project hierarchically, according to the AUTOSAR package structure.

Properties control bar A SystemDesk control bar that lets you specify all the properties of a selected element.

System Manager A SystemDesk control bar that displays a system and its child elements and offers context menu commands to create, configure, and manage them.

Toolbox A SystemDesk control bar that provides access to all the elements that are required for modeling software components graphically.

Undo/Redo The Undo command lets you restore project states that are made by performing actions such as creating or deleting elements. You can switch between the project states with the Undo and Redo commands.

V-ECU Manager A SystemDesk control bar for specifying V-ECU software and generating V-ECU implementation containers for virtual validation.

Working area A part of SystemDesk's user interface that provides access to SystemDesk diagrams, editors, etc.

Introduction to this Tutorial

Where to go from here

Information in this section

Introduction and User Profiles 19
This tutorial guides you through your first steps with SystemDesk.

Your First SystemDesk Task 20
Your first task is to model the software architecture for a simple Classic Platform system which manages the left and right direction indicators of a vehicle.

Overview of Lessons 22
When working through this tutorial, you should carry out its instructions as you go along.

Basics on Demo Scripts 23
The SystemDesk installation includes demo scripts in which the results of each lesson are stored.

How to Extract Demo Scripts 24
To extract the demo scripts.

How to Run Demo Scripts 26
To run demo scripts.

Introduction and User Profiles

First steps with SystemDesk

This tutorial guides you through your first steps with SystemDesk. It is divided into short lessons reflecting typical procedures in SystemDesk. You should work through these lessons before using the more detailed 🕮 SystemDesk Manual.

What will you learn?

This tutorial is based on the learning-by-doing concept. When you work through it, you learn the things that are typical for your daily work with SystemDesk, such as:

- Setting up a project
- Modeling a software architecture by creating and specifying software components
- Modeling ports and interfaces of software components and connecting them
- Specifying the SWC internal behavior of software components
- Importing and exporting data
- Configuring ECUs for virtual validation
- Creating V-ECU implementations and exporting them in V-ECU implementation containers
- Performing test builds of V-ECU implementations in conjunction with VEOS

Modular tutorial

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. The result of each lesson is stored in ready-made initial demo scripts that you can use as the starting point for the next lesson. For an overview, refer to Basics on Demo Scripts on page 23.

Glossary

The user documentation provides a glossary with brief explanations on the most important expressions used. Refer to Glossary on page 259. The individual lessons contain cross-references to the expressions, for example, Software component.

User profiles

This tutorial is intended for users that model software architectures and create V-ECUs for virtual validation.

It is assumed that you know basics of:

- The AUTOSAR methodology.
- Behavior modeling with TargetLink or a C code development environment.

Knowledge in handling the host PC and the Microsoft Windows operating system is also assumed.

Your First SystemDesk Task

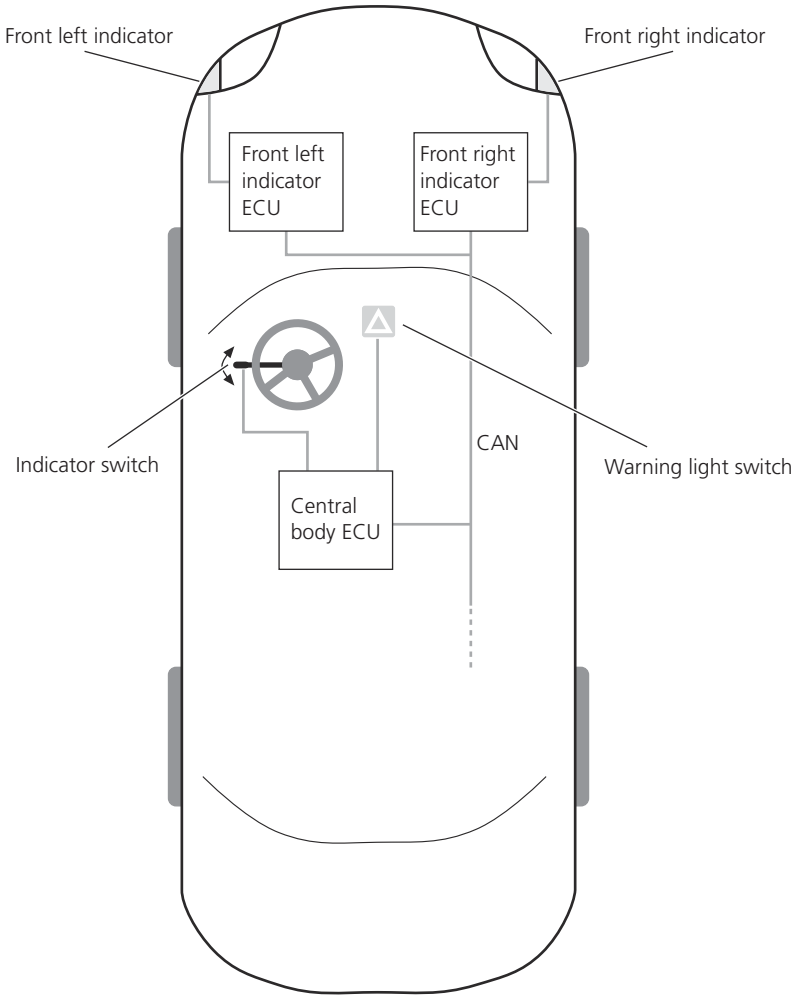
Introduction

Your first task is to model the software architecture for a simple Classic Platform system which manages the left and right direction indicators of a vehicle.

The following illustration shows the simple automotive electric/electronic (E/E) system, consisting of:

- A front left direction indicator.
- A front right direction indicator.
- An indicator switch.
- A warning light switch.
- A front left indicator ECU controlling the front left indicator.
- A front right indicator ECU controlling the front right indicator.
- A central body ECU controlling the status of the indicator switch and the warning light switch, and processing and toggling the indicator status.

For the sake of clarity, the rear direction indicators are not included in the system.



Your task is to perform a top-down design of the software architecture, i.e., to structure the software architecture into atomic software components with clearly defined interfaces and create views of the architecture by using hierarchical compositions. You will then define the SWC internal behavior of software

components by creating several runnable entities and defining how they interact. You will also formalize the hardware topology and the network communication.

Overview of Lessons

Introduction

When working through this tutorial, you should carry out its instructions as you go along.

Lessons

The tutorial contains the following lessons:

- Lesson 1: Setting up Your First Project on page 29
  You will create a project containing a package.
- Lesson 2: Modeling and Connecting Software Components on page 39
  You will create all the interacting software components and their interconnections to model a simple software architecture.
- Lesson 3: Modeling Sender Receiver Interfaces of Software Components on page 61
  You will model sender receiver interfaces of software components and use diagrams to assign them to ports of software components.
- Lesson 4: Modeling Client Server Interfaces of Software Components on page 91
  You will model client server interfaces of software components and use diagrams to assign them to ports of software components.
- Lesson 5: Modeling the SWC Internal Behavior of Atomic Software Components on page 115
  You will model the internal structure of a software component by an SWC internal behavior.
- Lesson 6: Modeling Measurements and Calibration Access on page 141
  You will model measurement access and create a calibration parameter.
- Lesson 7: Specifying Data Type Mapping Sets and Constant Specification Mapping Sets on page 153
  You will specify mappings of data types and constant specifications.
- Lesson 8: Exchanging SWC Containers on page 167
  You will exchange software component with TargetLink.
- Lesson 9: Creating and Configuring an ECU Instance on page 177
  You will create ECU instances.
- Lesson 10: Importing Network Communication Elements on page 185
  You will import an AUTOSAR file with network communication elements.
- Lesson 11: Modeling the System on page 189
  You will model a system.

- Lesson 12: Importing and Exporting Data on page 205
  You will write the project elements to AUTOSAR master files and export a system extract.
- Lesson 13: Modeling ECU Configurations and Basic Software Components on page 217
  You will model an ECU configuration and basic software components and generate RTE code for the purpose of simulation.
- Lesson 14: Generating V-ECU Implementations on page 239
  You will create V-ECU implementations and learn how to perform test builds for them. You will also learn how to export V-ECU implementation containers and how to integrate them into a simulation system in VEOS.

Requirements

SystemDesk modules

- If you have only a Modeling module, you can work through this tutorial from lesson 1 to lesson 12.
- If you have only a V-ECU Generation module, you can start this tutorial with lesson 1 and then skip to lesson 12, step 3.

For information on modules, refer to SystemDesk modules on page 14.

VEOS installation In order to perform test builds of V-ECU implementations and to integrate V-ECU implementations into a simulation system for offline simulation, you need a VEOS installation. Alternatively, you can export V-ECU implementation containers to build them later on.

Basics on Demo Scripts

Introduction

The SystemDesk installation includes demo scripts in which the results of each lesson are stored. You can use these scripts as starting points for consecutive lessons. This way you can work through each lesson without having to complete the previous lessons.

Overview of the demo scripts

The following table provides an overview of all the demo scripts included in the SystemDesk installation.

Tutorial Lesson	Associated Script(s)	SystemDesk Modules Required[1)]
Lesson 1	Lesson01_complete.py	Modeling or V-ECU Generation
Lesson 2	Lesson02_complete.py	Modeling
Lesson 3	Lesson03_Step_1_4.py Lesson03_Step_5.py Lesson03_Step_6.py Lesson03_complete.py	Modeling Modeling Modeling Modeling

Tutorial Lesson	Associated Script(s)	SystemDesk Modules Required[1)]
Lesson 4	Lesson04_Step_1_2.py Lesson04_Step_3_6.py Lesson04_complete.py	Modeling Modeling Modeling
Lesson 5	Lesson05_Step_1_3.py Lesson05_Step_4_5.py Lesson05_complete.py	Modeling Modeling Modeling
Lesson 6	Lesson06_complete.py	Modeling
Lesson 7	Lesson07_complete.py	Modeling
Lesson 8	Lesson08_complete.py	Modeling
Lesson 9	Lesson09_complete.py	Modeling
Lesson 10	Lesson10_complete.py	Modeling
Lesson 11	Lesson11_Step_1.py Lesson11_Step_2_4.py Lesson11_complete.py	Modeling Modeling Modeling
Lesson 12	Lesson12_Step_1.py Lesson12_Step_2.py Lesson12_Step_3.py Lesson12_complete.py	Modeling Modeling Modeling or V-ECU Generation Modeling or V-ECU Generation
Lesson 13	Lesson13_Step_1.py Lesson13_Step_2_3.py Lesson13_Step_4_5.py Lesson13_Step_6.py Lesson13_complete.py	V-ECU Generation V-ECU Generation V-ECU Generation V-ECU Generation V-ECU Generation
Lesson 14	Lesson14_Step_1.py Lesson14_Step_2.py[2)] Lesson14_Step_3.py Lesson14_Step_4.py[2)] Lesson14_complete.py	V-ECU Generation V-ECU Generation V-ECU Generation V-ECU Generation V-ECU Generation

1) For more information on the SystemDesk modules, refer to SystemDesk modules on page 14.

2) This step requires a VEOS installation.

How to Extract Demo Scripts

Objective

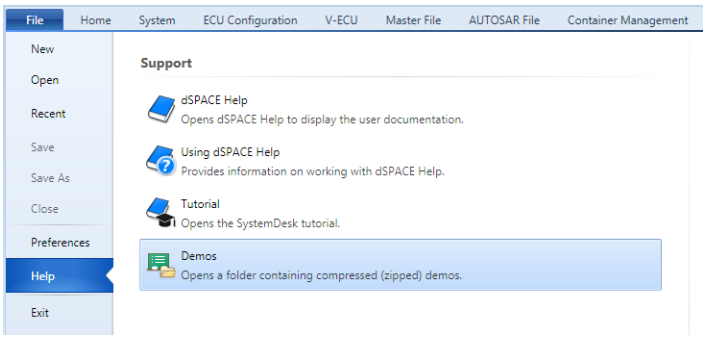
To extract the demo scripts.

Method

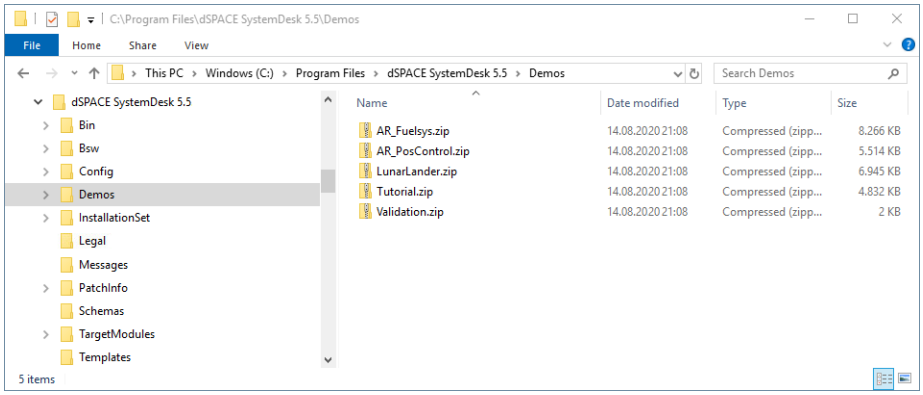
To extract demo scripts

1. Start SystemDesk.
2. On the File ribbon, click Help.

3 On the Help page, click Demos.



4 SystemDesk opens the `Demos` folder of the SystemDesk installation in the file explorer.

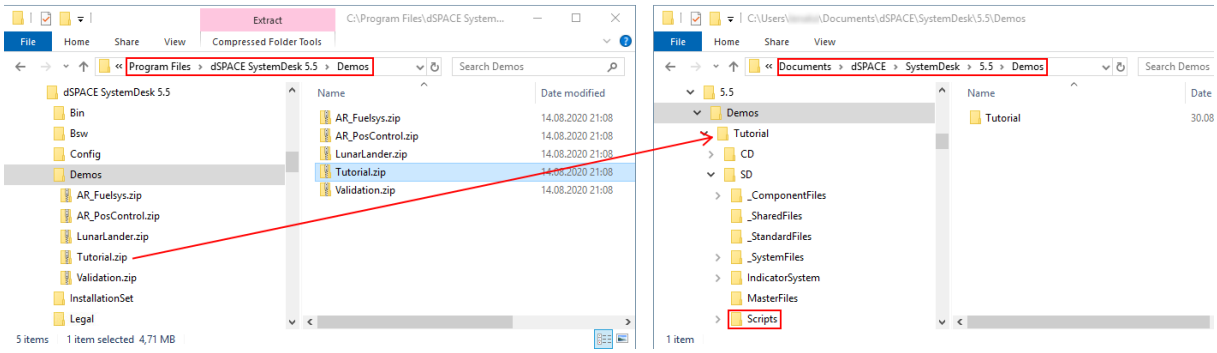


5 Extract the `Tutorial.zip` file to a working folder of your choice, for example, `<My Documents>\dSPACE\SystemDesk\5.5\Demos`.

6 SystemDesk creates the `Tutorial` folder in your working folder. The demo scripts are contained in the subfolder `SD/Scripts`.

Result

You have extracted the demo scripts of the SystemDesk installation to the `Tutorial/SD/Scripts` folder in a working folder of your choice.



Next steps

For instructions on how to run the demo scripts, refer to How to Run Demo Scripts on page 26.

For a list of all demo scripts, refer to Basics on Demo Scripts on page 23.

How to Run Demo Scripts

Objective

To run demo scripts.

Preconditions

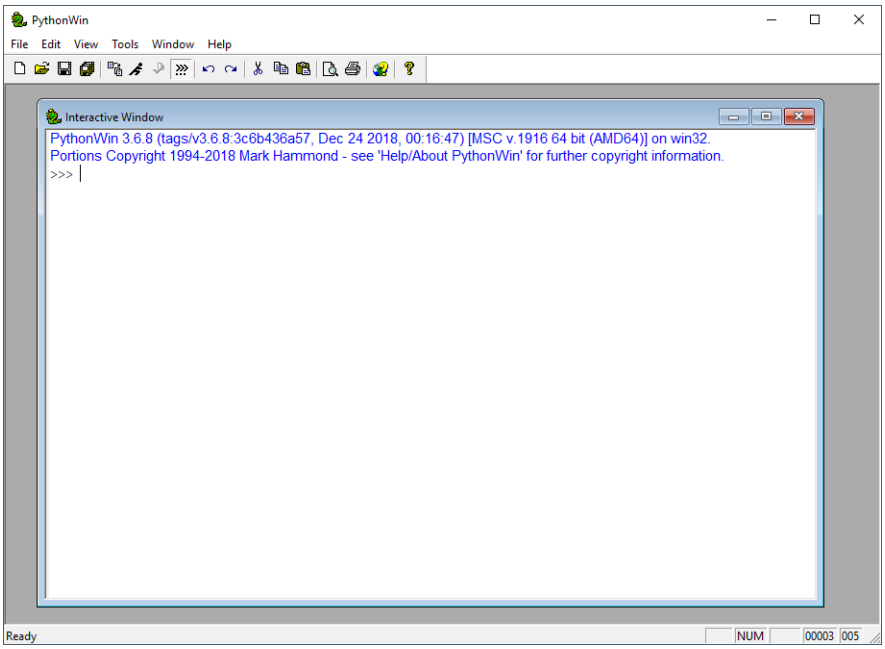
- You have extracted the `Tutorial.zip` file from the SystemDesk installation folder to your working folder as explained in How to Extract Demo Scripts on page 24.
- You need an external Python interpreter to run demo scripts. The SystemDesk installation includes PythonWin. However, you can also use other Python interpreters, such as PyScripter or Microsoft® Visual Studio®.

The following steps are formulated exemplarily for the PythonWin interpreter.

Method

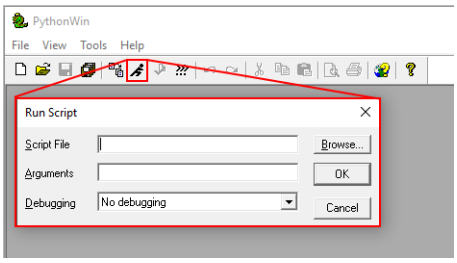
To run demo scripts

1 Start PythonWin.



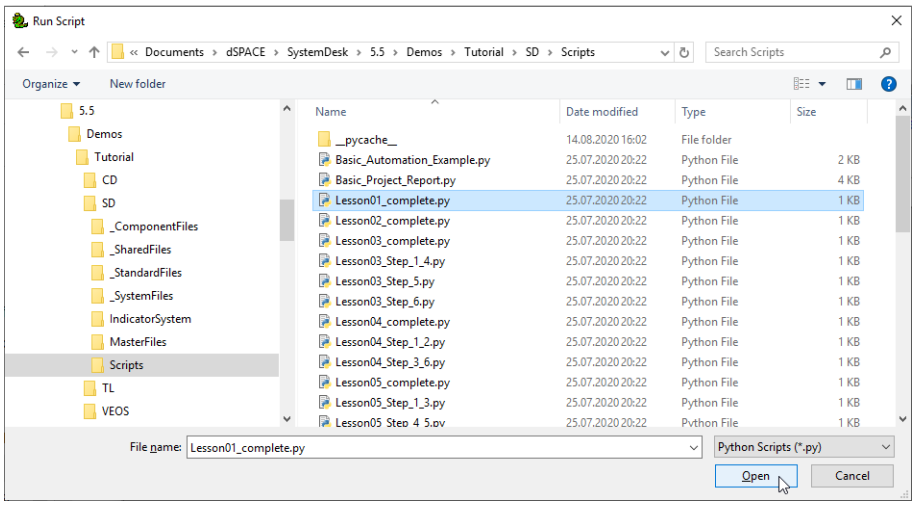
2 From the menu bar, choose File – Run or click .

3 The Run Script dialog opens.



4 In the Run Script dialog, click Browse to search the file system.

5 Go to the `Tutorial\SD\Scripts` folder in your working folder and open the script you wish to run.



6 Click OK to start script execution.

Result

SystemDesk creates a demo project `TutorialProject.sdp` that reproduces the steps indicated by the file name, compare Basics on Demo Scripts on page 23.

Next steps

You can save the SDP file for further use if required. At the beginning of the tutorial, you will learn how to work with projects.

Refer to Lesson 1: Setting up Your First Project on page 29.

Lesson 1: Setting up Your First Project

Where to go from here

Information in this section

Overview of Lesson 1 29
In this lesson, you will learn how to create a structured project containing a package.

Step 1: How to Create a Project 30
You will now create a project as a container for your work.

Step 2: How to Add a Package to the Project 31
Now you will begin to structure the project by using packages.

Step 3: How to Save, Close and Open a Project 32
You will now learn how to save, close, and open your project.

Step 4: How to Import AUTOSAR Templates 33
In this step, you will import AUTOSAR templates to the TutorialProject.

Result of Lesson 1 36
In this lesson, you learned how to work with projects.

Overview of Lesson 1

What will you learn?

A SystemDesk project contains all the parts and information that are required to specify a software architecture which runs on a network of ECUs.

The project is organized in packages. Packages can contain further packages or AR elements like software components, interfaces or compu methods.

Packages logically group your AR elements and establish a namespace for the elements in them. Elements in the same package must therefore have different names. You can build the project's hierarchical structure by adding packages to the project.

You can use the project as the basis for the following lessons.

Before you begin

To understand the principles of SystemDesk and the terminology used, you are recommended to read Introduction to SystemDesk on page 11.

Steps

In this lesson, you will learn how to create a structured project containing a package.

- Step 1: How to Create a Project on page 30
- Step 2: How to Add a Package to the Project on page 31
- Step 3: How to Save, Close and Open a Project on page 32
- Step 4: How to Import AUTOSAR Templates on page 33

Summary

To check if the steps you executed are correct, refer to Result of Lesson 1 on page 36.

Step 1: How to Create a Project

Objective

You will now create a project as a container for your work.

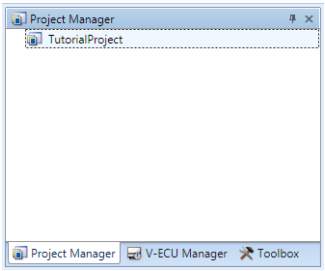
Method

To create a project

1. Start SystemDesk.
2. On the File ribbon, click New to open the New Project dialog.
3. Navigate to the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD` folder.
4. In the File name edit field, change the default name NewProject.sdp to `TutorialProject.sdp`.
5. Click Save.

   A SystemDesk prompt lets you specify whether you want to replace the existing `TutorialProject.sdp`. Click Yes. You can always restore the file from the `Tutorial.zip` archive.

SystemDesk displays the new project as a root element in the project tree in the Project Manager.



Result

You created a new project.

A project (SDP) file was created in the file system.

What's next

You will learn how to add a package to the project in the next step.

Step 2: How to Add a Package to the Project

Objective

The project you created in the previous step will be used throughout the whole tutorial. Now you will begin to structure the project by using packages.

In this step you will add a package named `SwComponentTypes` to your project, which will contain the software components you create throughout the tutorial.

AUTOSAR recommends certain package names for modeling elements, e.g. SwComponentTypes for all kinds of software component types. In step 4 of this lesson, you will import a package structure that uses these names.

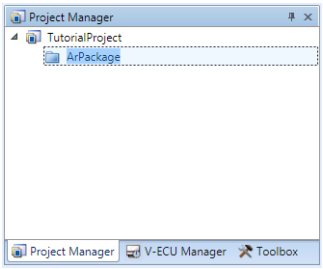
Method

To add a package to the project

1 In the Project Manager, right-click the TutorialProject project to open its context menu.

2 Select **New Package**.

The **Project Manager** displays a new package as a child of the project.

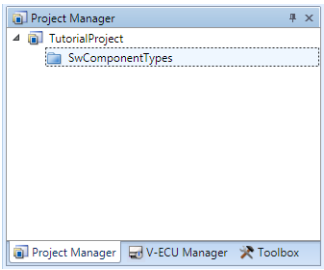


3 Rename the package `SwComponentTypes`.

The **Project Manager** displays the new name.

Result

You added a new package named `SwComponentTypes` (software component types) to the **TutorialProject**.



You will add software components to the package in Lesson 2.

What's next

You will learn how to save, close and open your project in the next step.

Step 3: How to Save, Close and Open a Project

Objective

Having created your first project and a package during the previous steps, you will now learn how to save, close, and open your project.

When it is closed, the project tree will no longer be visible in the **Project Manager**. You will also learn how to open an existing project.

Part 1

To save the project

1 On the **File** ribbon, click **Save**.

Interim result

You saved your project. You can now close it.

Part 2

To close the project

1 On the File ribbon, click Close.
The project and all its elements are closed.

Interim result

You closed your project. You can now open it again.

Part 3

To open an existing project

1 On the File ribbon, click Open.

2 In the Open Project dialog, change to the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD` folder, select the `TutorialProject.sdp` file, and click Open.
The project, including the SwComponentTypes package, is displayed in the Project Manager and can be used for further processing.

Result

You saved, closed and opened your project. You completed the setting up of a project.

What's next

You can now import standard definitions from AUTOSAR and a best practice package structure to the TutorialProject in the next step.

Step 4: How to Import AUTOSAR Templates

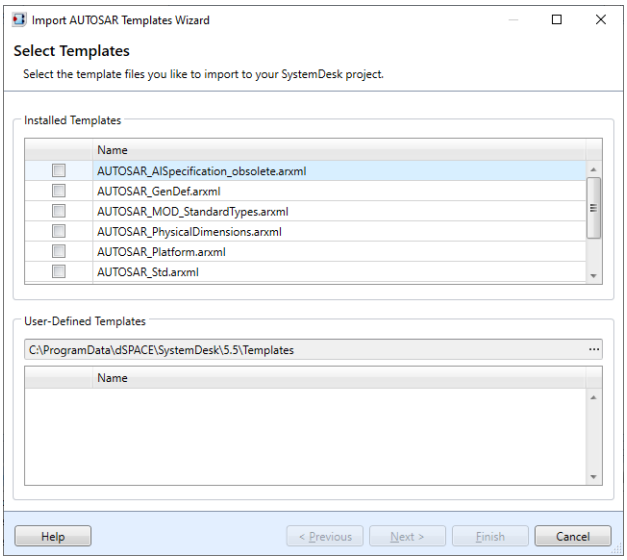
Objective

In this step, you will import AUTOSAR templates to the TutorialProject. This is useful if you want to add specific elements to your project such as definitions by AUTOSAR or a template package structure for your project.

Method

To import AUTOSAR templates

1. In the Project Manager, right-click the TutorialProject node.
2. From the context menu, select Import – AUTOSAR Templates.
   SystemDesk opens the Import AUTOSAR Templates Wizard, as shown in the following illustration.

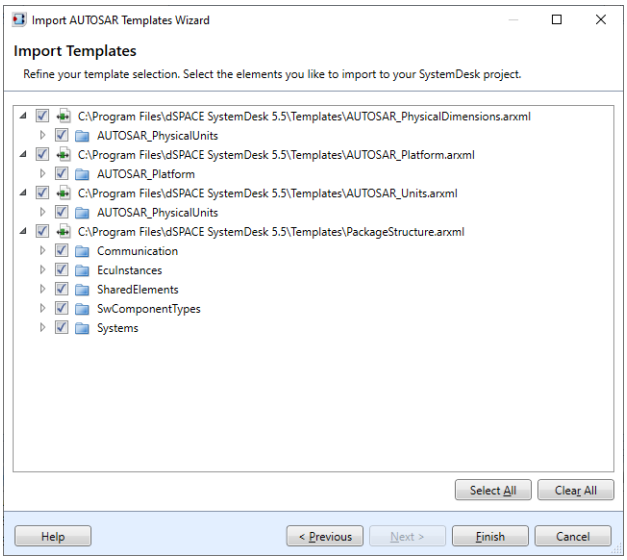


3. Select the templates marked with ✓ in the Select column of the following table that lists descriptions of the templates displayed in the Import AUTOSAR Templates Wizard.

Select	File	Description
-	`AUTOSAR_AISpecification_obsolete.arxml`	Definitions of units and physical dimensions as defined by AUTOSAR prior to the AUTOSAR 19-11 Release. For projects based on AUTOSAR 19-11, use the definitions specified in `AUTOSAR_PhyscialDimensions.arxml` and `AUTOSAR_Units.arxml`.
-	`AUTOSAR_GenDef.arxml`	Additional data types as defined by AUTOSAR, e.g., `void`. These data types are referenced by some standardized BSW service interfaces
-	`AUTOSAR_MOD_StandardTypes.arxml`	Standard implementation data types for the AUTOSAR Adaptive Platform.
✓	`AUTOSAR_PhysicalDimensions.arxml`	Definitions of physical dimensions used by AUTOSAR. You can use the physical dimensions as the basis for unit definitions in the TutorialProject.
✓	`AUTOSAR_Platform.arxml`	SW base type definitions and platform types as defined by AUTOSAR. You can use the definitions to specify data types.

Select	File	Description
-	`AUTOSAR_Std.arxml`	Additional data types as defined by AUTOSAR, e.g., `Std_ReturnType`. These data types are required for operations in client server interfaces and BSW APIs.
✓	`AUTOSAR_Units.arxml`	Definitions of physical units used by AUTOSAR. You can use the definitions to specify the units of physical values in the TutorialProject.
✓	`PackageStructure.arxml`	Best practice structure of packages to exchange data easily.
-	`TargetLink.TLDataTypes.armxl`	Definitions of data types used in SWCs generated by TargetLink. You can use the definitions to specify data types of SWC with code files generated by TargetLink.

4 Click Next to go to the wizard's second page.

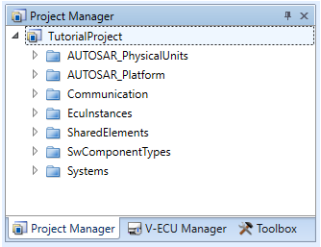


5 On the page, click Select All.

6 Click Finish.

Result

You imported AUTOSAR templates to your project for structuring it and using standardized types.



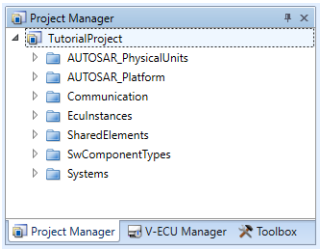
What's next

Now you can compare your work with the result you should have achieved. Refer to Result of Lesson 1.

Result of Lesson 1

Summary

If you have done all the steps in this lesson, your SystemDesk project should look like this:



In this lesson, you learned how to work with projects. You also learned how to add packages to your project to structure its elements.

Sorting items When you create or open a project, SystemDesk sorts all items in the **Project Manager** alphabetically. The demo scripts also do this, although following the instructions in the tutorial will sometimes lead to a different sorting.

You can reproduce the alphabetical sorting by selecting the `TutorialProject` node in the **Project Manager** and clicking the icon on the **Home – Edit** ribbon group. You can also move items individually by first selecting them and then using the and icons on the **Home – Edit** ribbon group.

Demo

You can reproduce the result of this lesson with the demo script `Lesson01_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information

For a detailed description of creating projects, refer to Working with Projects (📖 SystemDesk Manual).

What's next

Note

If you have only SystemDesk's V-ECU Generation module, you must skip to lesson 12, step 3 (refer to Step 3: How to Import AUTOSAR Master Files on page 214).

The next lesson shows you how to model the software architecture by creating software components for the direction indicator system.

Lesson 2: Modeling and Connecting Software Components

Where to go from here

Information in this section

Overview of Lesson 2 40
In this lesson, you will learn how to model different types of software components, structure them hierarchically, and connect them.

Step 1: How to Create Software Components 41
In this step, you will create three sensor actuator SW component types, which represent the link to the sensor/actuator hardware.

Step 2: How to Create and Open Composition Diagrams 43
You will now create composition diagrams.

Step 3: How to Add Software Components to a Composition SW Component Type 44
In this step, you will create prototypes of related software component types.

Step 4: How to Customize Composition Diagrams 47
You will now change the default settings of the diagram elements and customize the views.

Step 5: How to Add Ports to Software Components 51
You will now learn how to add ports to software components.

Step 6: How to Connect Software Components 54
You will now connect the ports in a composition diagram by using connectors to allow data exchange between SWCs.

Step 7: How to Connect a Composition SW Component Type to Its Inner Software Components 57
You will connect a composition to its inner software components via delegation connectors.

Result of Lesson 2 59
In this lesson, you learned how to model the software architecture consisting of compositions and atomic software components which are connected via their ports.

Overview of Lesson 2

Starting point

In lesson 1, you learned how to set up your project.

What will you learn?

A software component logically groups and encapsulates single functionalities which are to be implemented on an ECU. Software components communicate with each other via connected ports. Interfaces assigned to the ports define which data or control flow is exchanged and how.

In this lesson, you will learn how to model different types of software components, structure them hierarchically, and connect them.

This tutorial will familiarize you with the following software components:

Composition SW component type A composition SW component type has no functionality of its own and is only a structural element that contains other software components. Its ports and interfaces define the communication with software components outside of the composition, such that a change in software components contained in a composition does not affect the rest of the system.

Atomic software component The term atomic software component is a generic term for all software components which, in contrast to a composition SW component type, represent functionality of their own. SWC internal behaviors can therefore be assigned only to atomic software components. You will use atomic SWCs such as:

- *Application SW component type*
- *Sensor actuator SW component type*

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson01_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create Software Components on page 41
- Step 2: How to Create and Open Composition Diagrams on page 43
- Step 3: How to Add Software Components to a Composition SW Component Type on page 44
- Step 4: How to Customize Composition Diagrams on page 47
- Step 5: How to Add Ports to Software Components on page 51
- Step 6: How to Connect Software Components on page 54
- Step 7: How to Connect a Composition SW Component Type to Its Inner Software Components on page 57

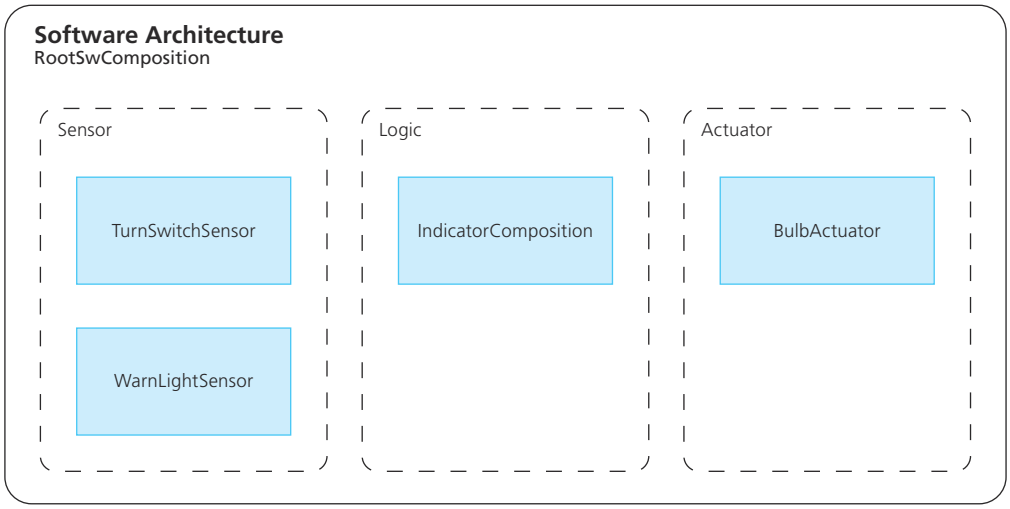
Summary

To check if the steps you executed are correct, refer to Result of Lesson 2 on page 59.

Step 1: How to Create Software Components

Objective

In this step, you will create three sensor actuator SW component types, which represent the link to the sensor/actuator hardware.



The sensor software components read the sensor data and check for errors before relaying the data to a software component which encapsulates the logic of the system. The actuator software component activates the bulb on receiving the respective command.

To encapsulate the logic of the direction indicator system, you will create a composition named **IndicatorComposition**. This means the architecture of the software components representing the logic can be changed without affecting the sensor actuator SW component types.

Finally, you will create another composition called RootSwComposition, which will contain the whole architecture with all software components.

Part 1

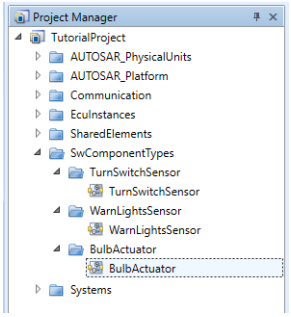
To create sensor actuator SW component types

1. In the Project Manager, right-click the SwComponentTypes/MySwc package.
2. From the context menu, select Delete.

   The MySwc package, which was imported with the `PackageStructure.arxml` template, is deleted from the project.
3. Right-click the SwComponentTypes package.
4. From the context menu, select New – Package and name it `TurnSwitchSensor`.
5. Right-click the TurnSwitchSensor package.
6. From the context menu, select New – New SWC-T – Sensor Actuator SW Component Type.

   The Project Manager displays a new sensor actuator SW component type as a child of the TurnSwitchSensor package.
7. Change the software component's name to `TurnSwitchSensor`.
8. Repeat steps 3 … 6 to create two more packages, each containing a sensor actuator SW component type of the same name.

   Name them `WarnLightsSensor` and `BulbActuator`.



Interim Result

You created three sensor actuator SW component types: TurnSwitchSensor, WarnLightsSensor, and BulbActuator.

Part 2

To create composition SW component types

1. In the Project Manager, right-click the SwComponentTypes package.
2. From the context menu, select New – Package and name it `IndicatorComposition`.
3. Right-click the IndicatorComposition package.
4. From the context menu, select New – Composition SW Component Type.

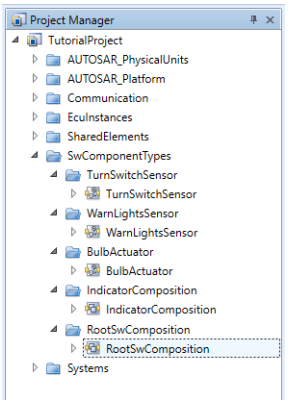
The Project Manager displays a new composition as a child of the IndicatorComposition package.

5 Change the composition's name to `IndicatorComposition`.

6 Repeat steps 1 … 4 to create another package containing a composition, and name both `RootSwComposition`.

Result

You created three sensor actuator SW component types, five packages and two empty compositions. RootSwComposition represents the software architecture.



What's next

You will create and open composition diagrams as graphical tools for modeling and visualizing compositions in the next step.

Step 2: How to Create and Open Composition Diagrams

Objective

A composition diagram allows you to specify your compositions by modeling and connecting software components graphically.

A composition diagram does not always have to show the entire software architecture, and you can define various diagrams for the same model. A composition SW component type can be shown in different composition diagrams, each representing a different aspect. This separates the model from the view and helps you to reduce the complexity of large models.

You will now create composition diagrams as the basis and inner view of the RootSwComposition and the IndicatorComposition, and then open the first one.

Method

To create a composition diagram

1 Right-click the RootSwComposition composition in the Project Manager.

2 From the context menu, select New – Empty Composition Diagram.

The **Project Manager** displays a new composition diagram as a child of **RootSwComposition**.

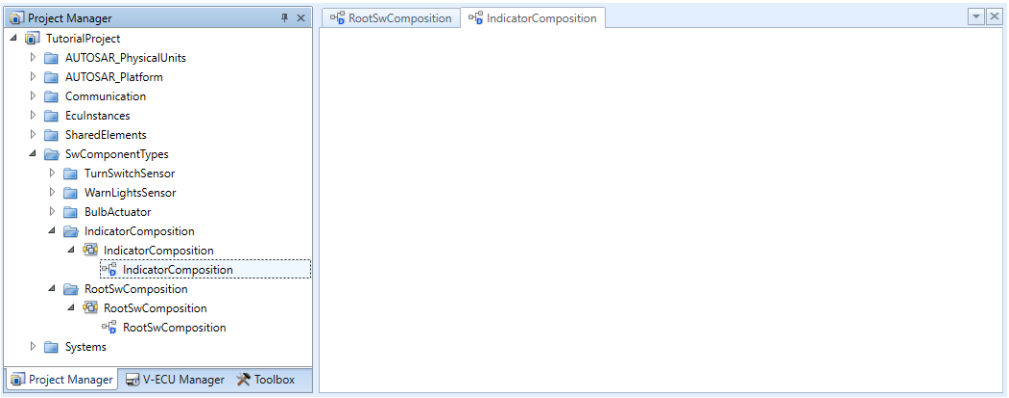
The diagram opens in the working area.

3 Repeat steps 1 … 2 for the **IndicatorComposition** composition.

The **IndicatorComposition** diagram opens in a new tab in the working area.

Result

You created two composition diagrams, **IndicatorComposition** and **RootSwComposition**, which are now open in separate tabs in the working area.



What's next

You can add software components to the compositions by using the composition diagrams in the next step.

Step 3: How to Add Software Components to a Composition SW Component Type

Objective

According to AUTOSAR, you can create *types* and *prototypes* for elements such as software components.

For software components, a *type* defines the element's properties and child elements such as ports or an SWC internal behavior.

A *prototype* describes the usage of an element as a part of another type, e.g., the instance of a software component in a composition SWC. It has all the definitions of a type and additional properties of its own such as a short name. This allows you to reuse elements.

In this step, you will model the **RootSwComposition** and add prototypes of the software component types you defined in the previous step:

- One prototype of **TurnSwitchSensor**, which will process the input from the indicator switch. Refer to Part 1 on page 45.

- One prototype of WarnLightsSensor, which will process the input from the warning light switch. Refer to Part 1 on page 45.
- Two prototypes of BulbActuator, which will control the left and right indicators. Refer to Part 2 on page 46.

You can start modeling a software architecture without defining the inner structure of a composition SW component type. You can then define the composition SW component type's inner software components.

To model the inner structure of IndicatorComposition, you will also:

- Create the IndicatorLogic atomic SWC and a prototype of it. Refer to Part 3 on page 46.

Part 1

To add prototypes of software components

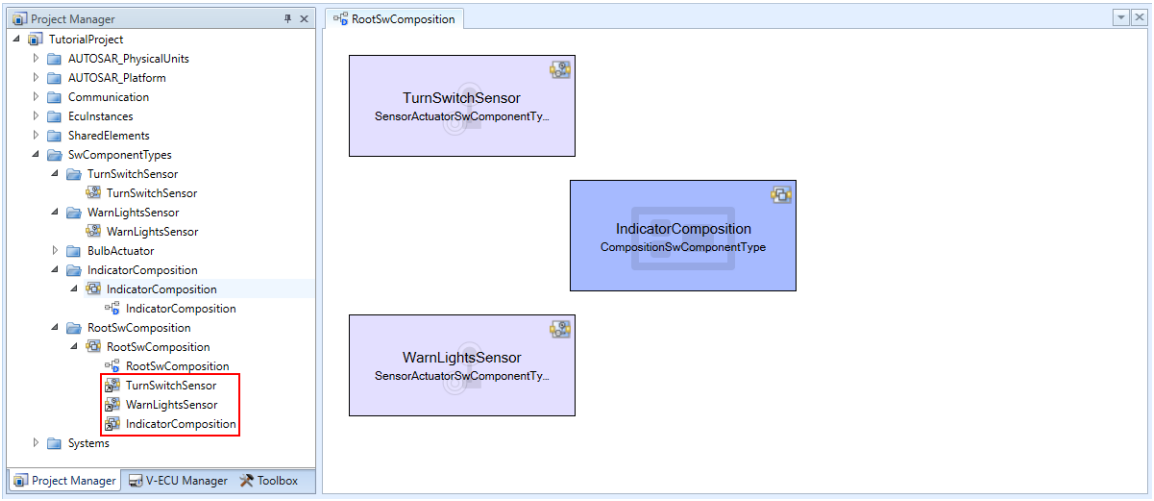
1 In the Project Manager, select the TurnSwitchSensor SWC and drag it to the opened RootSwComposition diagram.

SystemDesk displays a prototype of the TurnSwitchSensor SWC in the RootSwComposition diagram and as a child of the RootSwComposition in the Project Manager.

2 Repeat step 1 for the WarnLightsSensor and the IndicatorComposition SWCs.

Note

You can recognize prototypes in the Project Manager by the additional symbol, which indicates that the element type is defined somewhere else in the Project Manager. To find its definition, right-click the element and select Locate - Type in Project Manager. SystemDesk highlights the type definition in the Project Manager.



Interim Result

You have added prototypes of software components to RootSwComposition.

The BulbActuator SWC encapsulates the functionality to address one indicator bulb. Since two indicator bulbs, the front left and the front right, need to be addressed, you will use two prototypes of this SWC.

Now you will add two prototypes of the BulbActuator SWC to RootSwComposition.

Part 2

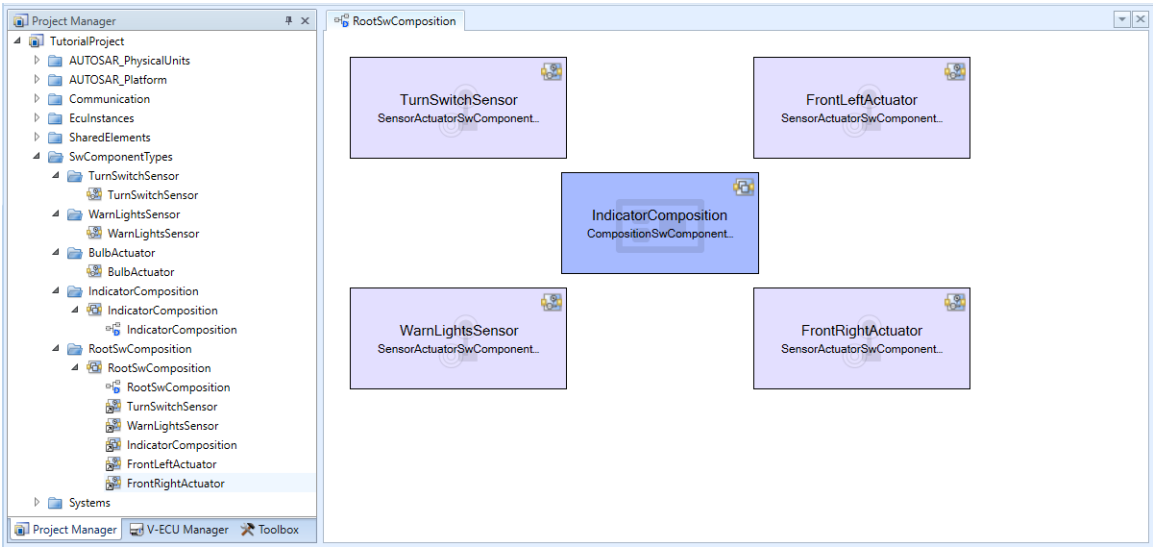
To add multiple prototypes of a software component

1. In the Project Manager, select the BulbActuator SWC and drag it to the opened RootSwComposition diagram.

   SystemDesk displays a prototype of the BulbActuator SWC in the RootSwComposition diagram and as a child of RootSwComposition in the Project Manager.
2. Rename the prototype of the BulbActuator SWC `FrontLeftActuator`.
3. Drag a second prototype of the BulbActuator SWC to the RootSwComposition diagram and rename it `FrontRightActuator`.

Interim Result

You added two prototypes of the BulbActuator SWC to RootSwComposition.



You will now create one SWC which will contain the logic and add it to IndicatorComposition.

Part 3

To create a software component via the composition diagram

1. In the Project Manager, create a new package in the SwComponentTypes package and name it `IndicatorLogic`.
2. Double-click the IndicatorComposition composition diagram in the Project Manager to open it in the working area.

3 In the working area, right-click the IndicatorComposition composition diagram to open its context menu.

4 From the context menu, select New – Application SW Component Type.

SystemDesk displays a prototype of the SWC in the IndicatorComposition composition diagram and as a child of IndicatorComposition in the Project Manager.

The application SW component type is added to the IndicatorComposition package.

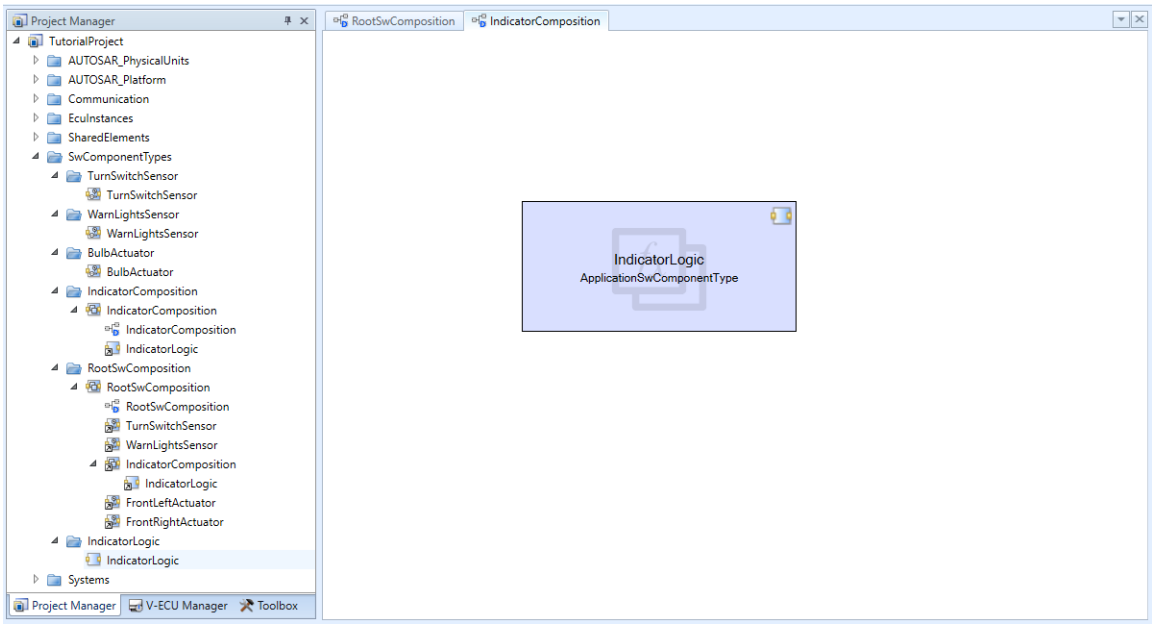
5 Rename the application SW component type `IndicatorLogic`.

6 Rename the application SW component prototype `IndicatorLogic`.

7 Click the IndicatorLogic SWC and move it to the IndicatorLogic package.

Result

You added all the required prototypes of software components to RootSwComposition and IndicatorComposition. The prototypes are visualized in the respective composition diagrams.



What's next

You can now customize your composition diagrams, i.e., move, resize and arrange diagram elements and change the fill color, in the next step.

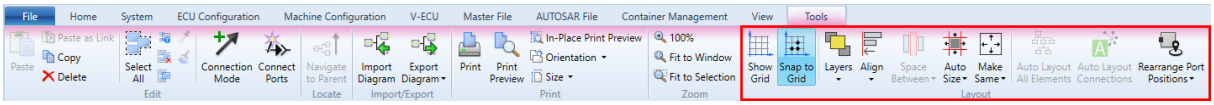
Step 4: How to Customize Composition Diagrams

Objective

In the previous steps, you worked with the composition diagram.

You will now change the default settings of the diagram elements and customize the views.

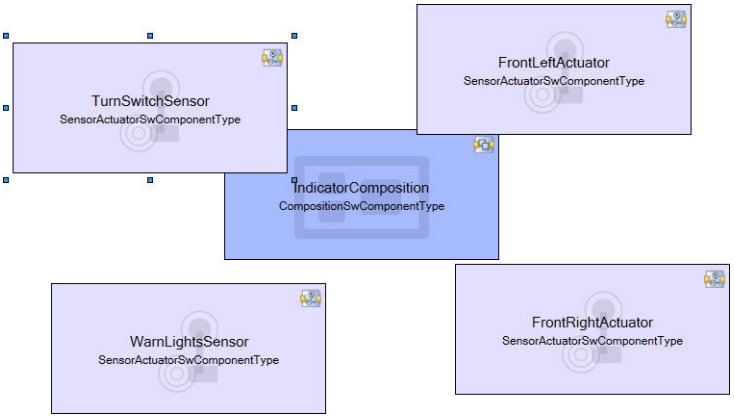
Tools (Diagram context) – Layout ribbon group SystemDesk provides the Tools (Diagram context) – Layout ribbon group for you to easily align, arrange and resize the elements of a diagram. In this lesson, you will familiarize yourself with the ribbon group by using some of its commands.



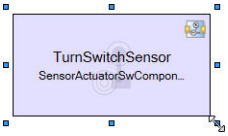
Part 1

To resize diagram elements and make them the same size

1 In the RootSwComposition diagram, select the TurnSwitchSensor SWC.
The SWC is marked by blue handles.

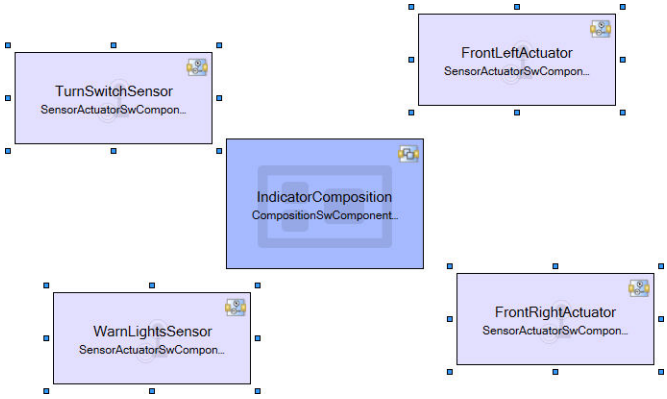


2 Position the mouse pointer on a handle and move the SWC to the required size while keeping the mouse pressed.



The TurnSwitchSensor SWC is resized.

3 To make diagram elements the same size, press **Ctrl**, select the other elements one after another, for example, all the SWCs, and click Layout – Make Same - Size on the Tools (Diagram context) ribbon.



Interim result

You resized the SWCs in the composition diagram. You can now arrange them.

Part 2

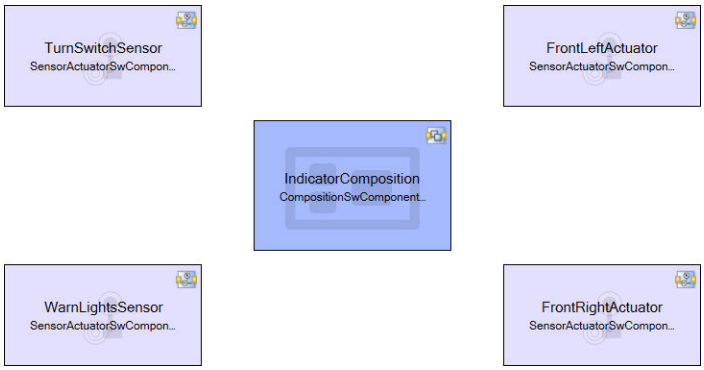
To arrange diagram elements

1. In the RootSwComposition diagram, click the TurnSwitchSensor SWC and move it to the top left corner of the diagram.
2. To align the TurnSwitchSensor and the WarnLightsSensor SWCs by their left boundaries, press **Ctrl**, select the TurnSwitchSensor and the WarnLightsSensor SWC and click Layout – Align – Left on the Tools (Diagram context) ribbon.

   The two SWCs are aligned.
3. Arrange the remaining SWCs as follows:
   - Align the TurnSwitchSensor and the FrontLeftActuator SWCs by their top boundaries by selecting both and clicking Layout – Align – Top on the Tools (Diagram context) ribbon.
   - Align the FrontLeftActuator and the FrontRightActuator SWCs by their right boundaries by selecting both and clicking Layout – Align – Right on the Tools (Diagram context) ribbon.
   - Align the WarnLightsSensor and FrontRightActuator SWCs by their bottom boundaries by selecting both and clicking Layout – Align – Bottom on the Tools (Diagram context) ribbon.
4. Resize the IndicatorComposition and place it in the middle of the SWCs.

Interim result

You arranged the elements in the composition diagram.

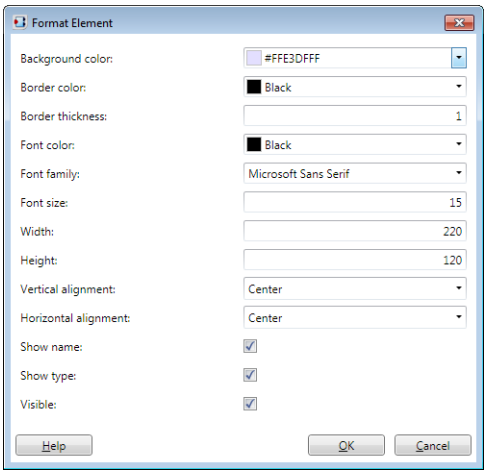


You can now change their colors.

Part 3

To change the color of diagram elements

1. In the RootSwComposition diagram, right-click the TurnSwitchSensor SWC.
2. From its context menu, select Format Element.
   The Format Element dialog opens.



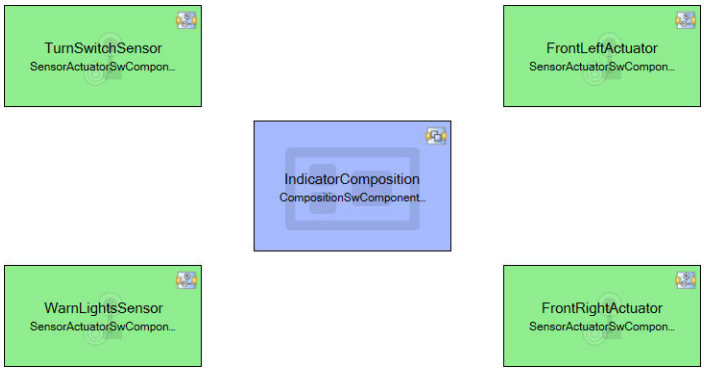
3. Select a Background color.
4. Click OK to close the Format Element dialog.
   The background color of the TurnSwitchSensor SWC changes.
5. Right-click the TurnSwitchSensor SWC.
6. From its context menu, select Copy Format.
7. Multiselect the WarnlightsSensor, the FrontLeftActuator, and the FrontRightActuator SWCs.
8. Right-click the selected SWCs.

9 From the context menu, select **Apply Format**.

The background color of the selected SWCs changes to the color of the **TurnSwitchSensor** SWC.

Result

You customized your **RootSwComposition** diagram (see the following example).



Tip

The **Diagrams** page of SystemDesk's **Preference** dialog lets you specify general settings for diagrams, such as the border thickness of elements. The changes of the general settings are applied to new elements.

What's next

You can add ports to the software components to define interconnection points between software components in the next step.

Step 5: How to Add Ports to Software Components

Objective

Having created the software components, you will now learn how to add ports to them.

Ports are the interconnection points between software components. Ports of different SWCs can be connected to indicate a data flow between them. The details, i.e., the kind of information that is actually transported between two ports, are defined by interfaces. Each port has an interface. Connected ports must have compatible interfaces.

There are two types of ports that indicate the direction of communication:

PPort Prototype The software component provides services or data to other software components using a provide port.

RPort Prototype The software component receives or uses services or data from other software components via a require port.

In this step, you will add ports to all the SWCs in RootSwComposition to connect them, i.e., to connect IndicatorComposition with its surrounding atomic SWCs.

Part 1

To add ports to an SWC via the Project Manager

1. In the Project Manager, select the TurnSwitchSensor SWC type.
2. From the context menu, select New – Provided Port Prototype.
   The Project Manager displays a provide port as a child of the SWC.
3. Change the port's name to `out_tss`.
4. Repeat steps 2 … 3 to add the following ports to the TurnSwitchSensor, the WarnLightsSensor and the BulbActuator SWCs.

Atomic SWC	Port Prototype	Short Name
TurnSwitchSensor	Required	`io_tss`
TurnSwitchSensor	Provided	`out_tss`
WarnLightsSensor	Provided	`out_wls`
WarnLightsSensor	Required	`io_wls`
BulbActuator	Required	`bulb`
BulbActuator	Required	`io_bulb`

Interim result

You added ports to the atomic software components of RootSwComposition.

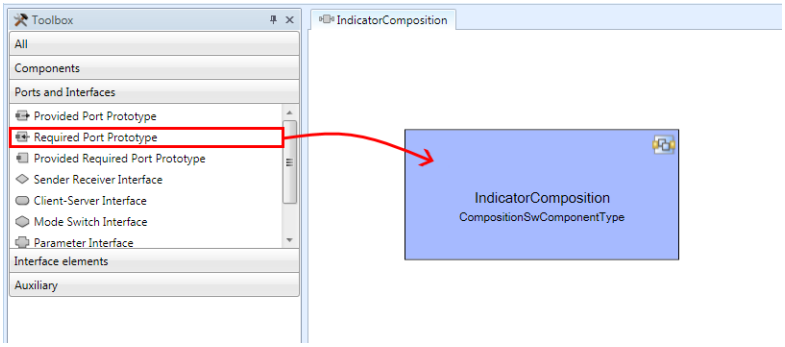
You can now add ports to IndicatorComposition.

Part 2

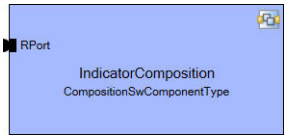
To add ports to an SWC via the Component Diagram

1. In the Project Manager, select the IndicatorComposition SWC.
2. From the context menu, select Open Component Diagram.
   The IndicatorComposition's component diagram opens.
3. Open the Toolbox.

4 From Ports and Interfaces, select Required Port Prototype, and drag it onto IndicatorComposition in the component diagram.

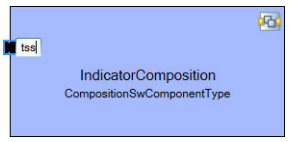


The port is added to IndicatorComposition in the component diagram and in the Project Manager.



5 Right-click the RPort in the diagram and select Rename from its context menu.

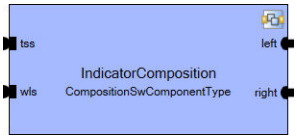
6 Type `tss` in the text field.



You renamed the port tss.

7 Repeat steps 3 … 6 to add the following ports to IndicatorComposition:

Port Prototype	Short Name
Required	`wls`
Provided	`left`
Provided	`right`

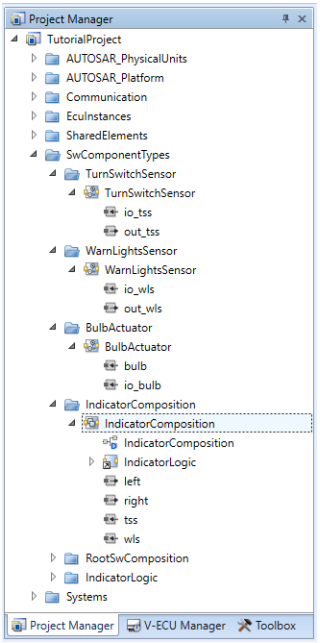


Result

You added ports to IndicatorComposition. They will be the counterpart to the ports of the SWCs in the RootSwComposition created in Part 1.

Tip

Instead of the component diagram, you can also use the composition diagram for steps 4 to 7 of Part 2.



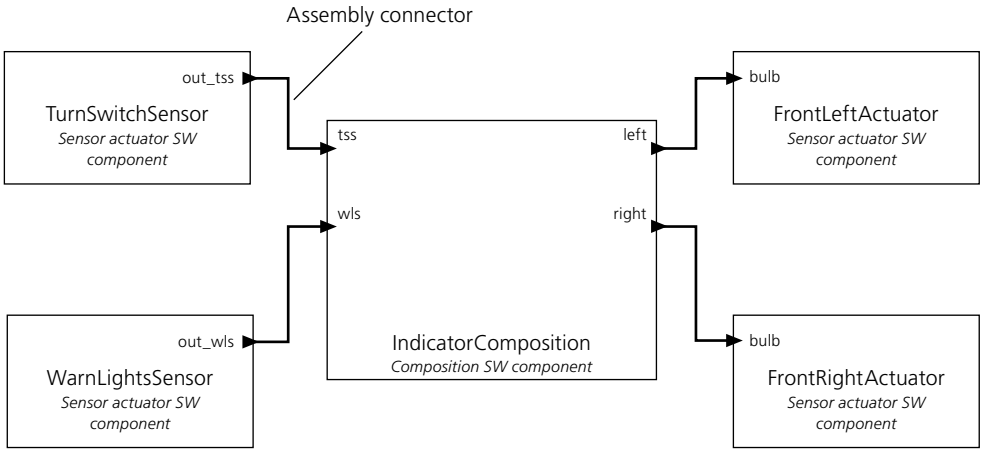
What's next

You can now connect the software components by using their ports in the next step.

Step 6: How to Connect Software Components

Objective

In the previous step, you added ports to the software components of RootSwComposition. You will now connect the ports in the RootSwComposition composition diagram by using connectors to allow data exchange between the SWCs. The following figure shows how to connect the ports by assembly connectors.

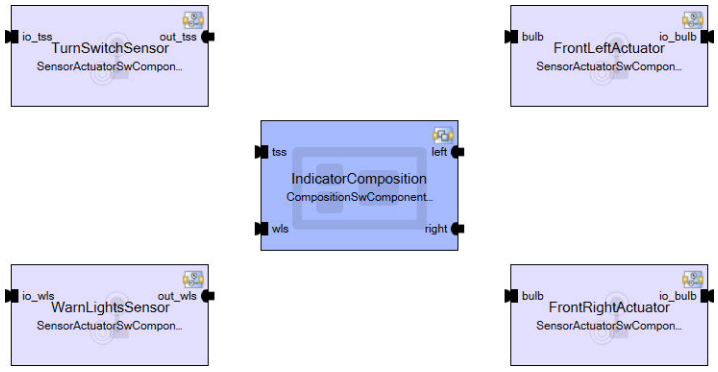


First you will move the ports in the composition diagram to arrange them more clearly.

Part 1

To move ports in a composition diagram

1. Open the RootSwComposition composition diagram.
2. Left-click the out port of the TurnSwitchSensor.
3. Move the selected port to the required position while keeping the left mouse button pressed.
4. Repeat steps 2 … 3 for all the other ports to arrange them as shown below.



Tip

You can also move multiple ports:

1. Multi-select ports by drawing a rectangle around them.
2. Drag one port vertically or horizontally by pressing the **Shift** key and the left mouse button. The other port(s) are moved in the same direction.

Interim result

You arranged the ports by moving them in the composition diagram. You will now connect the ports.

Part 2

To connect software components

1 On the Tools (Diagram context) ribbon, click Edit – Connection Mode to enable the connection mode.

The mouse pointer changes to  .

2 Click the out_tss port of the TurnSwitchSensor SWC and drag the assembly connector to the tss port of IndicatorComposition while keeping the left mouse button pressed.

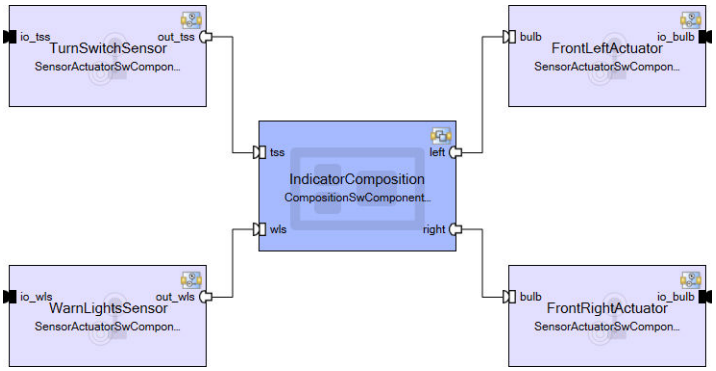
The two ports are connected.

3 Connect the remaining ports as follows:

- out_wls (WarnLightsSensor SWC) → wls (IndicatorComposition)
- left (IndicatorComposition) → bulb (FrontLeftActuator SWC)
- right (IndicatorComposition) → bulb (FrontRightActuator SWC)

Result

You connected the software components in the RootSwComposition composition diagram via their ports.



Tip

When you are working in the edit mode, you can also temporarily change to the connection mode by pressing the **Shift**-key.

What's next

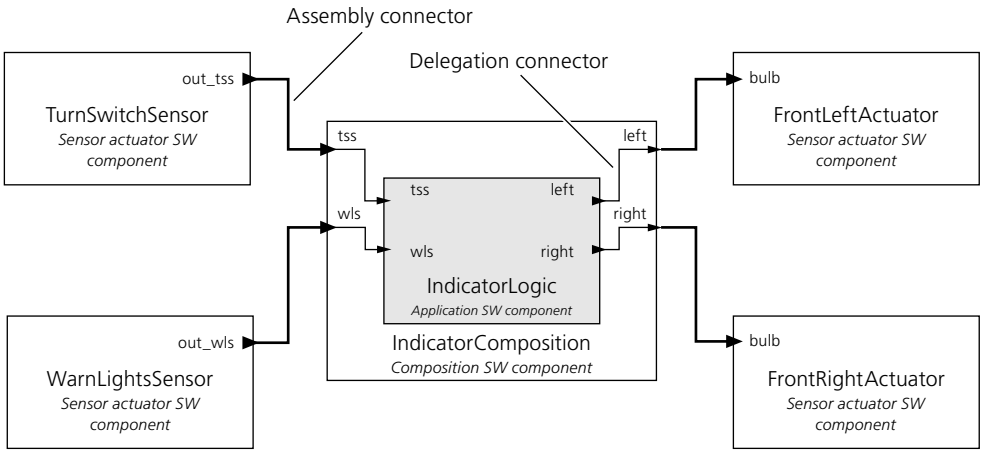
You can now connect IndicatorComposition to its underlying IndicatorLogic SWC in the next step.

Step 7: How to Connect a Composition SW Component Type to Its Inner Software Components

Objective

In the previous step, you connected the software components in the RootSwComposition composition diagram via assembly connectors.

To finish modeling the interconnections between all the software components, you have to connect the composition to its inner software components via delegation connectors. A delegation connector connects a delegation port of a composition and the port of a component contained in the composition. The following illustration shows the difference between assembly connectors and delegation connectors.



The software components of RootSwComposition are connected via assembly connectors. The composition is connected to its inner SWCs via delegation connectors. If IndicatorComposition contains more than one SWC, they can also be interconnected via assembly connectors.

You will now add ports to the IndicatorLogic software component by delegating the ports of IndicatorComposition to them.

Method

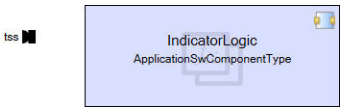
To connect a composition SW component type to its inner software components

1. On the Tools (Diagram context) ribbon, click Edit – Connection Mode to enable to the edit mode/connection mode.

   The mouse pointer changes to .

2. In the Project Manager, select the tss port of IndicatorComposition and drag it to the IndicatorComposition diagram.

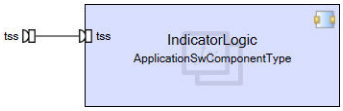
SystemDesk displays the tss port in the composition diagram. The composition itself has not been changed.



3 Click the tss port and drag it to the IndicatorLogic SWC.

A new port is added to the IndicatorLogic SWC and connected with the IndicatorComposition's tss port.

4 Rename the new port `tss`.



5 Repeat steps 2 to 4 for the remaining port pairs:

- wls
- left
- right

Result

You connected the SWCs of RootSwComposition and delegated the ports of IndicatorComposition to its inner software component via delegation connectors. By doing so, you added ports to the IndicatorComposition's inner software component, IndicatorLogic. You have completed the modeling of the software architecture.

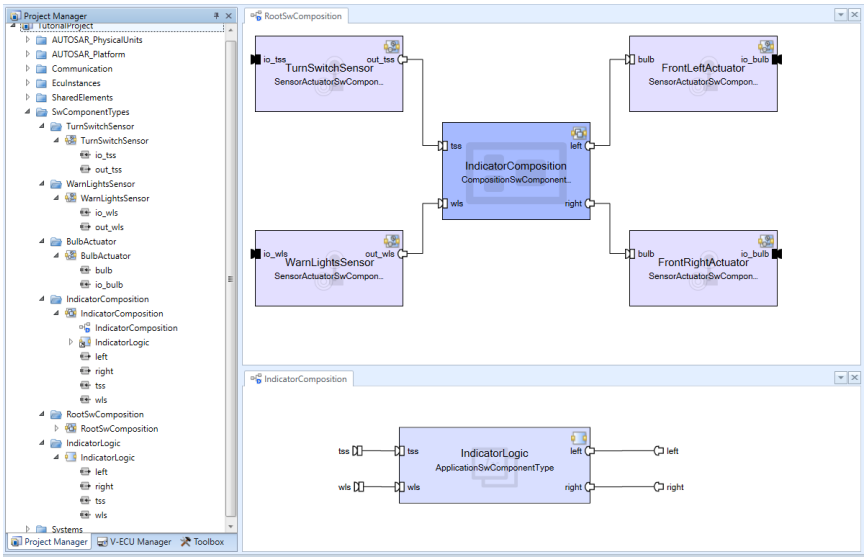
What's next

Now check your work against the result you should have achieved. Refer to Result of Lesson 2.

Result of Lesson 2

Summary

If you have done all the steps in this lesson, your SystemDesk project should look like this:



Tip

To reproduce the horizontal tiling of the composition diagrams shown in the illustration above, right-click the IndicatorComposition composition diagram tab and select New horizontal tab group.

In this lesson, you learned how to model the software architecture consisting of compositions and atomic software components which are connected via their ports. You created a RootSwComposition composition diagram for the entire software architecture, and you added software components to it graphically.

Demo

You can reproduce the result of this lesson with the demo script `Lesson02_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information

For a detailed description of creating and connecting software components, ports, and interfaces, refer to Working with AUTOSAR Elements (🕮 SystemDesk Manual).

For detailed information on working with the composition and component diagrams, refer to Working with Diagrams (🕮 SystemDesk Manual).

What's next

The next two lessons show you how to model the interfaces of software components, starting with sender receiver interfaces in the next lesson.

Lesson 3: Modeling Sender Receiver Interfaces of Software Components

Where to go from here

Information in this section

Overview of Lesson 3 62
You will now add and model sender receiver interfaces that are used to connect the ports of SWCs.

Step 1: How to Create Sender Receiver Interfaces 63
You will now create sender receiver interfaces.

Step 2: How to Add Variable Data Prototypes to Sender Receiver Interfaces 66
You will now add one variable data prototype to each sender receiver interface.

Step 3: How to Define Application Data Types and Compu Methods 68
In this step, you will define the necessary data types for your sender receiver interfaces.

Step 4: How to Assign Data Types to Variable Data Prototypes 73
Now you will assign the data types to the variable data prototypes.

Step 5: How to Assign Interfaces to Ports 75
You will now learn how to add the interfaces to the ports of the software components.

Step 6: How to Specify Queued or Nonqueued Communication and Initial Values 80
In this step, you will set the communication of data elements to queued or nonqueued.

Result of Lesson 3 88
In this lesson, you learned how to model sender receiver interfaces.

Overview of Lesson 3

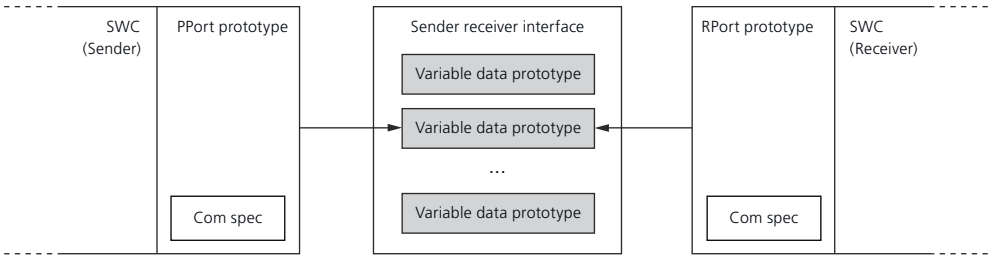
Starting point

In the previous lesson, you learned how to model a software architecture. You created software components and connected them via their ports.

What will you learn?

An interface is attached to a port and describes the data or operations that are provided or required by a software component via that port.

Sender receiver interfaces Sender receiver interfaces are used for data exchange between software components. The interface specifies which data is transferred from the sender to the receiver.



The illustration above shows sender receiver communication between software components according to AUTOSAR.

A sender receiver interface is composed of variable data prototypes and specifies the transferred data. The variable data prototypes are of a data type which describes the possible values of a variable on the physical and on the internal level. The internal values of the data type can be converted to physical values via a compu method. The range of the values can be restricted with data constr (data constraint) elements.

Modeling sender receiver interfaces You will now add and model the sender receiver interfaces which are used to connect the ports of the SWCs shown in the previous lesson.

To model sender receiver interfaces you will:

- Create sender receiver interfaces for the communication between the IndicatorComposition SWC and all other SWCs.
- Connect the WarnLightsSensor SWC to the sensors.
- Assign the interfaces to ports.
- Create and specify variable data prototypes for the data.
- Create and specify data types, and assign them to the variable data prototypes.
- For the variable data prototypes, add a compu method to convert internal values to physical values.
- Specify the communication type between the ports.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson02_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create Sender Receiver Interfaces on page 63
- Step 2: How to Add Variable Data Prototypes to Sender Receiver Interfaces on page 66
- Step 3: How to Define Application Data Types and Compu Methods on page 68
- Step 4: How to Assign Data Types to Variable Data Prototypes on page 73
- Step 5: How to Assign Interfaces to Ports on page 75
- Step 6: How to Specify Queued or Nonqueued Communication and Initial Values on page 80

Summary

To check if the steps you executed are correct, refer to Result of Lesson 3 on page 88.

Step 1: How to Create Sender Receiver Interfaces

Objective

You will now create sender receiver interfaces for the ports you created in the last lesson. Sender receiver interfaces model the data exchange between the sender (provided port prototype) and the receiver (required port prototype).

Part 1

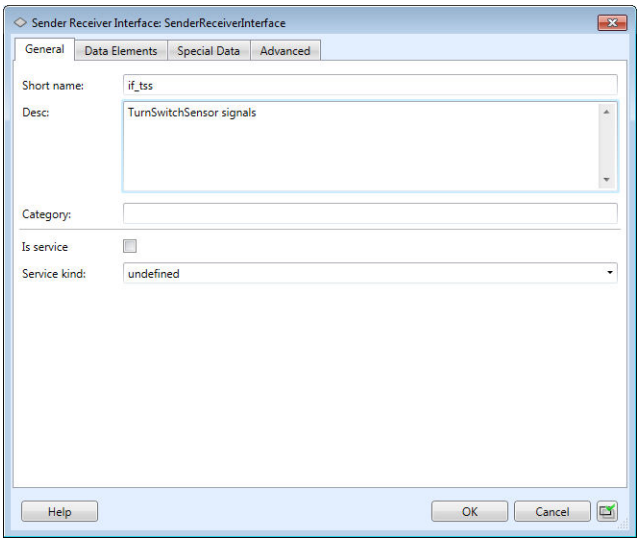
To create sender receiver interfaces used between software components

1. In the Project Manager, right-click the SharedElements/PortInterfaces package.
2. From its context menu, select New – Sender Receiver Interface.

   The Project Manager displays a new interface as a child of the package. It will become the interface defining the communication between the TurnSwitchSensor software component and IndicatorComposition.
3. In the Project Manager, double-click the new interface.

   The Sender Receiver Interface Properties dialog opens.
4. Change the interface's Short name to `if_tss`.

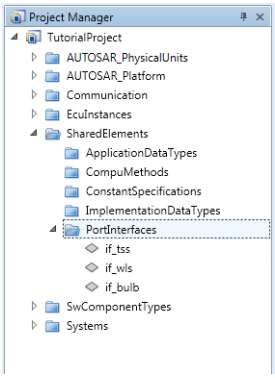
5 In the Desc edit field, type `TurnSwitchSensor signals`.



6 Click OK to confirm your settings and close the dialog.

7 Repeat steps 1 … 6 to create two more interfaces as follows.

Short Name	Desc	Purpose
`if_wls`	`WarnLightsSensor signals`	For communication between the WarnLightsSensor SWC and the IndicatorComposition
`if_bulb`	`Bulb command`	For communication between the IndicatorComposition and the BulbActuator SWC



Interim result

You created three interfaces, which define the communication within the IndicatorComposition. Now you will define one sender–receiver interface which defines the communication of the WarnLightsSensor SWC with the sensors which provide the status of the warning light switch.

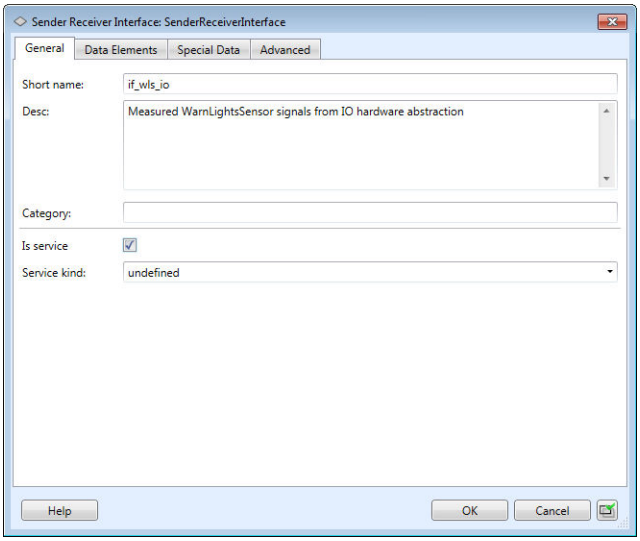
Part 2

To create sender receiver interfaces as the connection to the sensors

1. Right-click the PortInterfaces package.
2. From its context menu, select New – Sender Receiver Interface.

   The Project Manager displays a new interface as a child of the package. It will become the interface defining the communication between the WarnLightsSensor SWC and the sensors to receive the status of the warning light switch.
3. In the Project Manager, double-click the new interface.

   The Sender Receiver Interface Properties dialog opens.
4. Change the element's Short name to `if_wls_io`.
5. In the Desc edit field, type `Measured WarnLightsSensor signals from IO hardware abstraction`.
6. Select the Is service checkbox.



The Is service property indicates that the interface is used to connect to a basic software service.

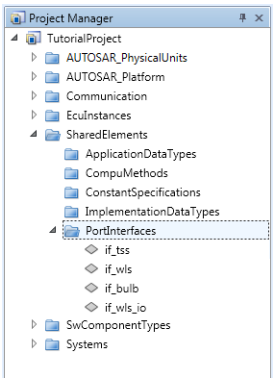
Note

The checkbox has three states:
- Not selected
- Selected
- Undefined (The checkbox is highlighted blue.)

7. Click OK to confirm your settings and close the dialog.

Result

You created three interfaces, which define the communication within the IndicatorComposition, and one interface defining the communication of WarnLightsSensor SWC with the sensors.



What's next

You will add variable data prototypes to your sender receiver interfaces in the next step.

Step 2: How to Add Variable Data Prototypes to Sender Receiver Interfaces

Objective

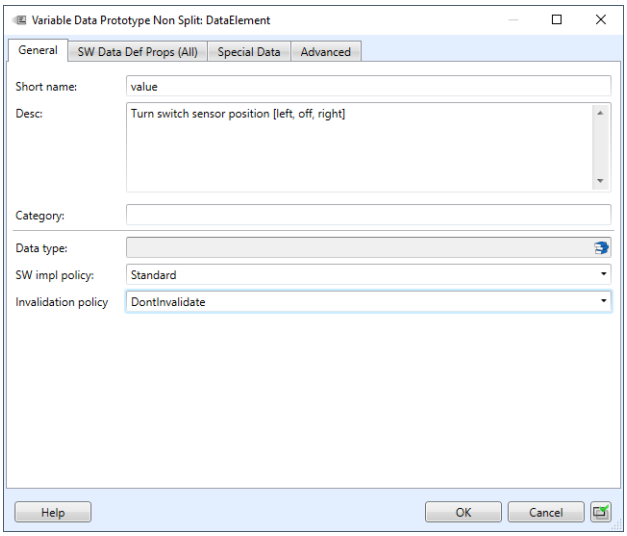
Sender receiver interfaces are composed of variable data prototypes, which represent the data sent through the interfaces. You will now add one variable data prototype to each sender receiver interface.

Method

To add variable data prototypes to sender receiver interfaces

1. In the Project Manager, right-click the if_tss interface.
2. From the element's context menu, select New – Data Element.
   The Project Manager displays a variable data prototype as a child of the interface.
3. In the Project Manager, double-click the new variable data prototype.
   The Variable Data Prototype Properties dialog opens.
4. Change the element's Short name to `value`.
5. In the Desc edit field, type `Turn switch sensor position [left, off, right]`.

6 In the Invalidation policy field, select `DontInvalidate`.
The ability to invalidate a data element is not required in this tutorial.



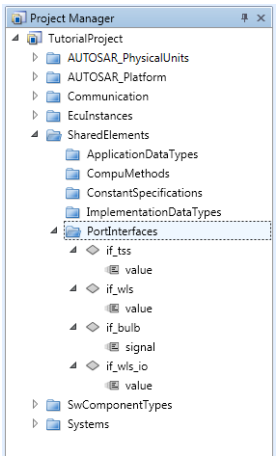
7 Click OK to close the dialog.

8 Repeat steps 1 … 7 to add three more variable data prototypes to interfaces as follows.

Interface	Variable Data Prototype	Desc
if_wls	`value`	`Warn lights sensor position [off, on]`
if_bulb	`signal`	`Bulb command [off, on]`
if_wls_io	`value`	`Measured position of warn lights sensor from IO hardware abstraction [off, on]`

Result

You added a variable data prototype to each of the four sender receiver interfaces.



What's next

To define the type of data sent through the interfaces, you have to define data types and assign them to the variable data prototypes.

You will learn how to define data types in the next step.

Step 3: How to Define Application Data Types and Compu Methods

Objective

The variable data prototypes you created in the last step will define the data which is exchanged through the sender receiver interfaces. To do this, you need to define their data types.

The properties of a data type define the values that a variable can contain. The data type can have the Boolean value category, for example.

AUTOSAR has defined two different levels of data types for developing ECU software. Application data types support the application view of developing ECU software while implementation data types let you specify implementation details such as SW base types or endianness. In this tutorial you will work with application data types.

Compu methods define the relationship between internal values on the target ECU and converted values (real-world, physical).

By using a data constr element, you can also specify a minimum and a maximum value to define the range of values for a data type. In contrast to data types of the value category, you do not need to define compu method and data constr elements for the Boolean category since the meanings of the internal and physical values are intuitive.

In this step, you will define the necessary data types for your sender receiver interfaces. AUTOSAR allows you to specify whether a data type will be accessible via a calibration and measurement tool. You will also define a compu method which defines the display text on the host PC for enum values on the target ECU. You will associate this compu method with a data type.

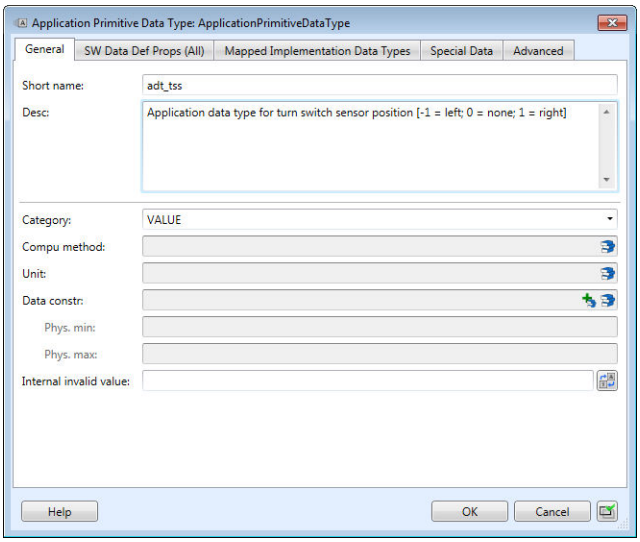
Part 1

To define application primitive data types

1. In the Project Manager, right-click the SharedElements/ApplicationDataTypes package.
2. From the context menu, select New – Application Primitive Data Type.
   The Project Manager displays a new data type as a child of the package.
3. Double-click the new element.
   The Application Primitive Data Type Properties dialog opens.

4 On the General page, specify the settings as follows:

Short name	Desc	Category
adt_tss	`Application data type for turn switch sensor position [-1 = left; 0 = none; 1 = right]`	VALUE

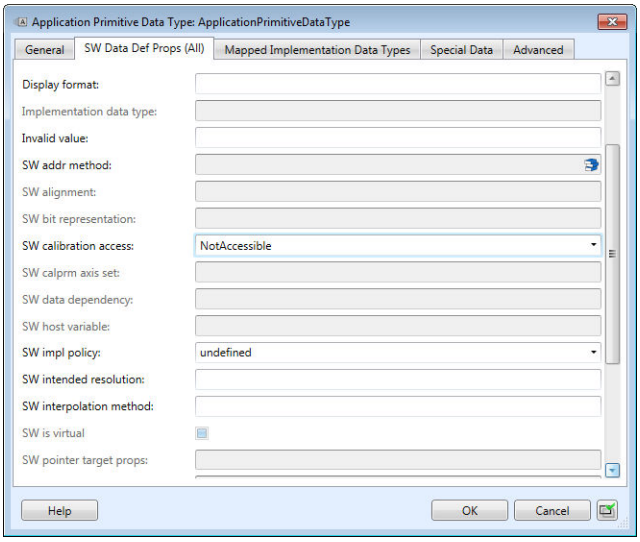


5 In the Unit field, click .

The Select Unit dialog opens.

6 Select NoUnit and click OK to return to the Application Primitive Data Type Properties dialog.

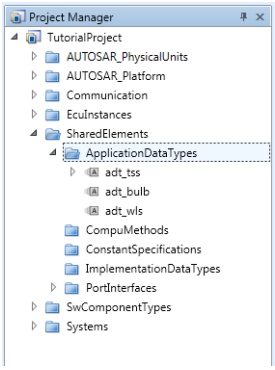
7 On the SW Data Def Props (All) page, select NotAccessible for the SW calibration access property.



8 Click OK to confirm your entries and close the dialog.

9 Create the following application primitive data types including their properties as described in steps 1 … 8:

Short name (General Page)	Desc (General Page)	Category (General Page)	SW calibration access (SW Data Def Props (All) Page)
adt_bulb	`Application data type for bulb command [0 = off; 1 = on]`	`BOOLEAN`	`NotAccessible`
adt_wls	`Application data type for warning light sensor position [0 = off; 1 = on]`	`BOOLEAN`	`NotAccessible`



Interim result

You specified the application primitive data types of the values which will be exchanged through the sender receiver interfaces.

Before you associate them with the variable data prototypes of the interfaces, you will now create a compu method that converts the internal integer values representing the possible indicator switch positions into text. The text explains the meaning of the integer values and can be displayed by Measurement and Calibration tools.

Part 2

To define compu methods

1 In the Project Manager, right-click the SharedElements/CompuMethods package.

2 From the context menu, select New – Compu Method.

The Project Manager displays a new compu method as a child of the package.

3 Double-click the compu method.

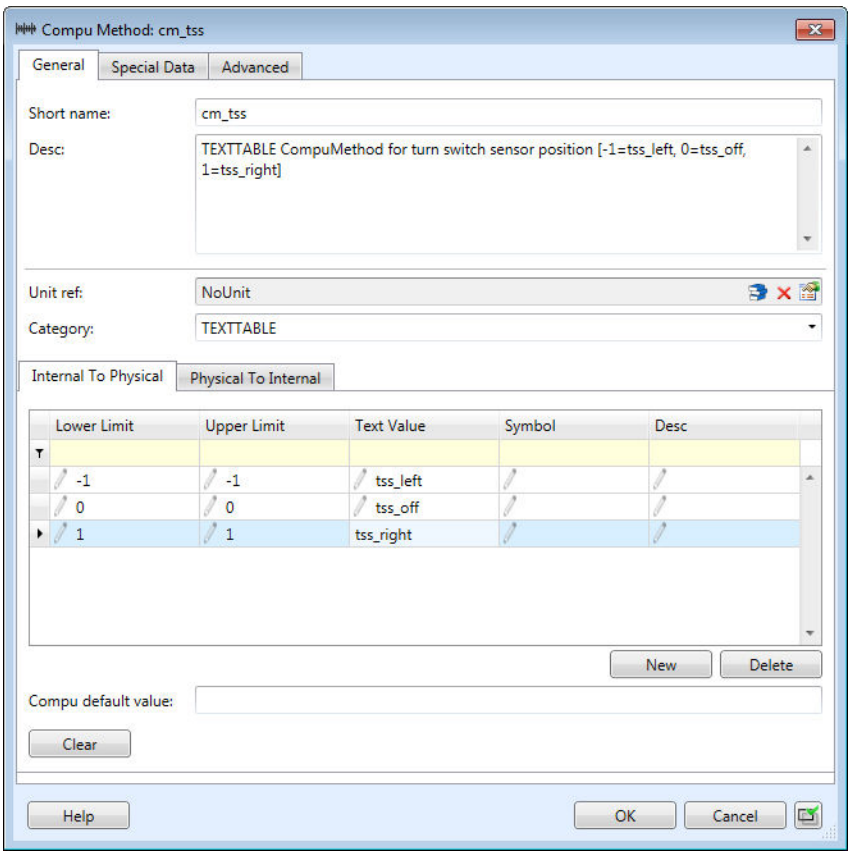
The Compu Method Properties dialog opens.

4 On the General page, specify the settings as follows:

Short name	Desc	Unit ref	Category
cm_tss	`TEXTTABLE CompuMethod for turn switch sensor position [-1=tss_left, 0=tss_off, 1=tss_right]`	`NoUnit`	`TEXTTABLE`

To enter the conversion definition, click New on the Internal To Physical page. Fill the table as follows:

Lower Limit	Upper Limit	Text Value
-1	-1	`tss_left`
0	0	`tss_off`
1	1	`tss_right`



5 Click OK to close the Compu Method Properties dialog.

Interim result

You created a compu method for the adt_tss data type.

You will now assign the compu method to the data type.

Part 3

To assign compu methods to data types

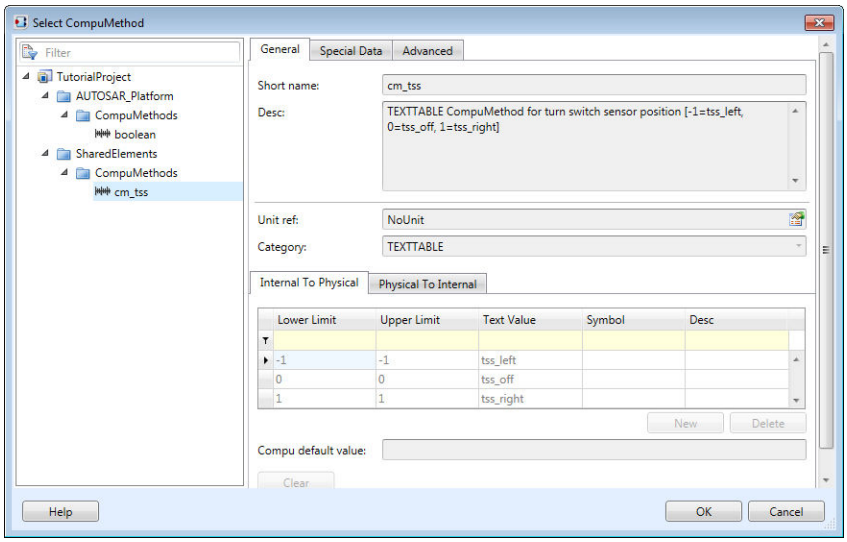
1 In the Project Manager, double-click the adt_tss application primitive data type.

The Application Primitive Data Type Properties dialog opens.

2 In the CompuMethod field, click .

The Select CompuMethod dialog opens.

3 In the tree view, select cm_tss.



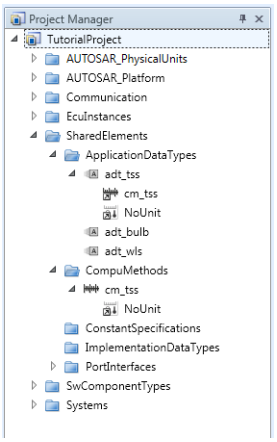
4 Click OK to close the dialog and return to the Application Primitive Data Type dialog.

The cm_tss compu method is assigned to the adt_tss data type.

5 Click OK to confirm your settings and close the dialog.

Result

You specified three data types and one compu method, and assigned the compu method to the adt_tss data type.



What's next

You will assign the data types to the variable data prototypes in the next step.

Step 4: How to Assign Data Types to Variable Data Prototypes

Objective

In the previous step, you learned how to define data types in the Project Manager. You created a data type for each variable data prototype of each sender receiver interface. Now you will assign the data types to the variable data prototypes.

Method

To assign data types to variable data prototypes

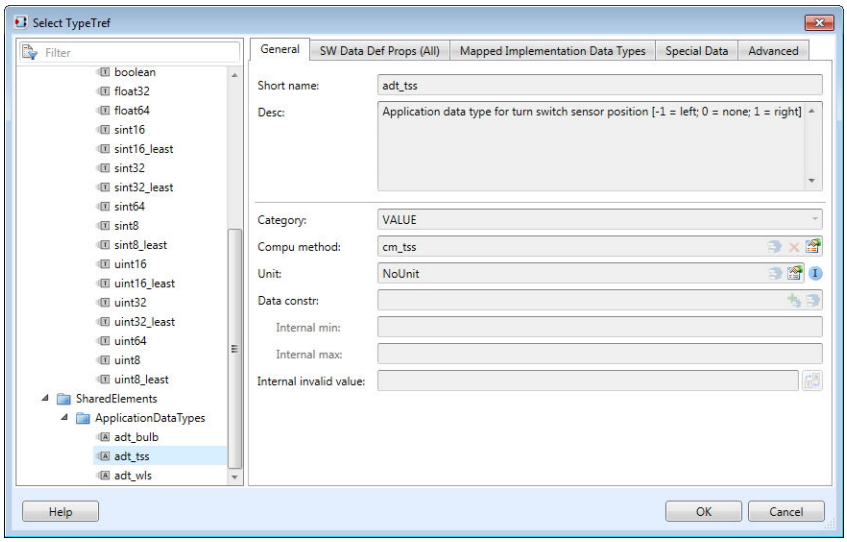
1 In the Project Manager, double-click the value variable data prototype of the if_tss interface.

The Variable Data Prototype Properties dialog opens.

2 Click in the Data Type field.

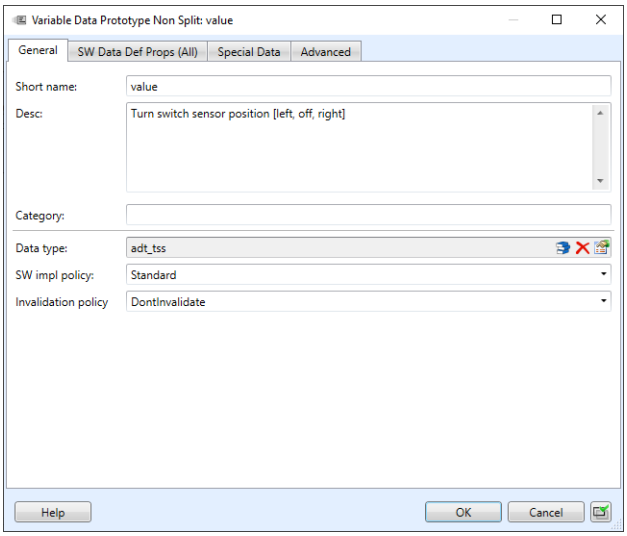
The Select TypeTref dialog opens.

3 Select the adt_tss data type.



4 Click OK to return to the Variable Data Prototype Properties dialog.

The adt_tss data type is associated with the value variable data prototype.



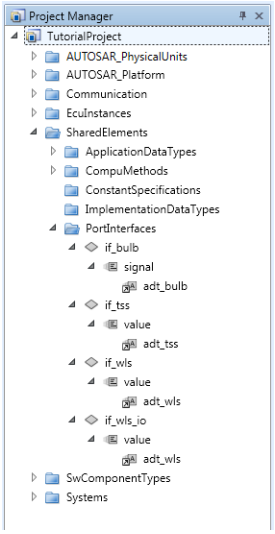
5 Click OK to close the Variable Data Prototype Properties dialog.

6 Repeat steps 1 … 5 to add three more data types to the remaining variable data prototypes as follows.

Interface	Variable Data Prototype	Data Type
if_bulb	signal	adt_bulb
if_wls	value	adt_wls
if_wls_io	value	adt_wls

Result

You specified the data which is sent through the interfaces by adding the data types to the variable data prototypes of the interfaces.



What's next

Now you can attach the interfaces to the ports of the software components in the next step.

Step 5: How to Assign Interfaces to Ports

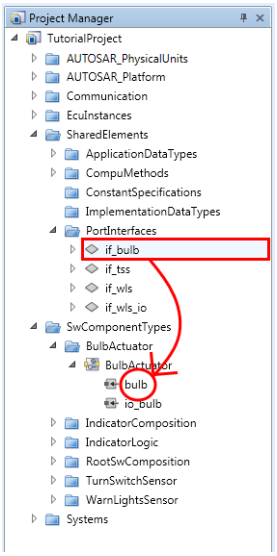
Objective

Having specified sender receiver interfaces including variable data prototypes for IndicatorComposition in the previous steps, you will now learn how to add the interfaces to the ports of the software components. One interface can be used for several ports.

Part 1

To assign interfaces in the Project Manager

1 In the Project Manager, select the if_bulb interface, and drag it to the BulbActuator's bulb port.



The interface is added to the port.

Interim result

You assigned an interface to a port of the BulbActuator SWC.

The Project Manager displays the interface as a child of the port. It has an additional symbol at the bottom left, indicating a reference to the definition of the interface in the Interfaces package of the Project Manager.

Part 2

To assign interfaces via the component diagram

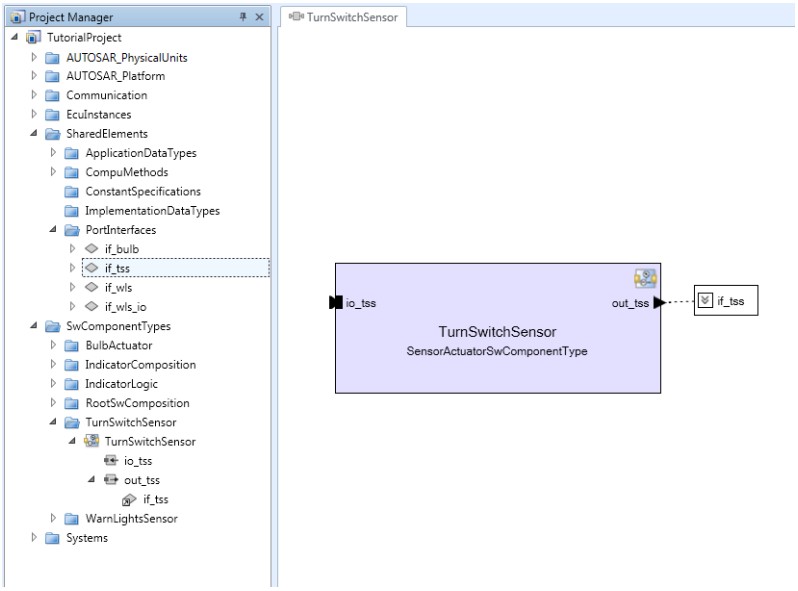
1 In the Project Manager, right-click the TurnSwitchSensor software component.

2 From the element's context menu, select Open Component Diagram.

The element's component diagram opens in SystemDesk's working area.

3 Drag the if_tss interface from the Project Manager to the out_tss port of the TurnSwitchSensor in the diagram.

The interface is assigned to the port.

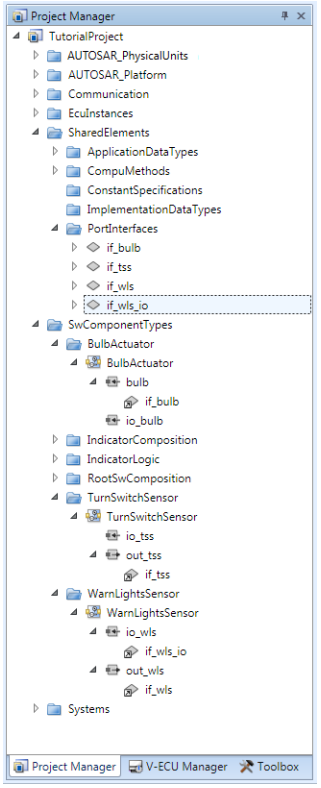


4 Repeat steps 1 … 3 to assign the following interfaces to ports:

Software Component	Port	Matching Interface
WarnLightsSensor	out_wls	if_wls
WarnLightsSensor	io_wls	if_wls_io

Interim result

You assigned interfaces to the ports of the TurnSwitchSensor SWC and the WarnLightsSensor SWC.



You can now use the IndicatorComposition composition diagram to assign interfaces to the remaining ports.

Part 3

To assign interfaces to delegation ports via the composition diagram

1 Open the IndicatorComposition composition diagram and assign interfaces to the ports of the IndicatorLogic atomic SWC as follows:

Port	Matching Interface
tss	if_tss
wls	if_wls
left	if_bulb
right	if_bulb

2 Assign interfaces to the delegation ports as follows:

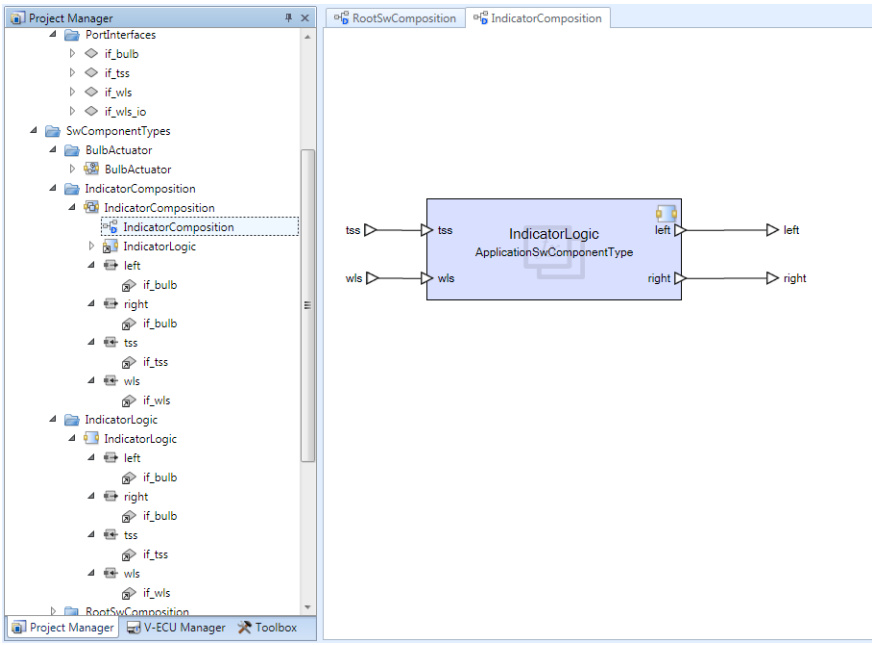
Delegation Port	Matching Interface
tss	if_tss
wls	if_wls

Delegation Port	Matching Interface
left	if_bulb
right	if_bulb

Result

You have assigned interfaces to ports as follows:

- Part 1: BulbActuator (FrontLeftActuator, FrontRightActuator); if_bulb
- Part 2: WarnLightsSensor and TurnSwitchSensor; if_wls, if_wls_io and if_tss
- Part 3: IndicatorLogic and IndicatorComposition; if_tss, if_wls, and if_bulb



What's next

Having assigned the interfaces to ports, you can now define whether the variables sent through each port are queued or nonqueued. If they are nonqueued, you can specify initial values in the next step.

Step 6: How to Specify Queued or Nonqueued Communication and Initial Values

Objective

Queued and nonqueued communication Sender receiver communication can be *queued* or *nonqueued*.

If it is queued, the sent data is processed using a *first-in-first-out* queue with a given length.

If it is nonqueued, the receiver always has access to the data sent last. You need to specify an initial value that is used if no data was sent yet. The initial value can only be specified in the com spec element of each port (according to AUTOSAR, the init value field of a variable data prototype is ignored for sender receiver communication). If the initial value is specified for both the sender and the receiver port the initial value specified at the receiver port is used.

Implementation and application constants To specify initial values, you have to create constants according to the data type of the sent data element. You can create constants for application and implementation data types. Application constants represent physical values whereas implementation constants represent internal values.

To generate RTE code with third-party tools, you might have to specify both implementation constants and application constants, and also a mapping between them. However, to generate RTE code with SystemDesk, it is sufficient to specify application constants only.

In this step, you will set the communication from the sensors through the WarnLightsSensor SWC to the IndicatorComposition SWC to queued, whereas all other data elements are nonqueued. You will create application constants and specify initial values for the nonqueued communication.

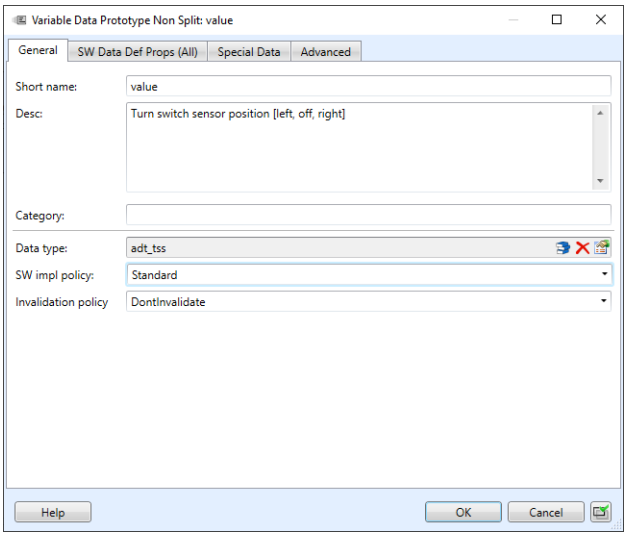
Part 1

To specify variable data prototypes for nonqueued communication

1 In the Project Manager, double-click the value variable data prototype of the if_tss interface.

The Variable Data Prototype Properties dialog opens.

2 For nonqueued communication, select Standard from the SW impl policy list.



3 Click OK to confirm your settings and close the dialog.

4 Repeat steps 1 … 3 for the signal variable data prototype of the if_bulb interface.

Interim result

You specified the variable data prototypes for nonqueued communication between the TurnSwitchSensor SWC, the IndicatorComposition SWC and the two BulbActuator SWC prototypes FrontLeftActuator and FrontRightActuator.

Part 2

To specify initial value constants

1 Add a new package named ConstantSpecifications to the BulbActuator package.

2 From the ConstantSpecifications package's context menu, select New – New SWC-T – Constant Specification.

SystemDesk generates a new constant specification in the ConstantSpecifications package.

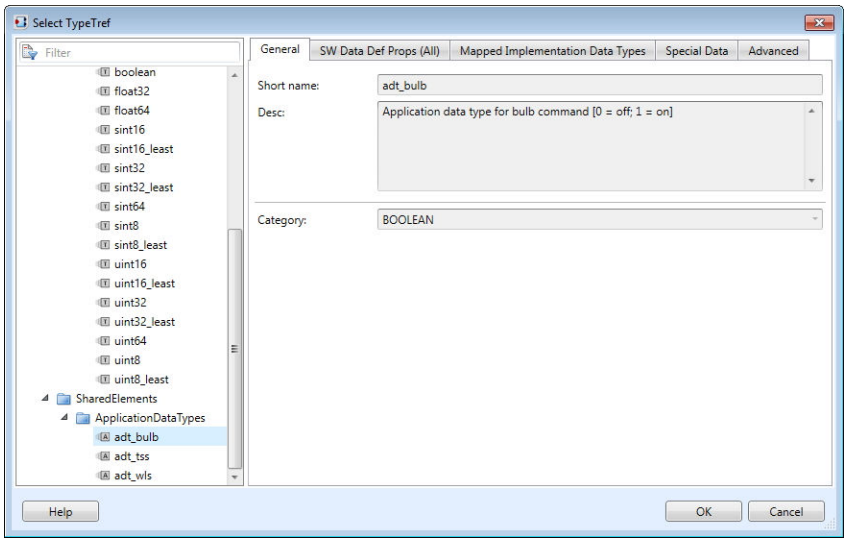
3 Double-click the new constant specification to open the Constant Specification Properties dialog.

4 Specify the settings of the constant specification as follows:

Setting	Entry
Short name	`CONST_bulb_signal`
Desc	`Initvalue constant for data element signal`
Category	`BOOLEAN`

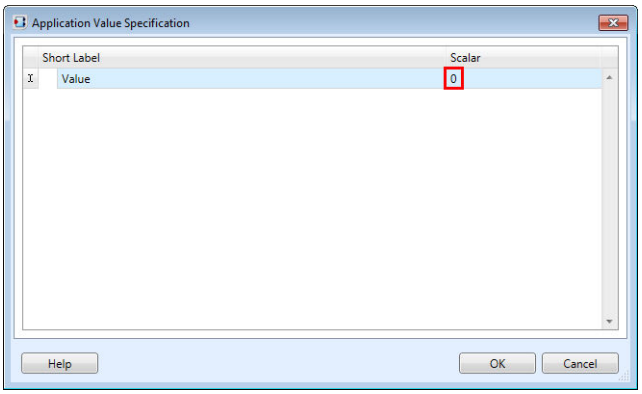
5 Click the ✚ button in the Value specification field.
The Select TypeTref dialog opens.

6 Select the adt_bulb application data type.

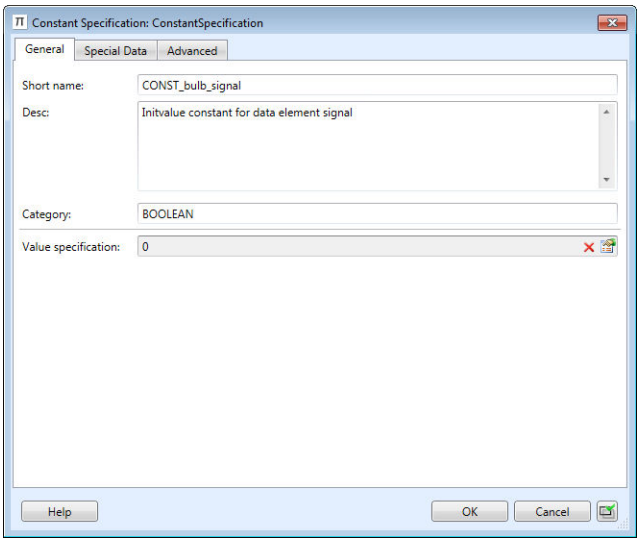


7 Click OK to close the Select TypeTref dialog.
The Application Value Specification dialog opens.

8 Enter 0 in the Scalar field.



9 Click OK to confirm your settings and return to the Constant Specification Properties dialog.



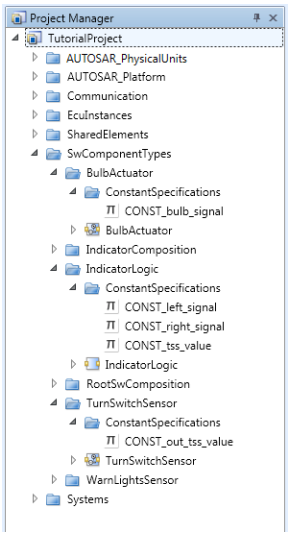
10 Click OK to close to the Constant Specification Properties dialog.

11 Repeat steps 1 … 10 to add ConstantSpecifications packages and constants to the IndicatorLogic and TurnSwitchSensor packages.

Package	Constant Short Name	Desc	Category	Application Data Type	Value Specification
IndicatorLogic	CONST_left_signal	Initvalue constant for data element signal	BOOLEAN	adt_bulb	0
	CONST_right_signal	Initvalue constant for data element signal	BOOLEAN	adt_bulb	0
	CONST_tss_value	Initvalue constant for data element value	VALUE	adt_tss	0
TurnSwitchSensor	CONST_out_tss_value	Initvalue constant for data element value	VALUE	adt_tss	0

Interim result

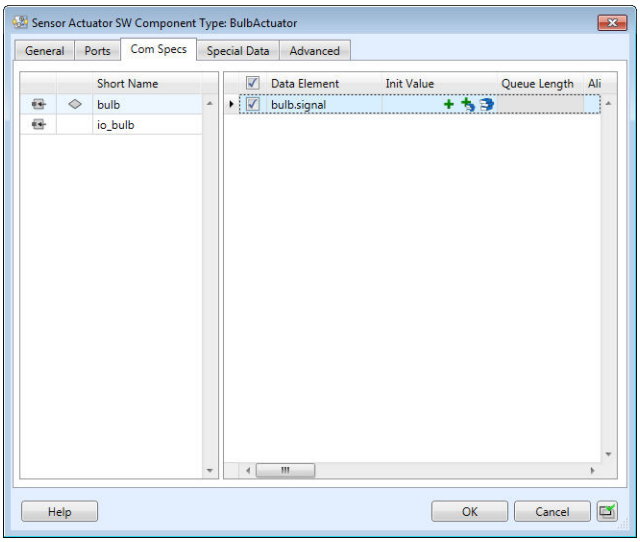
You specified initial value constants to be used for variable data prototypes of receiver ports with nonqueued communication.



Part 3

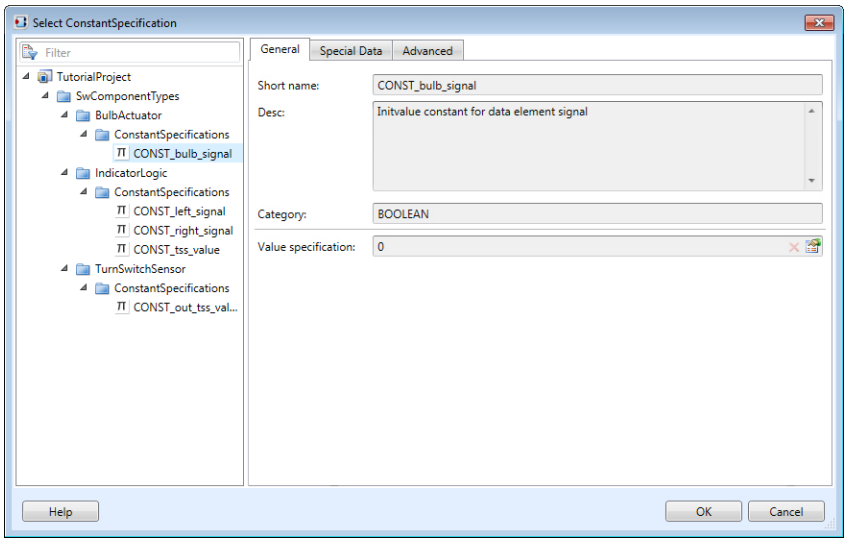
To create communication specifications and assign initial values

1. In the Project Manager, double-click the BulbActuator SWC to open its Properties dialog.
2. Change to the Com Specs page.
3. Click the bulb port and select the bulb.signal data element. SystemDesk creates a communication specification (com spec).

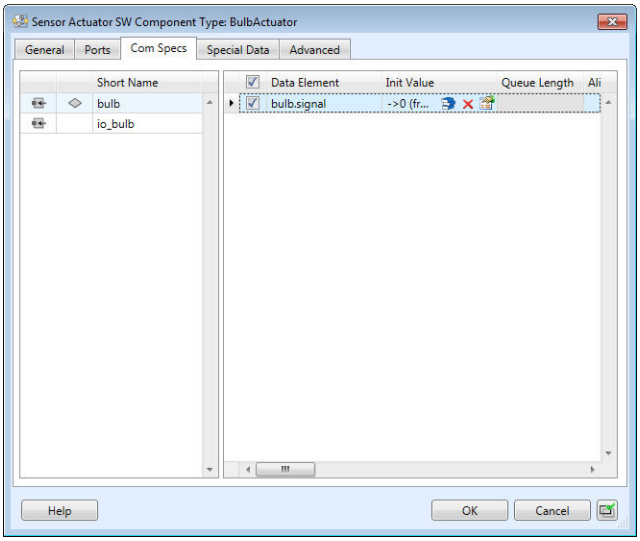


4. Click the button in the Init Value field.
   The Select ConstantSpecification dialog opens.

5 Select the CONST_bulb_signal constant specification.



6 Click OK to close the Select ConstantSpecification dialog and return to the Com Specs page of the BulbActuator SWC's Properties dialog.



7 Click OK to close the BulbActuator SWC's Properties dialog.

8 Repeat steps 1 … 7 to assign initial value constants to further ports as follows:

Software Component	Port	Variable	Constant
IndicatorLogic	left	signal	CONST_left_signal
IndicatorLogic	right	signal	CONST_right_signal
IndicatorLogic	tss	value	CONST_tss_value
TurnSwitchSensor	out_tss	value	CONST_out_tss_value

Interim result

You assigned initial value constants to ports with nonqueued communication.

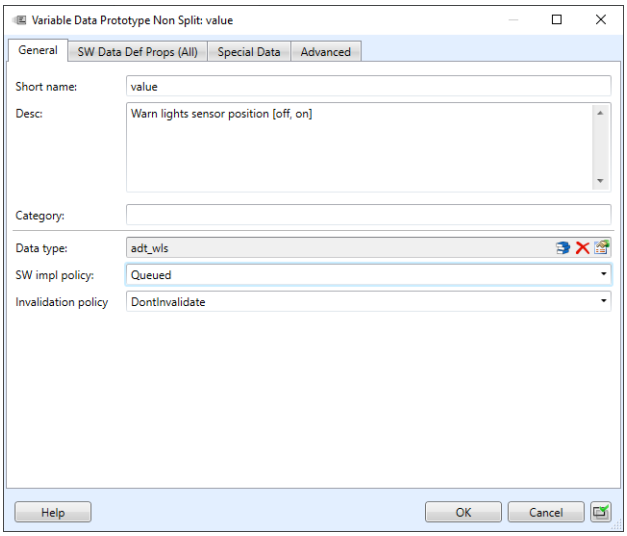
Part 4

To specify queued communication

1. In the Project Manager, double-click the value variable data prototype of the if_wls interface.

   The Variable Data Prototype Properties dialog opens.

2. Select the SW impl policy to be Queued.

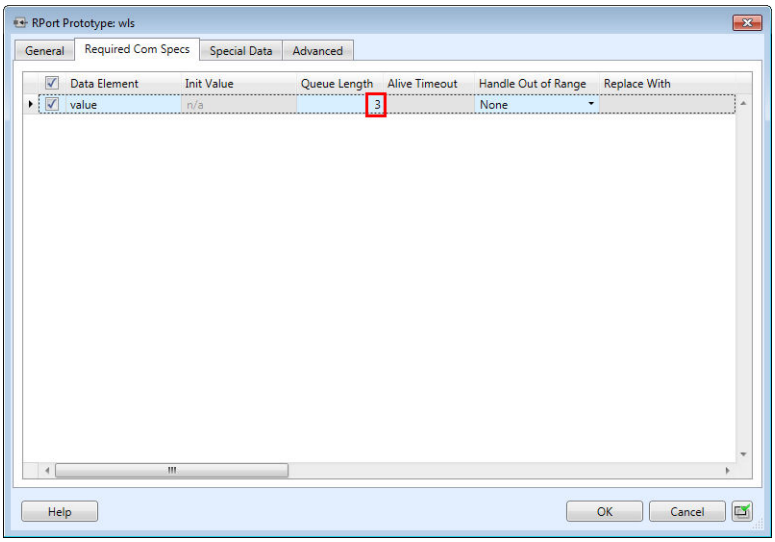


3. Click OK to confirm your settings and close the dialog.
4. In the Project Manager, double-click the wls port of the IndicatorLogic SWC.

   The RPort Prototype dialog opens.

5. On the Required Com Specs page, select the value variable data prototype.

6 Set the Queue Length to 3.



7 Click OK to close the dialog.

8 Repeat steps 1 … 7 for the value variable data prototype of the if_wls_io interface, and the io_wls port of the WarnLightsSensor SWC.

Result

You specified that the communication between the sensors, the WarnLightsSensor SWC and the IndicatorComposition SWC is queued.

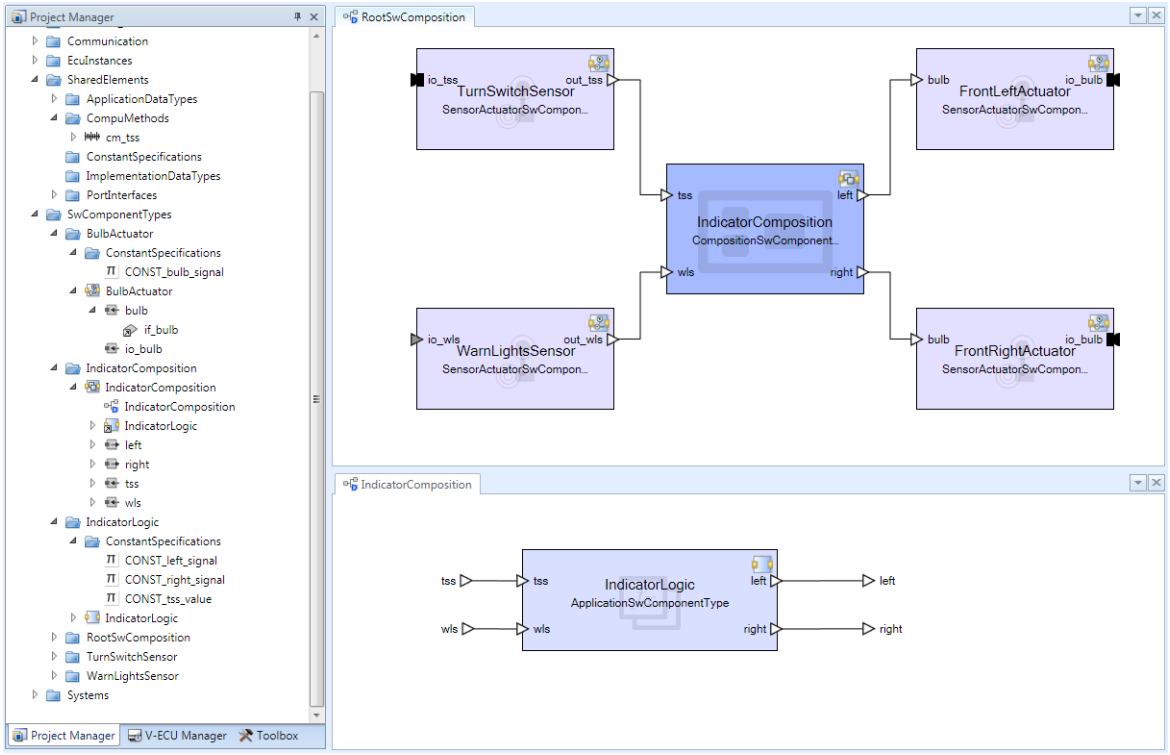
What's next

In this step, you modeled the sender receiver interfaces of software components. Now check your work against the result you should have achieved. Refer to Result of Lesson 3.

Result of Lesson 3

Summary

If you have done all the steps in this lesson, your SystemDesk project should look like this:



In this lesson, you learned how to model sender receiver interfaces. To define sender receiver communication, you added and specified variable data prototypes as data which is sent from the sender to the receiver. To specify the variable data prototype, you created and added a data type with a compu method to it. You assigned the interfaces to ports, and you defined init values and a queue length for the ports.

Demo

You can reproduce the result of this lesson with the demo script `Lesson03_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Note

The `Lesson03_complete.py` demo script also completes the modeling of all the other sender receiver interfaces.

Further information

- For detailed information on working with AR elements such as interfaces, refer to Working with AUTOSAR Elements (📖 SystemDesk Manual).
- For detailed information on working with the composition and component diagrams, refer to Working with Diagrams (📖 SystemDesk Manual).

What's next

The next lesson shows you how to model client server interfaces.

Lesson 4: Modeling Client Server Interfaces of Software Components

Where to go from here

Information in this section

Overview of Lesson 4 92
You will model client server interfaces for the ports of software components.

Step 1: How to Define Data Types and Data Constr Elements 93
In this step, you will define data types and constrain their physical values.

Step 2: How to Define Compu Methods, Units and Physical Dimensions 96
You will now define a compu method which transforms the internal value of a data type into the physical value with unit information.

Step 3: How to Create Client Server Interfaces 101
In this step, you will create two client server interfaces.

Step 4: How to Add and Define Client Server Operations 103
In this step, you will define a client server operation for each client server interface.

Step 5: How to Assign Compu Methods to Data Types, and Data Types to Argument Data Prototypes 106
In this step, you will assign data types to argument data prototypes.

Step 6: How to Assign Client Server Interfaces to Ports 110
You will now assign client server interfaces to the ports of software components.

Result of Lesson 4 112
In this lesson, you learned how to model client server interfaces.

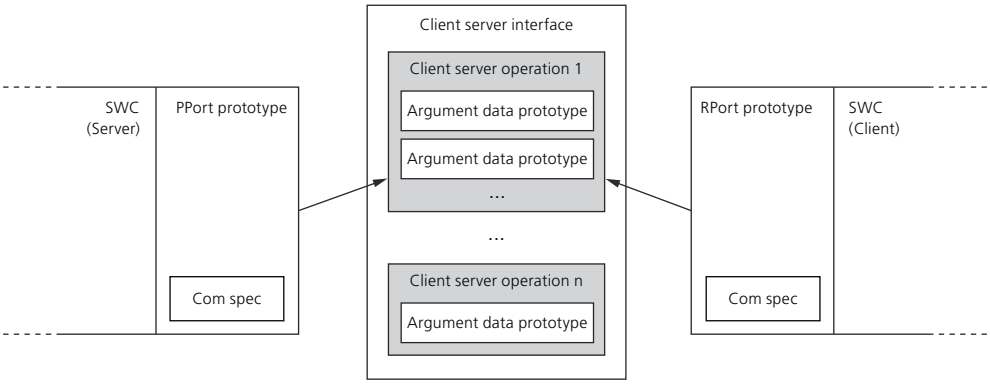
Overview of Lesson 4

Starting point

In the previous lesson, you learned how to model sender receiver interfaces and add them to ports.

What will you learn?

Client server interface Client server interfaces allow a client to call an operation at a server, which in turn provides the result to the client.



The illustration above shows client server communication between software components according to AUTOSAR.

A client server interface that is composed of client server operations with argument data prototypes specifies the server operations that a client can call. The argument data prototypes can be the client input or the server result. Like the variable data prototypes, argument data prototypes have a data type that can be additionally specified by a compu method and a data constr element.

Modeling client server interfaces You will model client server interfaces for the remaining two ports of your software components, which connect the TurnSwitchSensor and the BulbActuator SWCs to the sensors/actuators.

To model client server interfaces you will:

- Create client server interfaces to connect the TurnSwitchSensor and BulbActuator SWC to the sensors/actuators.
- Create client server operations for the client server interfaces to get or set the sensor value from the sensors/actuators.
- Create and specify argument data prototypes for the values.
- Assign the adt_bulb data type to an argument data prototype of a client server operation which is called by BulbActuator.
- Create the adt_tss_io data type, and assign it to the argument data prototype of a client server operation which is called by the TurnSwitchSensor.

  The TurnSwitchSensor SWC describes the physical position [-5 cm; 5 cm] of the indicator switch using the adt_tss_io data type.
- Add a compu method to the adt_tss_io data type to convert internal values to physical values in centimeters.

- Define a unit with a physical dimension for the compu method.
- Define a data constr element to specify the range of physical values.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson03_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Define Data Types and Data Constr Elements on page 93
- Step 2: How to Define Compu Methods, Units and Physical Dimensions on page 96
- Step 3: How to Create Client Server Interfaces on page 101
- Step 4: How to Add and Define Client Server Operations on page 103
- Step 5: How to Assign Compu Methods to Data Types, and Data Types to Argument Data Prototypes on page 106
- Step 6: How to Assign Client Server Interfaces to Ports on page 110

Summary

To check if the steps you executed are correct, refer to Result of Lesson 4 on page 112.

Step 1: How to Define Data Types and Data Constr Elements

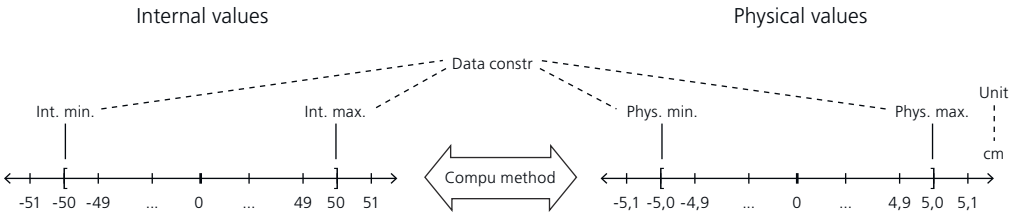
Objective

Argument data prototypes need a data type defining the type of the argument. You can constrain the physical values of a data type with a data constr element.

By using a data constr element, you can specify a minimum and a maximum value to define the range of values for a data type.

In a later step, you will create two client server interfaces. In this step, you will define the **adt_tss_io** data type for one of them and constrain its physical values. For the other client server interface, you will reuse the **adt_bulb** data type.

The illustration below shows you the relations between compu method, unit and data constr element for the adt_tss_io data type.



Method

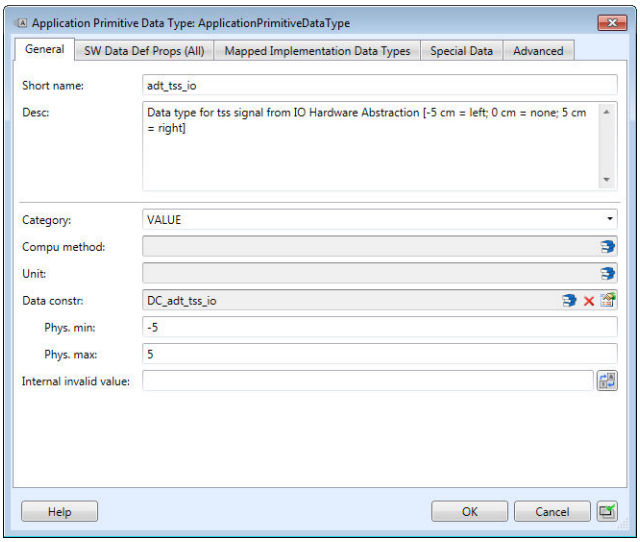
To define data types and data constr elements

1. In the Project Manager, right-click the SharedElements/ApplicationDataTypes package.
2. From the context menu, select New – Application Primitive Data Type.
   The Project Manager displays a new application primitive data type as a child of the package.
3. Double-click the application primitive data type.
   The Application Primitive Data Type Properties dialog opens.
4. On the General page of the dialog, specify the settings as follows:

Setting	Entry / Selection
Short name	`adt_tss_io`
Desc	`Data type for tss signal from IO Hardware Abstraction [-5 cm = left; 0 cm = none; 5 cm = right]`
Category	`VALUE`

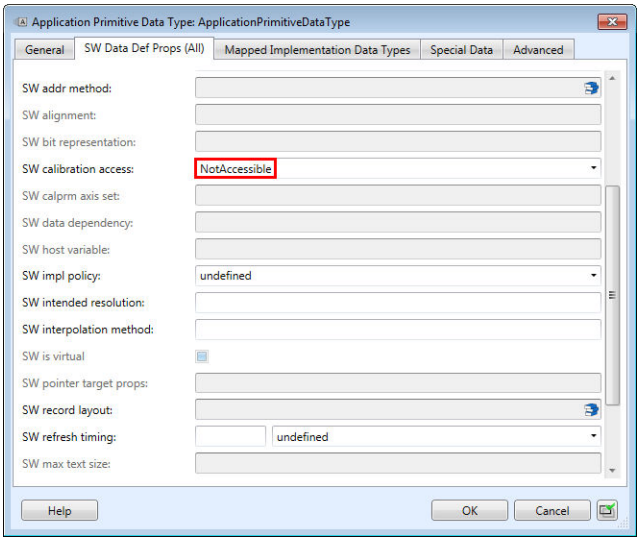
5. Click the button in the Data Constr field.
   A new Data Constr element named adt_tss_io_DC is created as a child of the AppDataTypes package.
6. Click the button in the Data Constr field.
   The Data Constr dialog opens.
7. Change the element's Short name to `DC_adt_tss_io`.
8. Enter `-5` as Phys. min, and `5` as Phys. max.

9 Click OK to close the dialog and return to the Application Primitive Data Type Properties dialog.



10 Change to the SW Data Def Props (All) page.

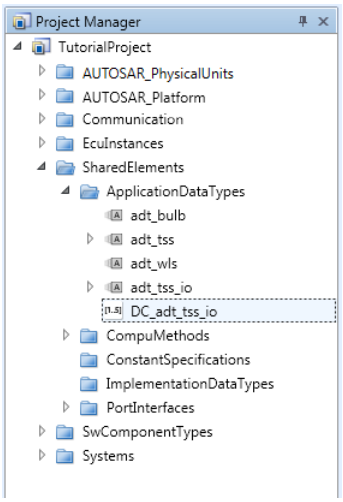
11 For the SW calibration access property, select NotAccessible.



12 Click OK to confirm your settings and close the dialog.

Result

You created a data type and added a data constr element to it. You will use this data type for a client server operation of the if_tss_io interface in a later step.



What's next

You will now define a compu method with a unit and a physical dimension for the adt_tss_io data type in the next step.

Step 2: How to Define Compu Methods, Units and Physical Dimensions

Objective

The dt_tss_io data type describes the sensor output for the position of the indicator switch. It can be a physical value between -5 cm and 5 cm. Internally, this is stored as a value between -50 and 50. You will now define a compu method which transforms the internal value into the physical value with unit information.

Part 1

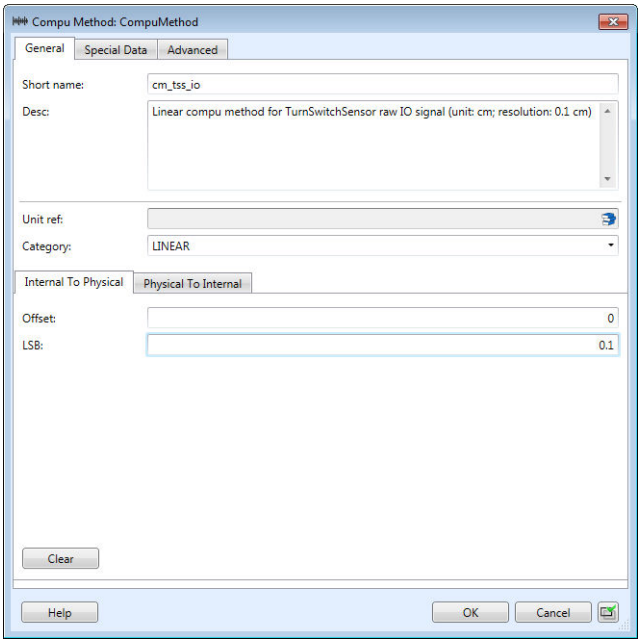
To define a compu method

1. Right-click the SharedElements/CompuMethods package.
2. From the context menu, select New – Compu Method.
   The Project Manager displays a new compu method as a child of the package.
3. Double-click the compu method.
   The Compu Method Properties dialog opens.
4. On the General page, specify the settings as follows:

Setting	Entry / Selection
Short name	`cm_tss_io`
Desc	`Linear compu method for TurnSwitchSensor raw IO signal (unit: cm; resolution: 0.1 cm)`
Category	`LINEAR`

On the **Internal To Physical** page of the **General** page, leave the offset at the default value of **0** and an LSB of **0.1**.

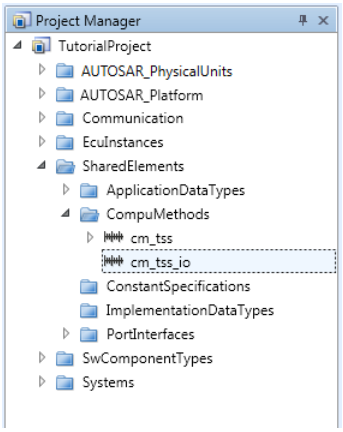
The LSB value represents a multiplicative factor, the offset an additive term. With this setting, the internal value is therefore multiplied by 0.1 and then 0 is added to it, transforming an internal value of 1 into a physical value of 0.1 cm = 1 mm.



5 Click **OK** to close the **Compu Method** Properties dialog.

Interim result

You defined a compu method.

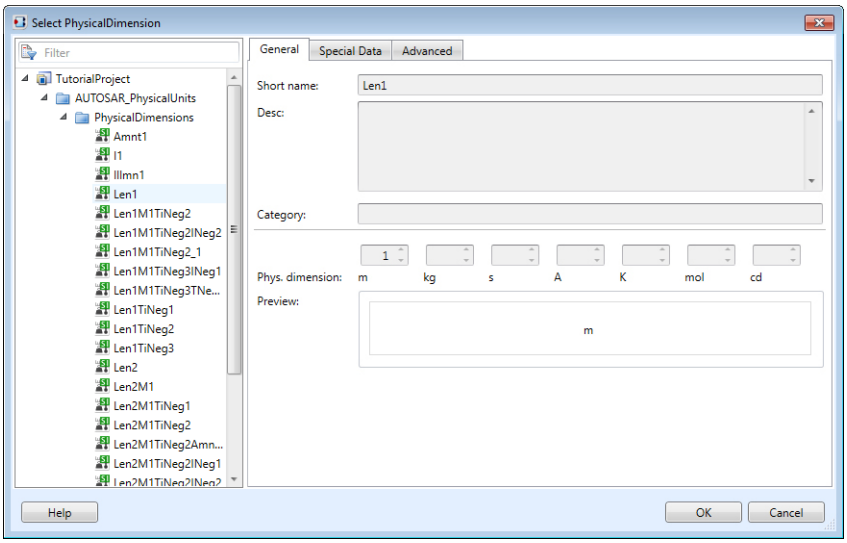


You will now define a unit with its physical dimension.

Part 2

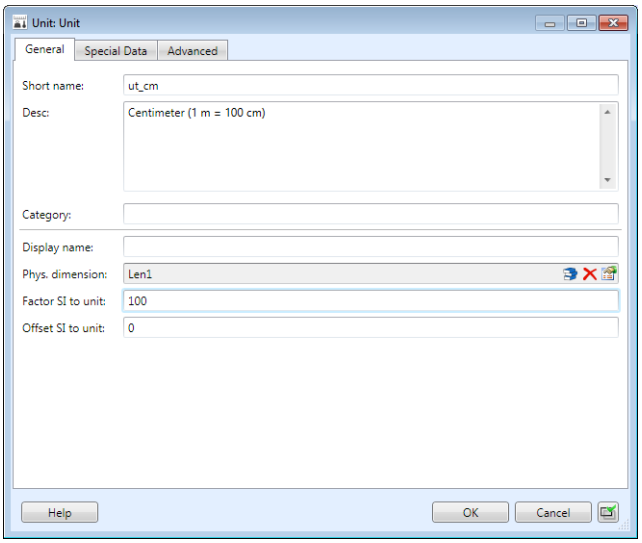
To define a unit

1. In the Project Manager, create a new package named `Units` as a child of the SharedElements package.
2. Right-click the Units package.
3. From the context menu, select New – Unit.
   The Project Manager displays a new unit as a child of the package.
4. Double-click the unit.
   The Unit Properties dialog opens.
5. On the General page, enter `ut_cm` as Short name.
6. In the Desc edit field, type `Centimeter (1 m = 100 cm)`.
7. Click the button in the Phys. dimension field.
   The Select Physical Dimension dialog opens.
8. Select Len1 from the tree view.



9. Click OK to return to the Unit Properties dialog.
   The Len1 physical dimension is associated with the ut_cm unit.

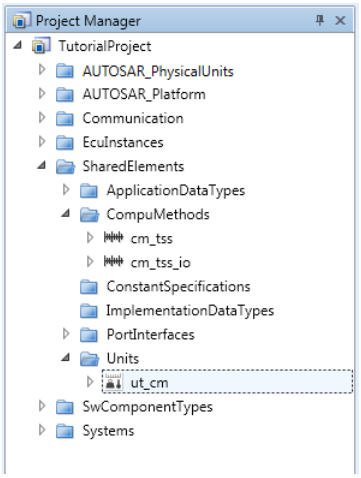
10 Enter `100` in the Factor SI to unit edit field. This factor converts a derived unit to a SI unit, i.e., centimeter to meter.



11 Click OK to close the Unit Properties dialog.

Interim result

You defined a unit.



Now you will assign the unit to the compu method.

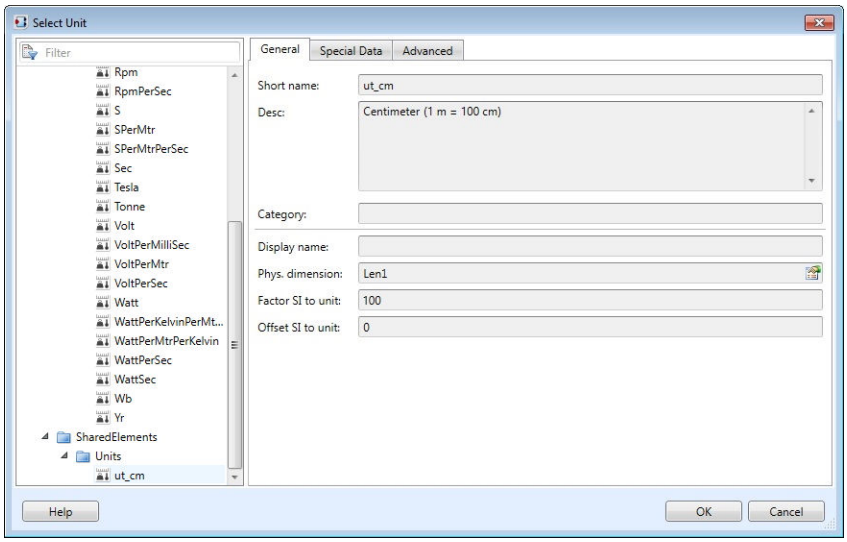
Part 3

How to assign a unit to a compu method

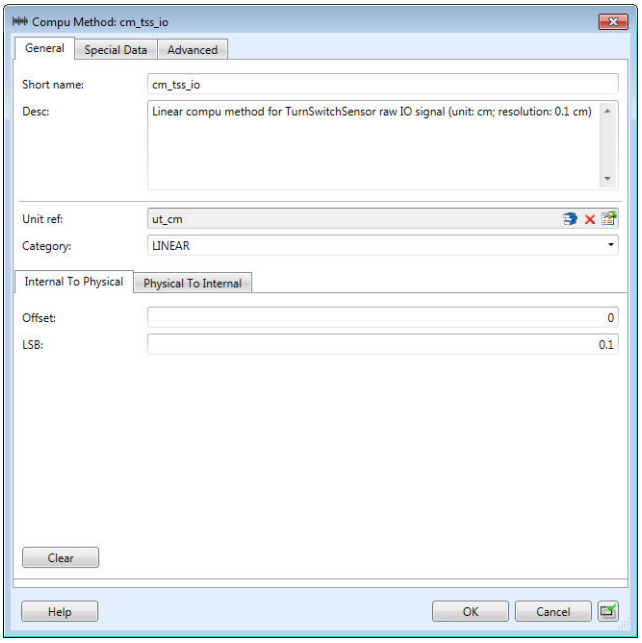
1 In the Project Manager, double-click the cm_tss_io compu method.
The Compu Method Properties dialog opens.

2 On the General page, click the Unit ref button.
The Select Unit dialog opens.

3 Select ut_cm from the tree view.



4 Click **OK** to return to the **Compu Method** Properties dialog.
The ut_cm unit is associated with the cm_tss_io compu method.



5 Click **OK** to close the **Compu Method** Properties dialog.

Result

You defined a compu method with a unit and a physical dimension.

What's next

You will create two client server interfaces in the next step (refer to Step 3: How to Create Client Server Interfaces on page 101).

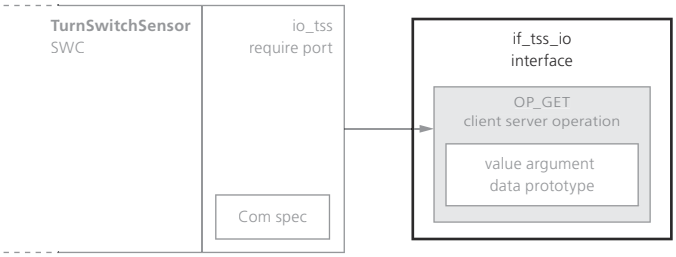
Step 3: How to Create Client Server Interfaces

Objective

In this step, you will create two client server interfaces:

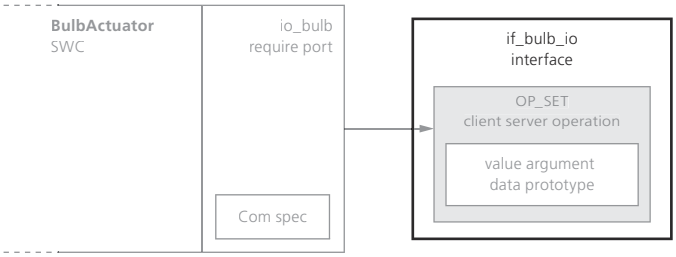
- if_tss_io for the io_tss port of the TurnSwitchSensor SWC.

  This client server interface is used later on to get the position of the turn switch sensor from the sensors.



- if_bulb_io for the io_bulb port of the BulbActuator SWC.

  It will be used to activate the bulb via an operation provided by the actuators.



Method

To create client server interfaces

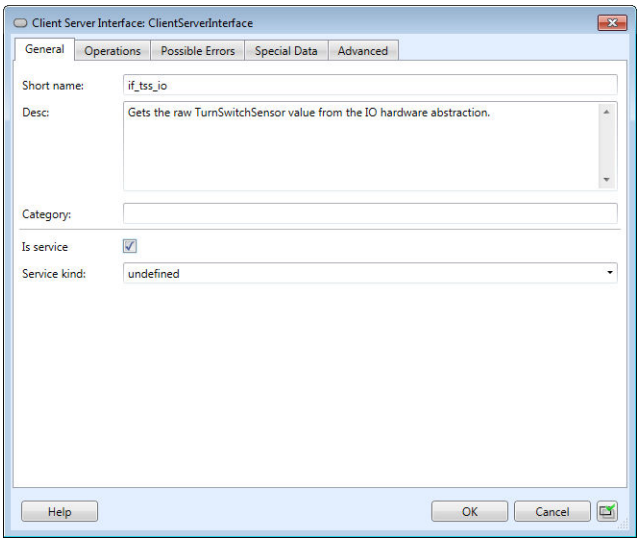
1. In the Project Manager, right-click the SharedElements/PortInterfaces package.
2. From its context menu, select New – Client Server Interface.

   The Project Manager displays a new interface as a child of the package. It will become the interface defining the communication between the TurnSwitchSensor software component and the sensors.
3. In the Project Manager, double-click the new interface.

   The Client Server Interface Properties dialog opens.
4. Change the element's default Short name to `if_tss_io`.
5. In the Desc edit field, type `Gets the raw TurnSwitchSensor value from the IO hardware abstraction.`

6 Select the Is service checkbox.

The Is service property indicates that the interface is used to connect to a basic software service.



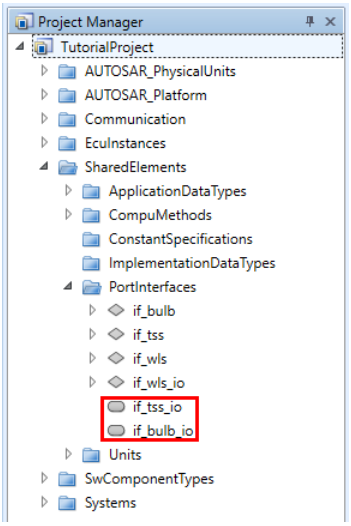
7 Click OK to confirm your settings and close the dialog.

8 Repeat steps 1 … 7 to create another client server interface for the communication between the BulbAcutator SWC and the actuators.

- Name the interface `if_bulb_io`.
- Type `Sets the bulb value using the IO hardware abstraction.` in the Desc edit field.
- Select the Is service checkbox.

Result

You created two client server interfaces for communication with the sensors/actuators.



What's next

In the next step, you will define client server operations for all client server interfaces, and you will add argument data prototypes to the client server operations.

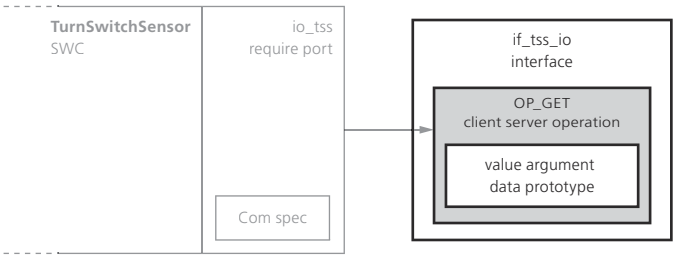
Step 4: How to Add and Define Client Server Operations

Objective

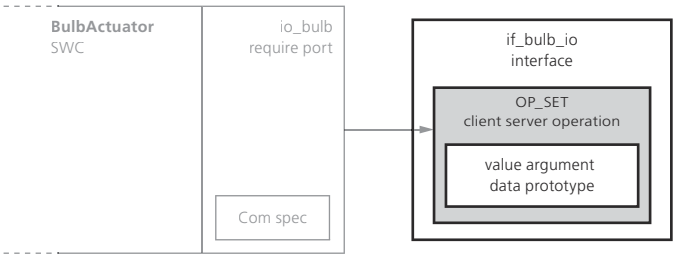
Client server interfaces define client server operations, which are provided by a server and can be used by a client.

In this step, you will first define a client server operation for each client server interface:

- OP_GET which is used by the TurnSwitchSensor SWC to get the position of the indicator switch



- OP_SET, by which a prototype of the BulbActuator SWC can switch the light of the bulb on and off



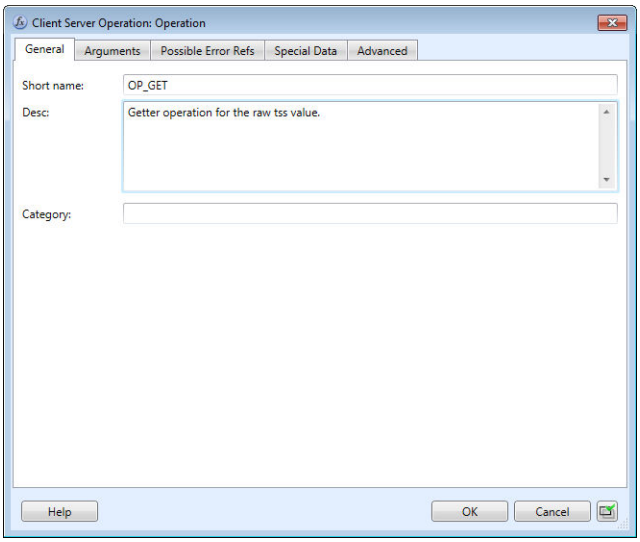
For both client server operations, you will define argument data prototypes which are passed to or returned from the operation.

Method

To add and define client server operations

1. In the Project Manager, double-click the if_tss_io client server interface.
   The Client Server Interface Properties dialog opens.
2. On the Operations page, select New.
   A new operation is displayed in the list.
3. Double-click the new operation.
   The Client Server Operation Properties dialog opens.

4 On the General page, rename the client server operation to `OP_GET` and type `Getter operation for the raw tss value.` in the Desc edit field.

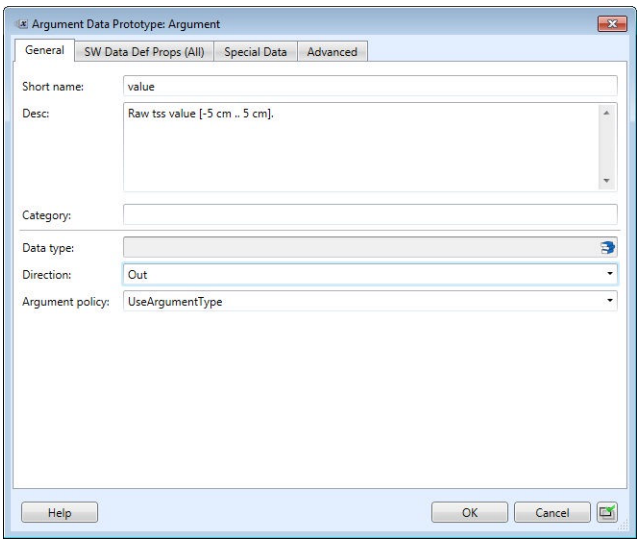


5 On the Arguments page, select New.
A new argument is displayed in the list.

6 Double-click the new argument.
The Argument Data Prototype dialog opens.

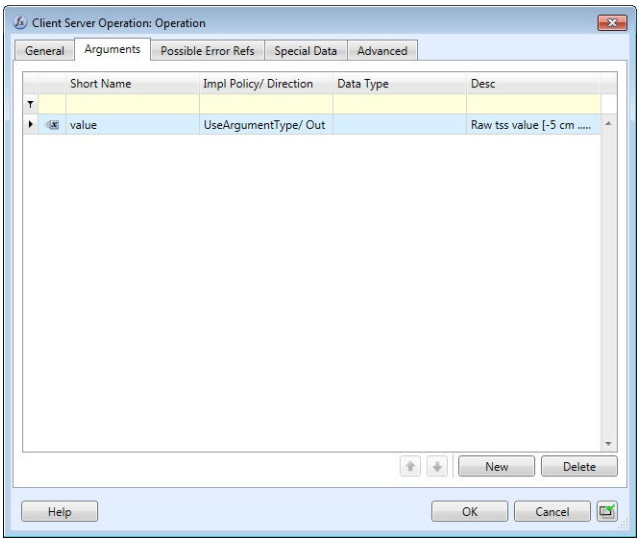
7 On the General page, specify the settings as follows:

Setting	Entry / Selection
Short name	`value`
Desc	`Raw tss value [-5 cm .. 5 cm].`
Direction	`Out`

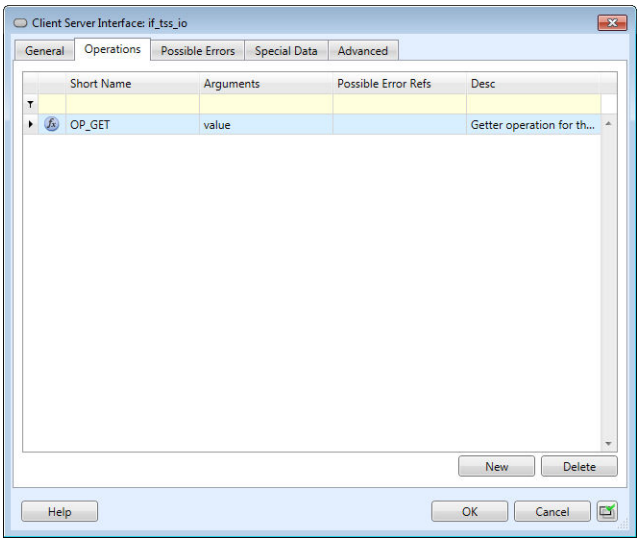


The direction indicates that the argument data prototype is delivered from the client server operation of the server to the client.

8 Click OK to confirm your settings and return to the Client Server Operation dialog.



9 Click OK to close the Client Server Operation dialog and return to the Client Server Interface dialog.



10 Click OK to close the Client Server Interface dialog.

11 Repeat steps 1 … 10 to define a client server operation and argument data prototype for the if_bulb_io interface. Name the client server operation `OP_SET` and type `Setter operation for the bulb value.` in the Desc edit field.

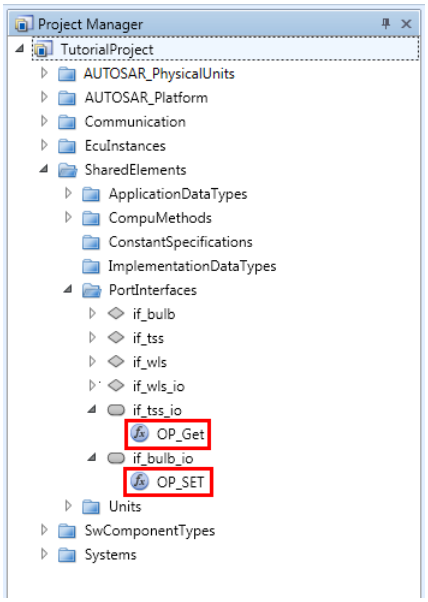
Specify the settings of the argument data prototype as follows:

Setting	Entry / Selection
Short name	`value`
Desc	`Value of the bulb signal [off/on].`
Direction	`In`

The direction indicates that the argument data prototype is delivered from the client as input to the client server operation of the server.

Result

You defined the client server operations with their argument data prototypes for the client server interfaces.



What's next

You will assign a compu method to a data type, and assign the data type to an argument data prototype in the next step.

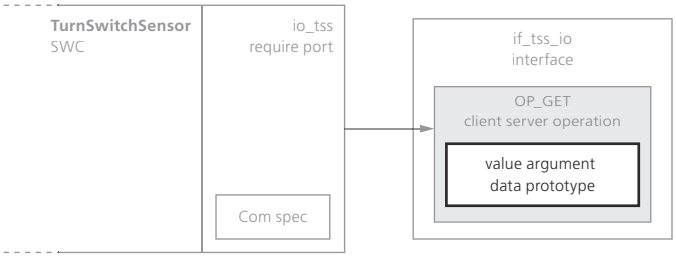
Step 5: How to Assign Compu Methods to Data Types, and Data Types to Argument Data Prototypes

Objective

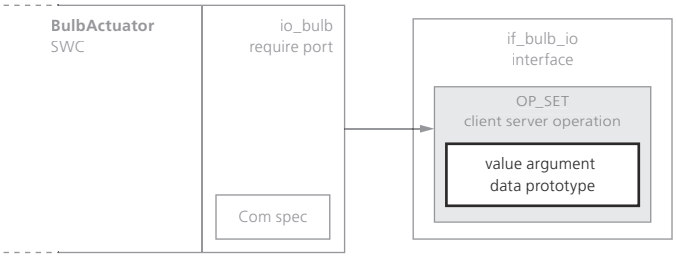
In this step, you will assign the following:

- The cm_tss_io compu method to the adt_tss_io data type to finish its definition. Refer to Part 1 on page 107.

- The adt_tss_io data type to the value argument data prototype of the if_tss_io interface. Refer to Part 2 on page 108.



- The adt_bulb data type to the value argument data prototype of the if_bulb_io interface. Refer to Part 2 on page 108.



Part 1

To assign a compu method to a data type

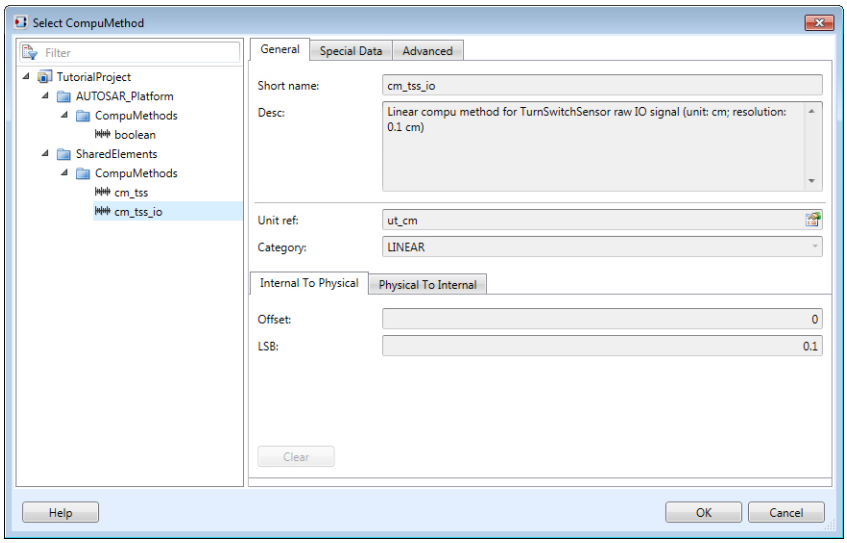
1 In the Project Manager, double-click the adt_tss_io application primitive data type.

The Application Primitive Data Type Properties dialog opens.

2 On the General page, click the button in the Compu Method field.

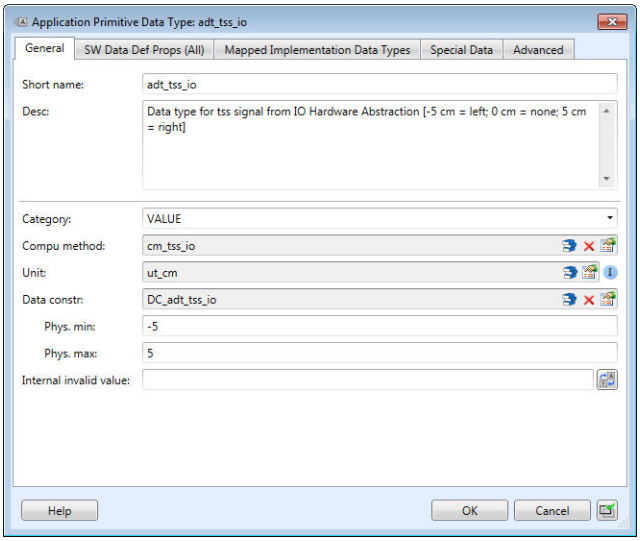
The Select Compu Method dialog opens.

3 Select cm_tss_io from the tree view.



4 Click OK to return to the Application Primitive Data Type Properties dialog.

The cm_tss_io compu method is associated with the adt_tss_io data type, and ut_cm is displayed as the unit of the data type.



5 Click OK to close the Application Primitive Data Type Properties dialog.

Interim result

You assigned the cm_tss_io compu method to the adt_tss_io data type, which completed the definition of the adt_tss_io data type.

You will now assign the adt_tss_io data type to the value argument data prototype of the if_tss_io interface. You will also add the adt_bulb data type to the value argument data prototype of the if_bulb_io interface.

Part 2

To select data types for argument data prototypes

1 In the Project Manager, double-click the OP_GET operation of the if_tss_io interface.

The Client Server Operation Properties dialog opens.

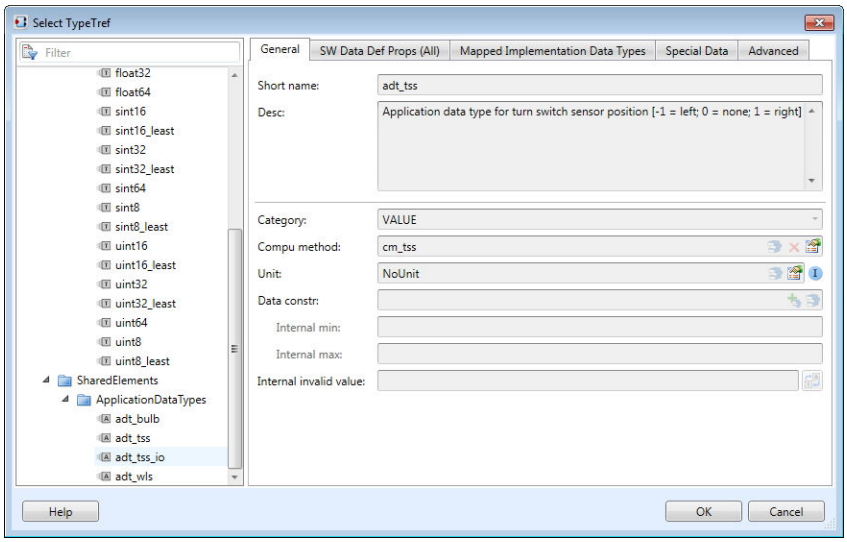
2 On the Arguments page, double-click the value argument data prototype.

The Argument Data Prototype Properties dialog opens.

3 Click the button in the Data type field.

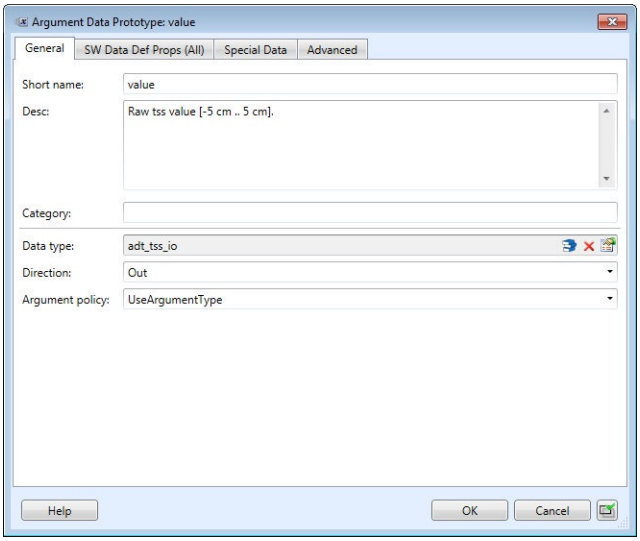
The Select TypeTref dialog opens.

4 Select the adt_tss_io data type.



5 Click OK to return to the Argument Data Prototype Properties dialog.

The adt_tss_io data type is associated with the value argument data prototype.



6 Click OK to close the Argument Data Prototype Properties dialog and return to the Client Server Operation properties dialog.

7 Click OK to close the Client Server Operation Properties dialog.

8 Add the adt_bulb data type to the value argument data prototype (if_bulb_io/OP_Set) as described in steps 1 … 7.

Result

You finished the definition of the client server interfaces.

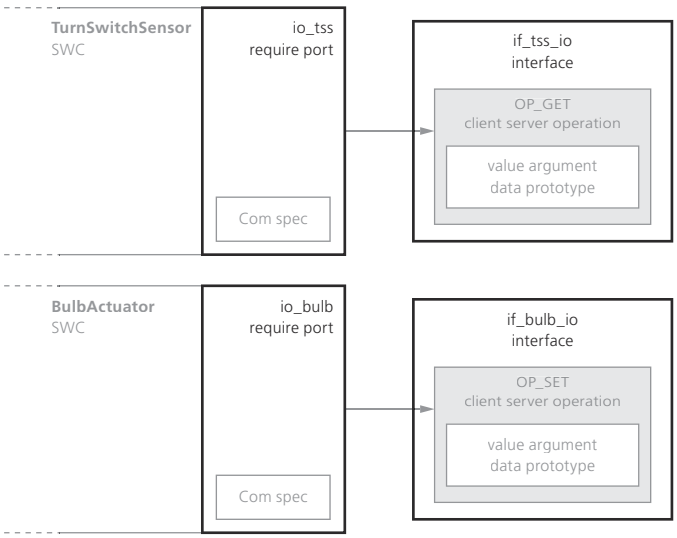
What's next

You will assign the interfaces to the ports in the next step.

Step 6: How to Assign Client Server Interfaces to Ports

Objective

You will now assign the client server interfaces to the ports of the TurnSwitchSensor and BulbActuator software components.



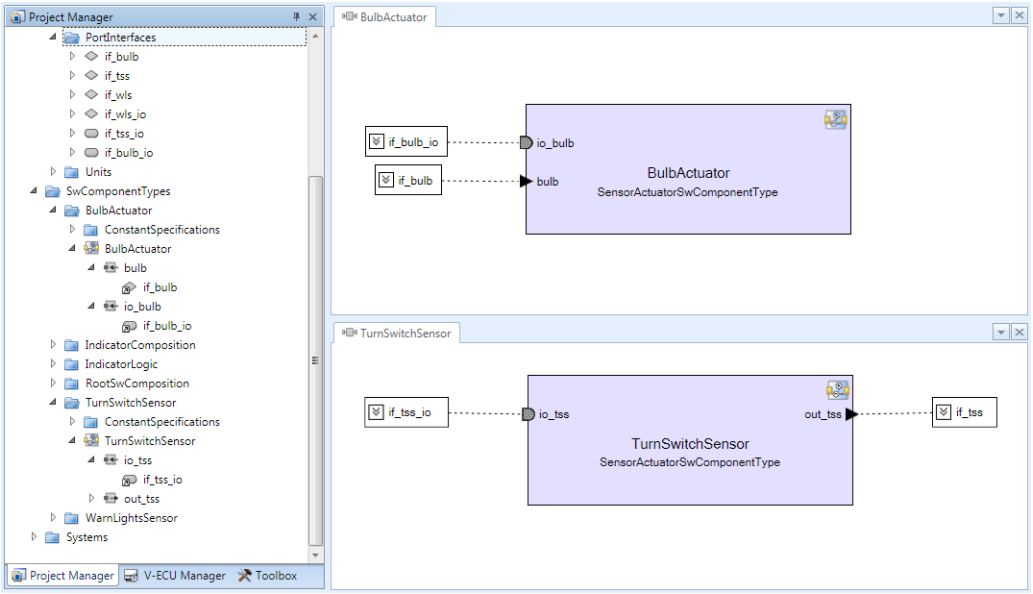
Method

To assign client server interfaces to ports

1. In the Project Manager, right-click the TurnSwitchSensor software component.
2. From the element's context menu, select Open Component Diagram.
   The element's component diagram opens in SystemDesk's working area.
3. Drag the if_tss_io interface from the Project Manager to the io_tss port of the TurnSwitchSensor in the diagram.
   The interface is added to the port and displayed in the Project Manager as a child of the port. It has an additional symbol at the bottom left, indicating a reference to the definition of the interface in the Interfaces package of the Project Manager.
4. Repeat steps 1 … 3 to assign the if_bulb_io interface to the io_bulb port of the BulbActuator SWC.

Result

You assigned the client server interfaces to the ports.



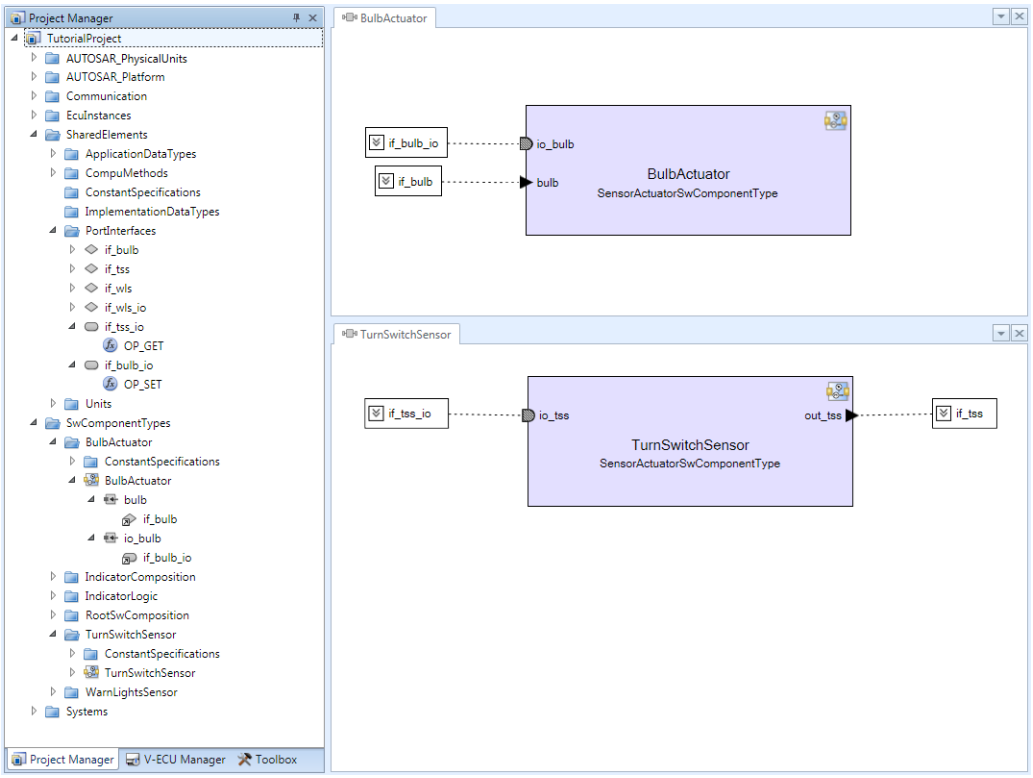
What's next

In this lesson, you modeled the client server interfaces of software components. Now check your work against the result you should have achieved. Refer to Result of Lesson 4.

Result of Lesson 4

Summary

If you have done all the steps in this lesson, your SystemDesk project should look like this:



In this lesson, you learned how to model client server interfaces. To define client server communication, you added and specified client server operations with argument data prototypes to use as arguments for the client server operation. To specify the argument data prototype, you created and added an application data type with a compu method, a data constr element and a unit with a physical dimension to it. You assigned the interfaces to ports.

Ports with *Is service* enabled are grayed out.

Demo

You can reproduce the result of this lesson with the demo script `Lesson04_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information

- For detailed information on working with AR elements like interfaces, refer to Working with AUTOSAR Elements (🕮 SystemDesk Manual).

- For detailed information on working with the composition and component diagrams, refer to Working with Diagrams (📖 SystemDesk Manual).

What's next

The next lesson shows you how to model the SWC internal behavior of atomic software components.

Lesson 5: Modeling the SWC Internal Behavior of Atomic Software Components

Where to go from here

Information in this section

Overview of Lesson 5 116
In this lesson you will define the SWC internal behavior for a software component.

Step 1: How to Add an SWC Internal Behavior 119
You will now add an SWC internal behavior to a software component.

Step 2: How to Create Runnable Entities 120
In this step, you will create runnable entities.

Step 3: How to Trigger Runnable Entities with RTE Events 123
You will now create RTE events and assign them to runnable entities.

Step 4: How to Create Interrunnable Variables 127
In this step, you will create the interrunnable variables for a software component.

Step 5: How to Specify Data Access by Runnable Entities 134
In this step, you will specify the data accesses to variable data prototypes and interrunnable variables.

Result of Lesson 5 138
In this lesson, you learned how to model the SWC internal behavior of atomic software components.

Overview of Lesson 5

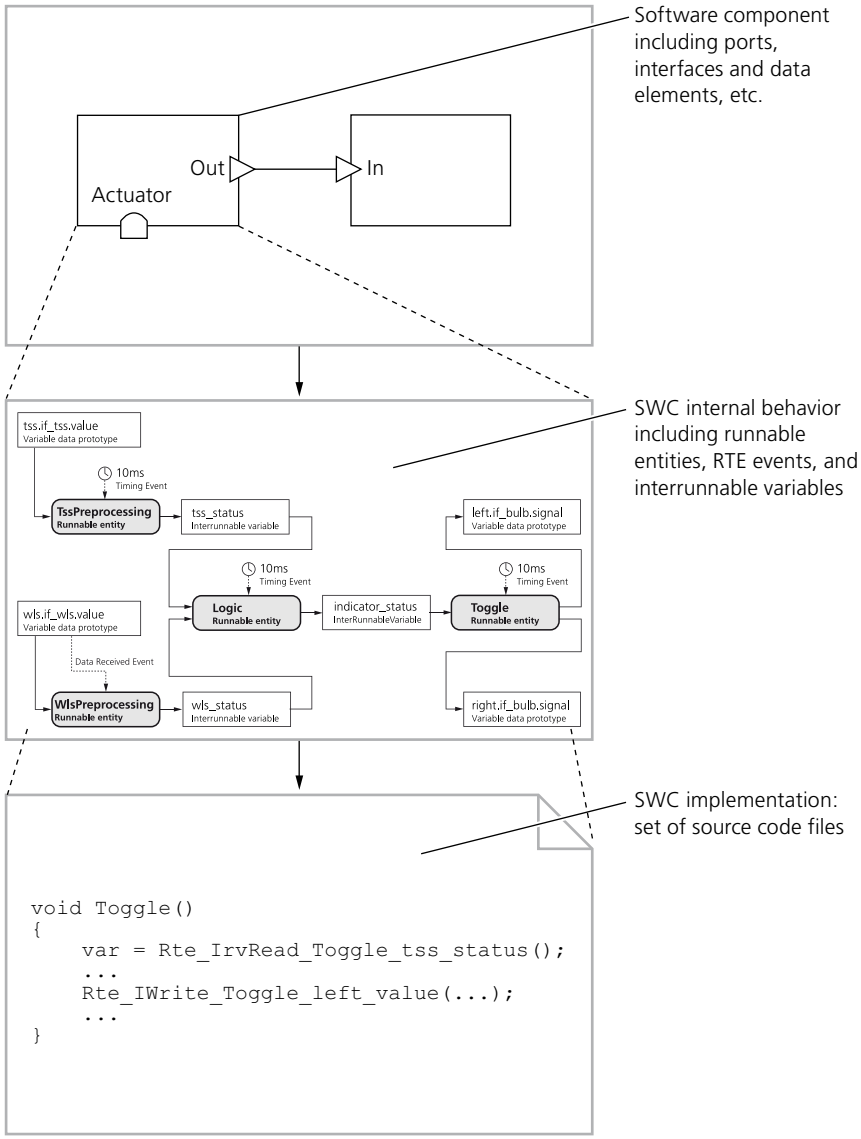
Starting point

In the previous lesson, you learned how to specify communication via client server interfaces. You created client server interfaces, specified the client server operations and their argument data prototypes, and assigned client server interfaces to ports.

What will you learn?

In this lesson, you will learn how to model the SWC internal behavior of atomic software components.

SWC internal behavior An element that represents the internal structure of an atomic software component. It is characterized by elements such as runnable entities, RTE events, data access definitions, interrunnable variables, and their interdependencies. An SWC internal behavior is refined by an SWC implementation that describes implementation details such as code files or a given platform (microprocessor type and compiler).



As shown in the illustration above, an SWC internal behavior is the internal structure of an atomic software component. An SWC is divided into runnable entities which represent C functions. RTE events define when runnable entities are triggered.

Data accesses define a runnable entity's read or write access to communication data. A runnable entity can access communication data that is transferred between SWCs via sender receiver or client server interfaces. In addition a runnable entity can access interrunnable variables that specify data that is transferred between runnable entities of one SWC.

An SWC implementation contains implementation details such as the code files that implement the runnable entities of an SWC internal behavior.

Modeling an SWC internal behavior To model an SWC internal behavior of the IndicatorLogic application SW component type in this lesson, you will:

- Create an SWC internal behavior.
- Create several runnable entities for the SWC internal behavior.
- Trigger the runnable entities you created with RTE events.
- Create interrunnable variables.
- Specify data access by runnable entities, i.e., access to interrunnable variables and variable data prototypes defined in interfaces.
- Integrate an SWC implementation.

Note

In this lesson you will define the SWC internal behavior for the IndicatorLogic software component.
You will import the SWC internal behavior of the BulbActuator SWC in the next lesson. The demo script `Lesson05_complete.py` reproduces a complete SWC internal behavior model of all the SWCs.

You can use the project as the basis for the following lessons.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson04_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Add an SWC Internal Behavior on page 119
- Step 2: How to Create Runnable Entities on page 120
- Step 3: How to Trigger Runnable Entities with RTE Events on page 123
- Step 4: How to Create Interrunnable Variables on page 127
- Step 5: How to Specify Data Access by Runnable Entities on page 134

Summary

To check if the steps you executed are correct, refer to Result of Lesson 5 on page 138.

Step 1: How to Add an SWC Internal Behavior

Objective

You will now add an SWC internal behavior to the IndicatorLogic atomic software component. You will use this SWC internal behavior later on for configuring runnable entities and RTE events.

Method

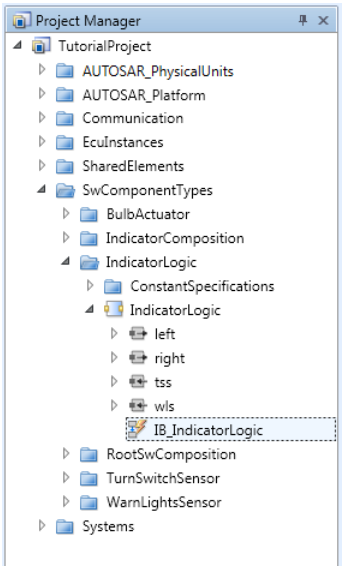
To add an SWC internal behavior

1. In the Project Manager, right-click the IndicatorLogic software component in the IndicatorLogic package.
2. From the context menu, select New – SWC Internal Behavior.

   The Project Manager displays a new empty SWC internal behavior as a child of the atomic SWC.
3. Change the element's default name to `IB_IndicatorLogic`.

Result

You added an empty SWC internal behavior to the IndicatorLogic atomic SWC.



What's next

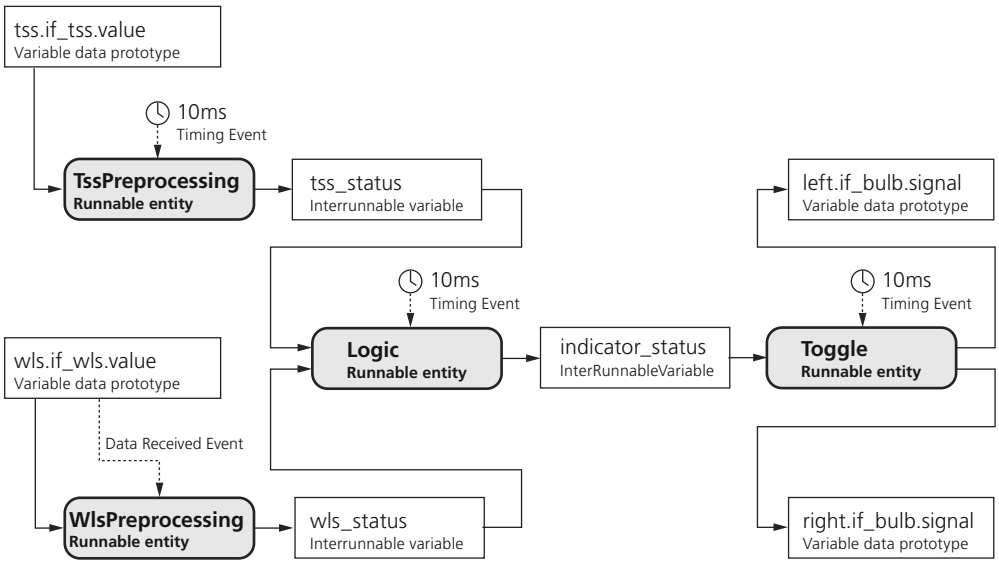
You can now add runnable entities to the SWC internal behavior and specify their properties in the next step.

Step 2: How to Create Runnable Entities

Objective

Runnable entities are parts of the SWC internal behavior. They correspond to functions in the SWC implementation. They are therefore the smallest units provided by the software component which can be triggered and executed by the run-time environment.

The following illustration is a schematic view of the SWC internal behavior including the runnable entities of the IndicatorLogic atomic SWC.



The TssPreprocessing and WlsPreprocessing runnable entities receive the sensor input from the ports they are connected to, and preprocess the data. Then they relay the data to the Logic runnable entity.

The Logic runnable entity computes the operation to be performed. It relays the data to the Toggle runnable entity.

The Toggle runnable entity computes the signal for the lamps, for example, to turn them on and off.

In this step, you will create these runnable entities.

Method

To create runnable entities

1. In the Project Manager, double-click the IB_IndicatorLogic SWC internal behavior.

   The SWC Internal Behavior Properties dialog opens.

2. On the Runnables page, click New.

   A new runnable is displayed in the list.

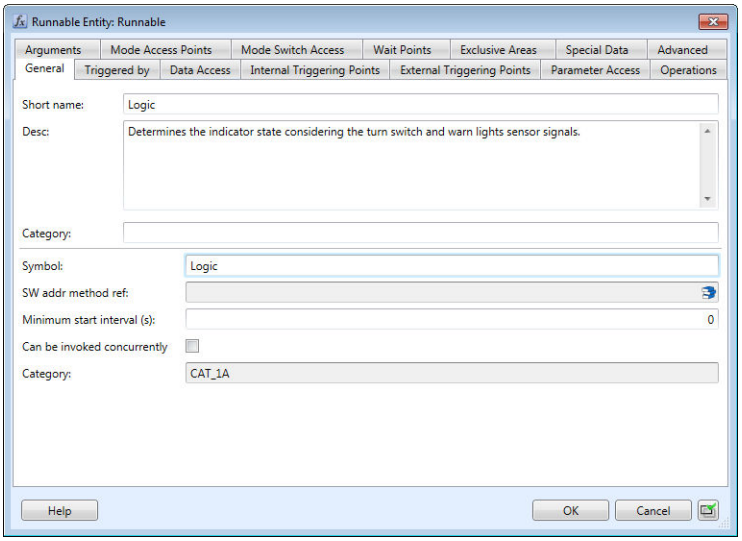
3. Double-click the new runnable.

   The Runnable Entity Properties dialog opens.

4 On the General page, specify the settings as follows:

Setting	Entry
Short name	`Logic`
Desc	`Determines the indicator state considering the turn switch and warn lights sensor signals.`
Symbol	`Logic`

The Symbol property describes the runnable entity's name in the C code implementation. It is internally used by the RTE to call the runnable entity.

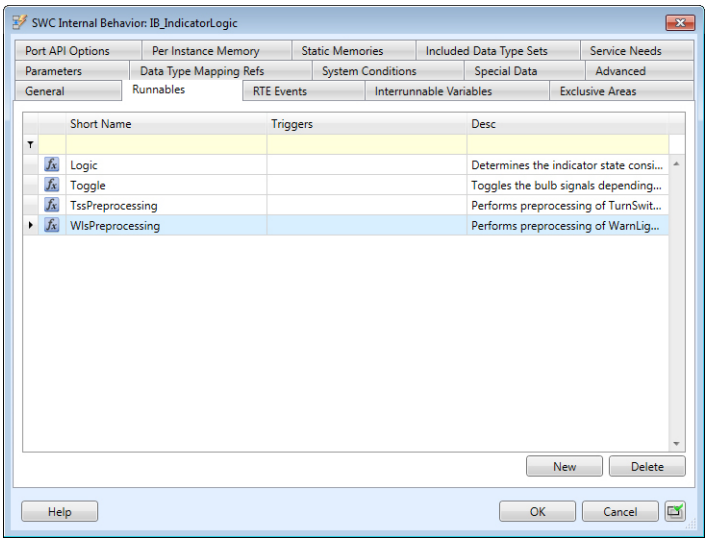


5 Click OK to confirm your settings and close the Runnable Entity Properties dialog.

6 Create and specify the following three runnable entities as described in steps 2 … 5:

Short Name	Desc	Symbol
Toggle	`Toggles the bulb signals depending on the indicator state.`	`Toggle`
TssPreprocessing	`Performs preprocessing of TurnSwitchSensor signals.`	`TssPreprocessing`
WlsPreprocessing	`Performs preprocessing of WarnLightsSensor signals.`	`WlsPreprocessing`

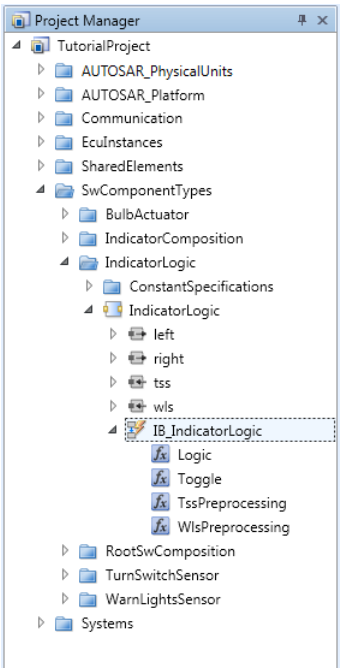
The new runnable entities are displayed on the **Runnables** page of the **SWC Internal Behavior** properties dialog.



7 Click OK to close the **SWC Internal Behavior** Properties dialog.

Result

You created four runnable entities for the SWC internal behavior of the IndicatorLogic atomic SWC. They are displayed as children of the IB_IndicatorLogic SWC internal behavior element in the **Project Manager**.



What's next

You will create RTE events to trigger runnable entities in the next step.

Step 3: How to Trigger Runnable Entities with RTE Events

Objective

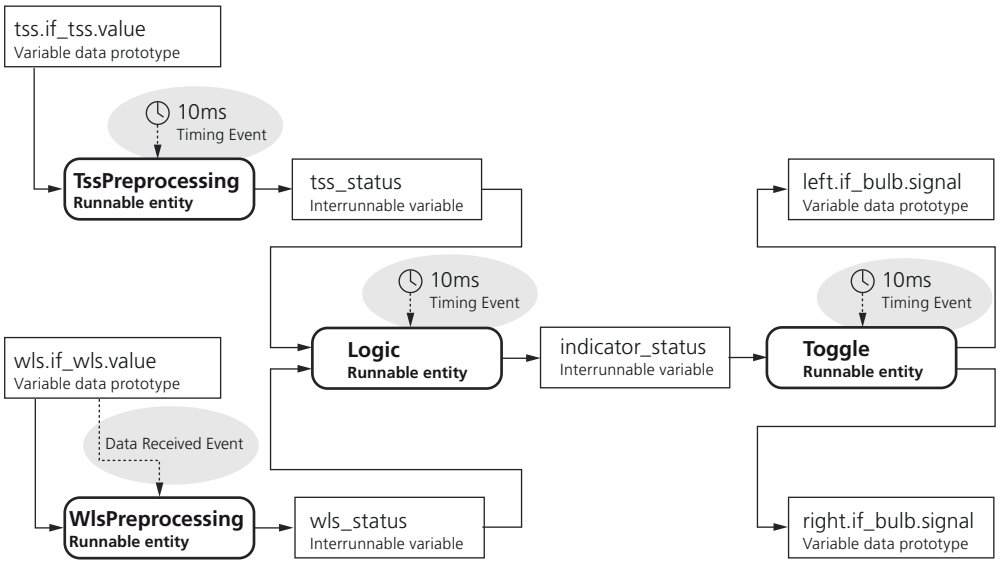
The execution of a runnable entity is triggered by an RTE event.

You will use two types of RTE events:

Timing Event An event that triggers the runnable entity periodically.

Data Received Event An event that triggers the runnable entity as soon as a variable data prototype is received at the port.

The following illustration again shows you the SWC internal behavior of the IndicatorLogic SWC: The RTE events which trigger the runnable entities are highlighted.



The timing events for the TssPreprocessing, the Logic and the Toggle runnable entities will be triggered every 10ms.

The WlsPreprocessing runnable entity will be triggered by a data received event of the value variable data prototype at the IndicatorLogic SWC's wls port. Data received events are typically used with queued data transfer.

You will now create RTE events and assign them to the runnable entities you created in the previous step.

Method

To trigger runnable entities with RTE events

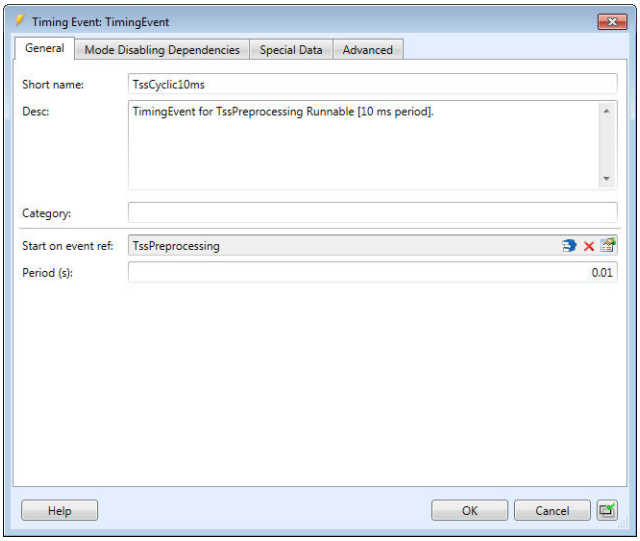
1. In the Project Manager, open the SWC Internal Behavior Properties dialog of the IB_IndicatorLogic SWC internal behavior.
2. On the RTE Events page, select Timing Event and click New.
   A new timing event is displayed on the RTE Events page.
3. Double-click the new timing event.
   The Timing Event Properties dialog opens.

4 On the General page, specify the settings as follows:

Setting	Entry
Short name	TssCyclic10ms
Desc	TimingEvent for TssPreprocessing Runnable [10 ms period].
Period (s)	0.01

5 In the Start on event ref field, click .
The Select RunnableEntity dialog opens.

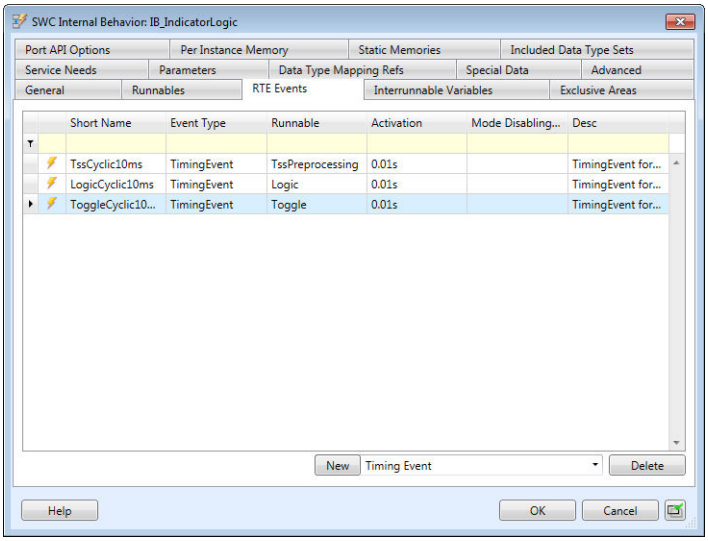
6 Select TssPreprocessing and click OK to return to the Timing Event dialog.



7 Click OK to close the dialog and return to the SWC Internal Behavior dialog.

8 Repeat steps 2 … 7 to create two more timing events:

Short Name	Desc	Start on Event Ref	Period
LogicCyclic10ms	`TimingEvent for Logic Runnable [10 ms period].`	Logic	0.01
ToggleCyclic10ms	`TimingEvent for Toggle Runnable [10 ms period].`	Toggle	0.01

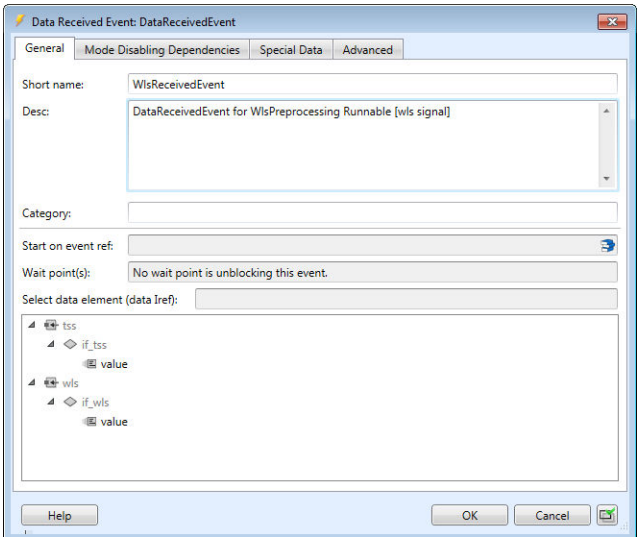


9 To create a data received event, select Data Received Event on the RTE Events page of the SWC Internal Behavior Properties dialog and click New.

10 Double-click the new data received event on the RTE Events page.

The Data Received Event Properties dialog opens.

11 On the General page, enter `WlsReceivedEvent` as the Short name and type `DataReceivedEvent for WlsPreprocessing Runnable [wls signal]` in the Desc edit field.

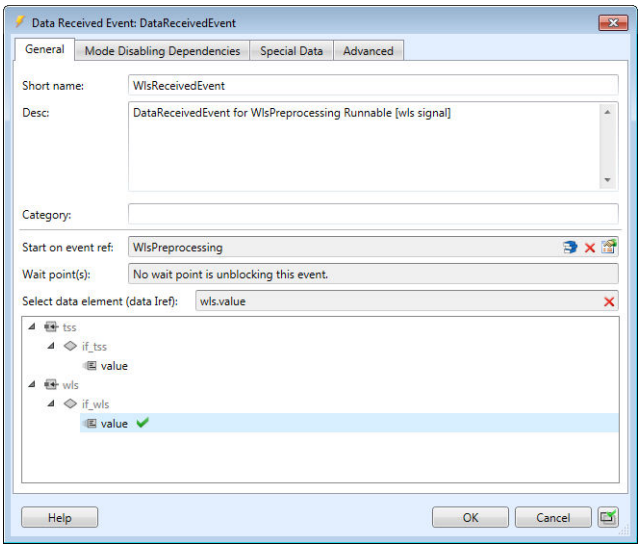


12 In the Start on event ref field, click .

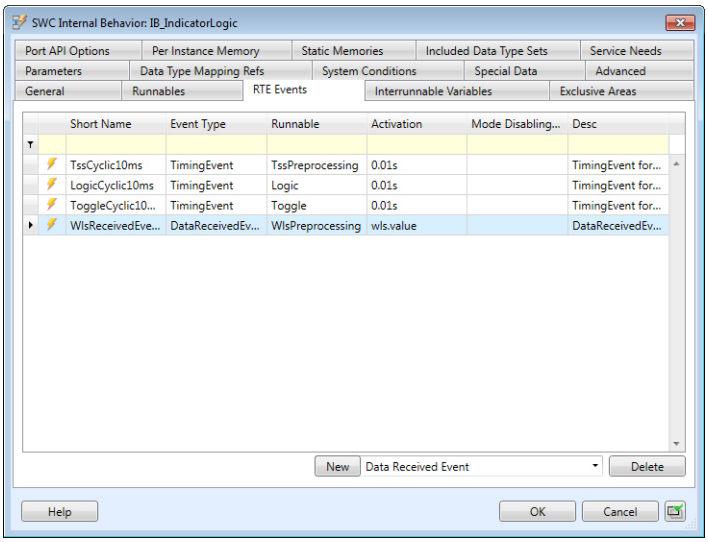
The Select RunnableEntity dialog opens.

13 Select WlsPreprocessing element and click OK to return to the Data Received Event dialog.

14 In the Select data element (data Iref) tree, select the wls – if_wls – value element.



15 Click OK to close the dialog and return to the SWC Internal Behavior dialog.



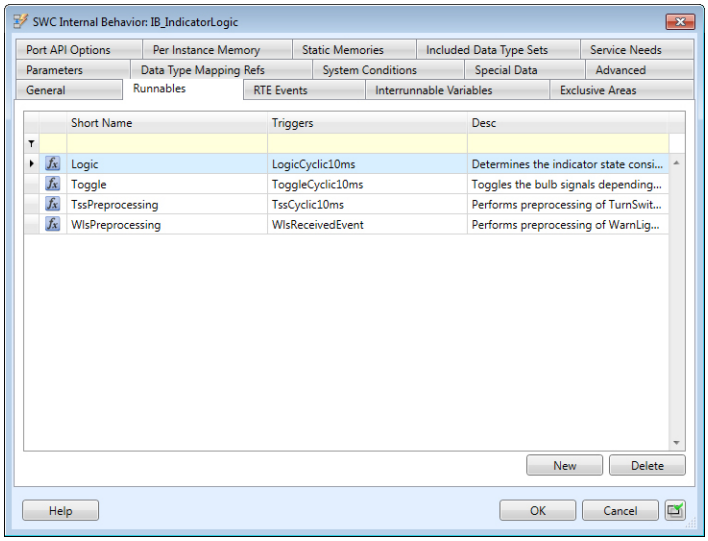
16 Click OK to close the dialog.

Result

You created RTE events which will trigger your runnable entities. The timing events will trigger the runnable entities TssPreprocessing, Logic and Toggle every 10 ms. The WlsPreprocessing runnable entity is triggered as soon as a value variable data prototype is received.

You will now learn how to assign the RTE events to the runnable entities in order to specify which runnable entity should be triggered by which RTE event.

You can check your work on the **Runnables** page of the **SWC Internal Behavior** Properties dialog, which lists all the runnable entities and the assigned RTE events.



What's next

You will create interrunnable variables for the communication between runnable entities in the next step.

Step 4: How to Create Interrunnable Variables

Objective

In addition to sender receiver and client server communication, which is used for communication across SWCs, interrunnable communication is used for communication between runnables of the same SWC. The runnable entities have access to the same interrunnable variable. The interrunnable variable is a variable data prototype that requires an application data type which can have a compu method and a data constr element like the variable data prototype of a sender receiver interface.

Interrunnable variables can be:

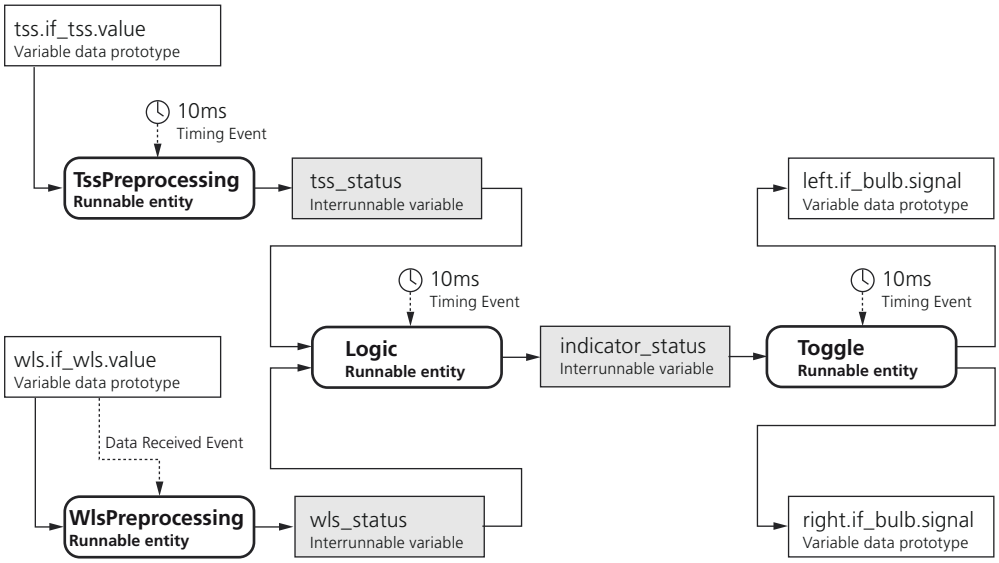
- *Implicit*, which means the runnable entity works on a local copy of an interrunnable variable.

  A copy of read interrunnable variables is created on entry of the runnable. Changes are written back when the runnable entity completes. This avoids concurrent access to interrunnable variables during runnable entity execution.
- *Explicit*, which means runnable entities can directly access an interrunnable variable.

  Changes are immediately visible to other runnable entities with explicit access to the interrunnable variable.

In this step, you will create the interrunnable variables for the IndicatorLogic software component.

The following illustration shows which interrunnable variables are used for communication between the runnable entities.



Creating interrunnable variables

To create interrunnable variables, you have to perform the following steps:

1. Create an application data type and an associated compu method. Refer to Part 1 on page 128.
2. Create an interrunnable variable and add the application data type. Refer to Part 2 on page 130.
3. Edit the constant specification associated with the interrunnable variable. Refer to Part 3 on page 133.

Part 1

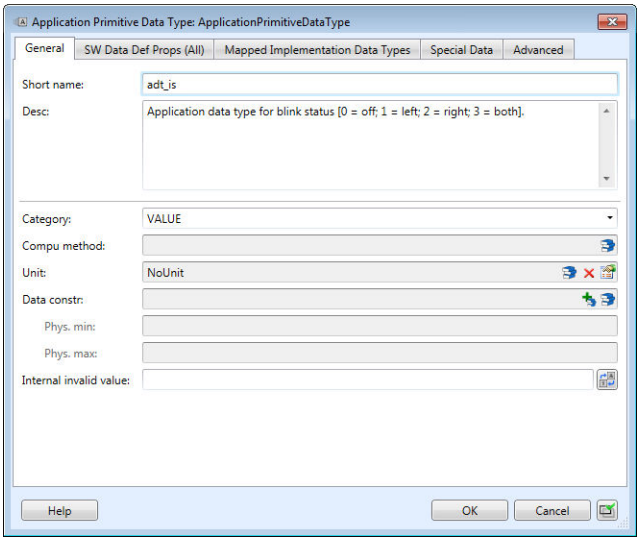
To create an application data type and associated compu method

1 In the Project Manager, select SwComponentTypes – IndicatorLogic and add two new packages with the name `ApplicationDataTypes` and `CompuMethods`.

2 In the ApplicationDataTypes package, create a new application primitive data type with the following settings:

Setting	Entry / Selection
General Page	
Short name	`adt_is`
Desc	`Application data type for blink status [0 = off; 1 = left; 2 = right; 3 = both].`
Category	`VALUE`
Unit	`NoUnit`

Setting	Entry / Selection
SW Data Def Props (All) Page	
SW Calibration access	`NotAccessible`



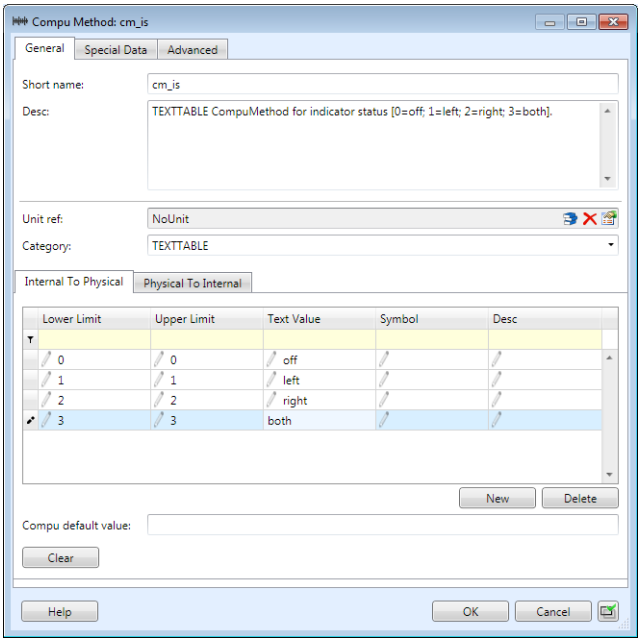
3 In the CompuMethods package, create a new compu method.

4 On the General page of the new compu method's Properties dialog, specify the settings as follows:

Setting	Entry / Selection
Short name	`cm_is`
Desc	`TEXTTABLE CompuMethod for indicator status [0=off; 1=left; 2=right; 3=both].`
Unit ref	`NoUnit`
Category	`TEXTTABLE`

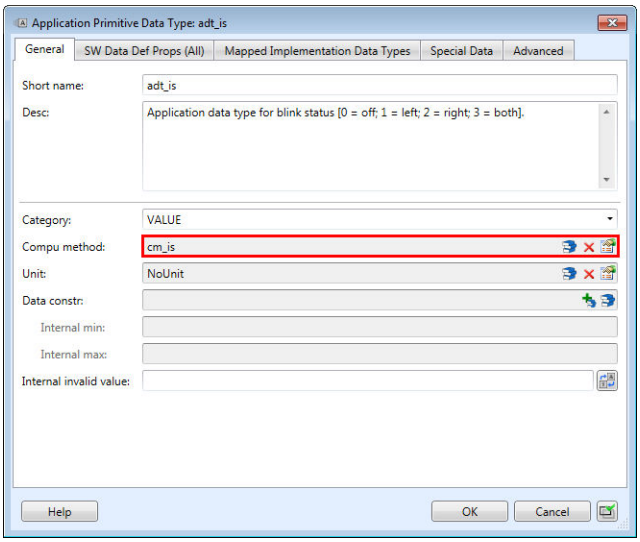
To enter the conversion definition, click New on the Internal to Physical page. Fill the table as follows:

Lower Limit	Upper Limit	Text Value
0	0	`off`
1	1	`left`
2	2	`right`
3	3	`both`



5 Click OK to close the dialog.

6 On the General page of the adt_is application primitive data type's Properties dialog, assign the cm_is compu method to the adt_is data type.



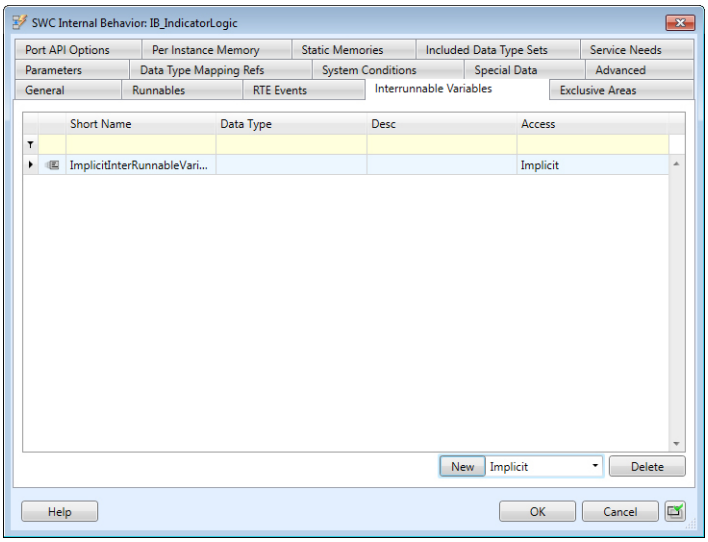
Part 2

To create interrunnable variables

1 In the Project Manager, open the SWC Internal Behavior Properties dialog of the IB_indicatorLogic SWC internal behavior.

2 On the Interrunnable Variables page, select Implicit and click New.

A new interrunnable variable is displayed on the Interrunnable Variables page.



3 Double-click the new interrunnable variable.
The Variable Data Prototype dialog opens.

4 Change the Short name to `indicator_status`.

5 In the Desc edit field, type `Blink status`, and specify the Category as VALUE.

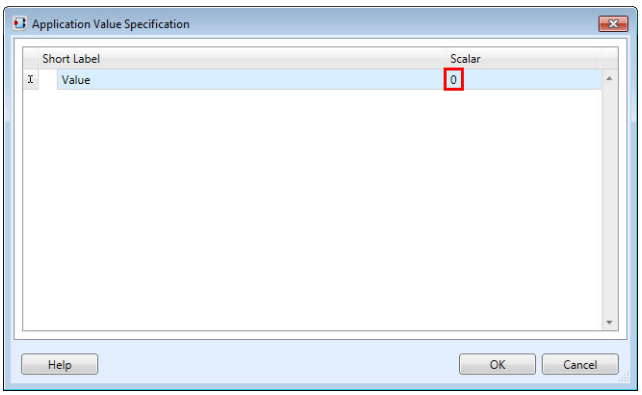
6 In the Data type field, click the button.
The Select TypeTref dialog opens.

7 Select the adt_is data type from the SwComponentTypes/IndicatorLogic/ApplicationDataTypes package.

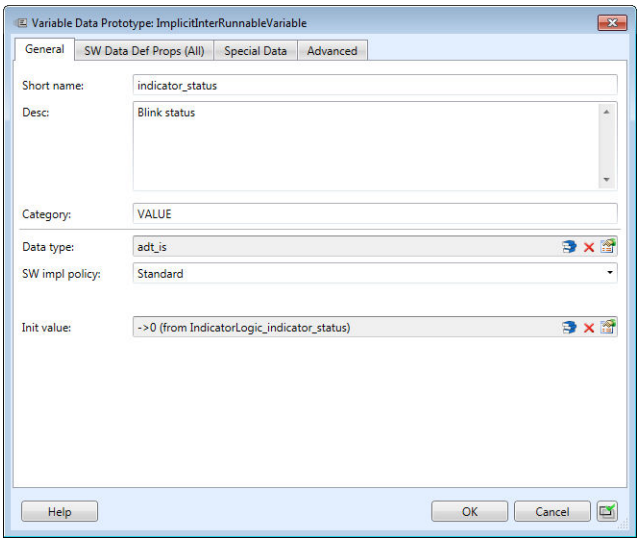
8 Click OK to return to the Variable Data Prototype Properties dialog.

9 In the Init value field, click .
The Application Value Specification dialog opens.

10 Enter `0` in the Scalar field.

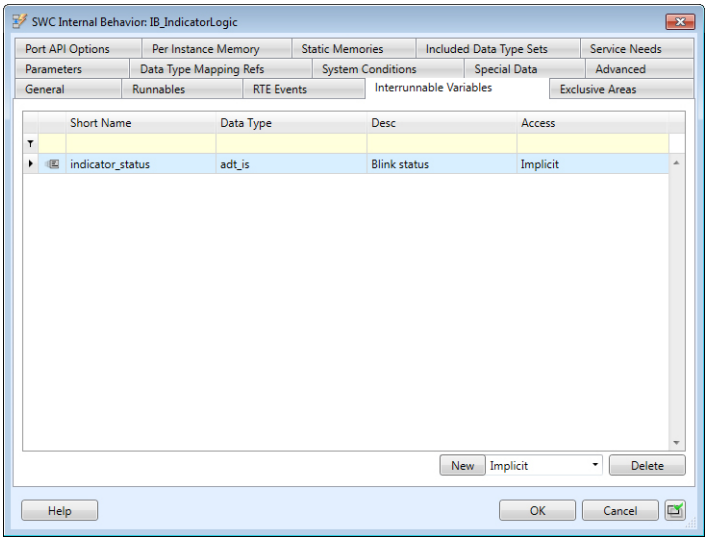


11 Click **OK** to return to the **Variable Data Prototype** Properties dialog.



12 Click **OK** to confirm your settings and close the dialog.

The new specifications of the interrunnable variable are displayed on the **Interrunnable Variables** page of the **SWC Internal Behavior** Properties dialog.

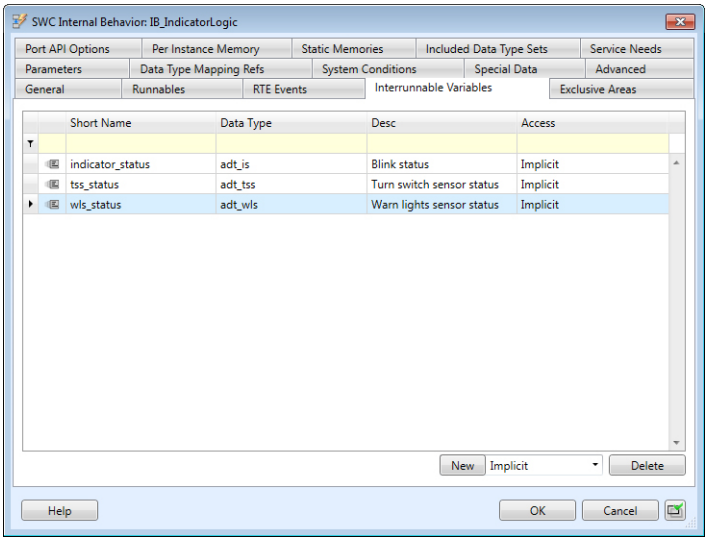


13 Click **OK** to confirm your settings and close the **SWC Internal Behavior** dialog.

SystemDesk generates a new constant **IndicatorLogic_indicator_status** in the **IndicatorLogic/ConstantSpecifications** package. You will configure the constant in Part 3.

14 Create two more implicit interrunnable variables and add the data types you specified in lesson 3 to them as follows:

Short Name	Desc	Data Type	Init Value
tss_status	`Turn switch sensor status`	adt_tss	0
wls_status	`Warn lights sensor status`	adt_wls	0

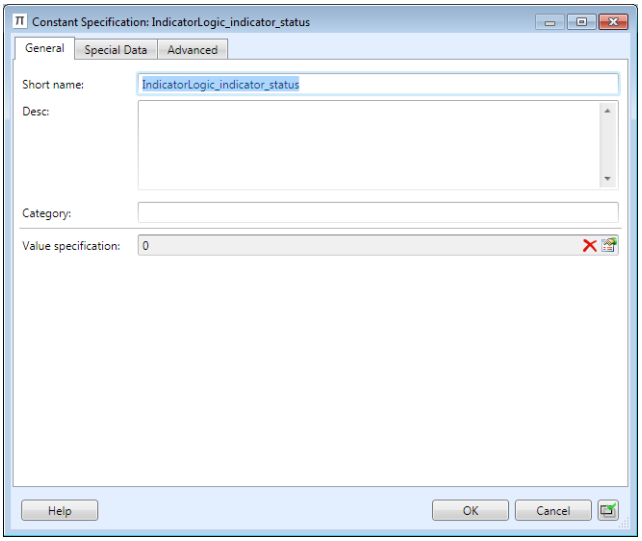


15 Click OK to close the SWC Internal Behavior Properties dialog.

Part 3

To edit the constant specifications of the interrunnable variables

1 In the IndicatorLogic/ConstantSpecifications package, double-click the IndicatorLogic_indicator_status constant specification to open its Properties dialog.



2 On the General page of the Constant Specification Properties dialog, change the Short Name to `CONST_indicator_status` and enter `Blink status` in the Desc field.

3 In the Category field, enter `VALUE`.

4 Click OK to close the Properties dialog.

5 Repeat steps 1 to 4 to edit the constant specifications of the tss_status and wls_status interrunnable variables as follows:

Constant Specification	Short Name	Desc	Category
IndicatorLogic_tss_status	CONST_tss_status	`Turn switch sensor status`	VALUE
IndicatorLogic_wls_status	CONST_wls_status	`Warn lights sensor status`	BOOLEAN

Result

You created three interrunnable variables which the runnable entities can use for communication.

What's next

You will specify which runnable entity has access to which variable data prototypes and interrunnable variables in the next step.

Step 5: How to Specify Data Access by Runnable Entities

Objective

You have to specify a runnable entity's access to communication data. Variable data prototypes of sender receiver interfaces and interrunnable variables can be accessed either *implicitly* or *explicitly*.

Access to an interrunnable variable (implicit/explicit) is defined by its prototype. You can also create data accesses to interrunnable variables.

You can specify the following accesses to variable data prototypes of sender receiver interfaces:

- Data Read/Write. Defines implicit access. Cannot be used for queued communication.

  With data read access, the runnable entity makes a local copy of the variable data prototype when it starts executing.

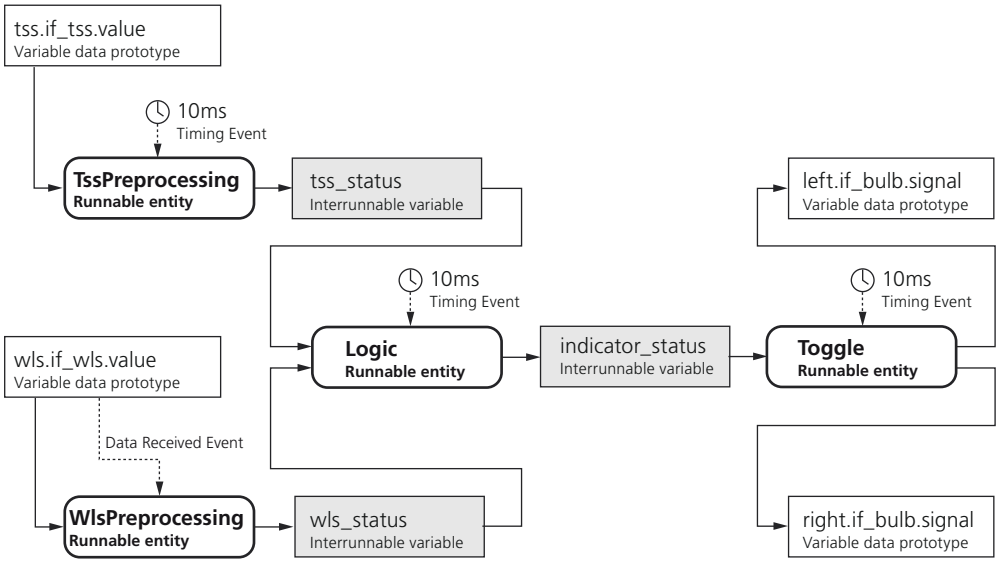
  With data write access, the local copy is made available to other SWCs when the runnable entity returns.
- Data Send/Receive. Defines explicit access. Can be used for either non-queued or queued communication.

  With data send access, changes to the variable data prototype are immediately visible to other SWCs.

  With data receive access, a variable data prototype is read every time you access it.

If you have connected the variable data prototype to a provide (require) port, SystemDesk automatically specifies the access as write/send (read/receive).

The following illustration again shows you the SWC internal behavior of the IndicatorLogic SWC: The different variable data prototypes are highlighted.



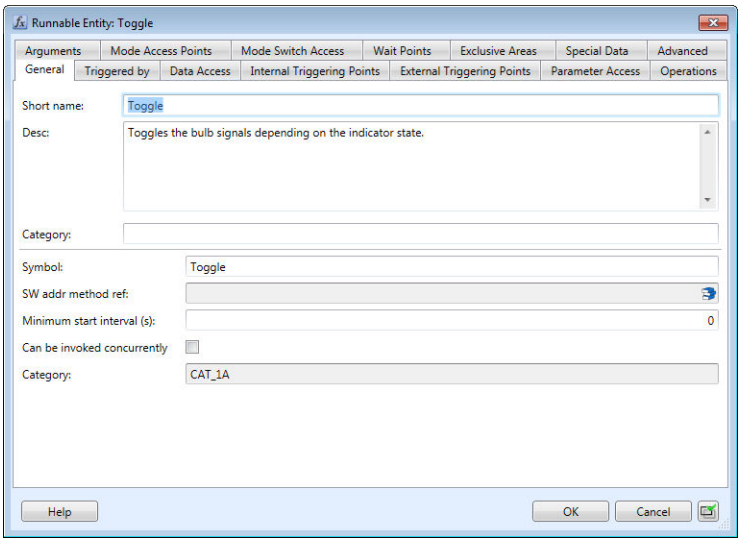
In this step, you will specify the data accesses to the variable data prototypes and interrunnable variables as indicated by the arrows, and according to the type of communication. You will define implicit access for nonqueued communication and explicit access for queued communication.

Method

To specify data access by runnable entities

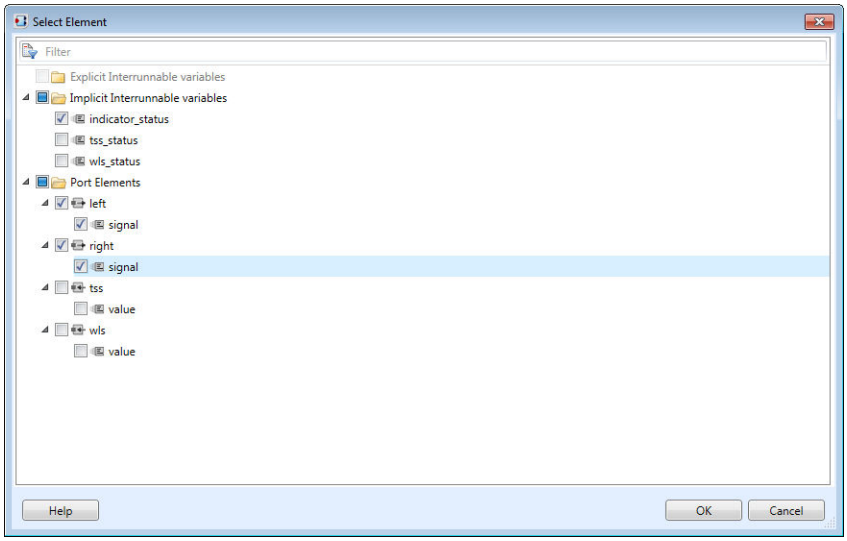
1 In the Project Manager, select SwComponentTypes – IndicatorLogic – IndicatorLogic – IB_IndicatorLogic and double-click the Toggle runnable entity.

The Runnable Entity Properties dialog opens.



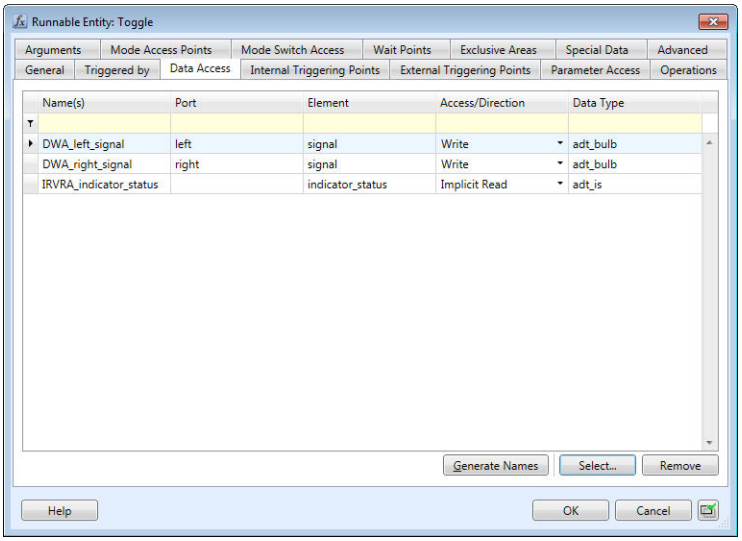
2 On the Data Access page, click Select.
The Select Element dialog opens.

3 Select the three variable data prototypes:
- Implicit Interrunnable variables – indicator_status
- Port Elements – left – signal
- Port Elements – right – signal.



4 Click OK to close the dialog.

The three variable data prototypes are displayed on the Data Access page of the Runnable Entity Properties dialog.



5 Click OK to confirm your settings and close the Runnable Entity Properties dialog.

6 Specify the data access for the three remaining runnable entities as follows:

- Logic runnable entity:

Implicit Interrunnable Variable	Access
wls_status	Implicit Read
tss_status	Implicit Read
indicator_status	Implicit Write

- WlsPreprocessing runnable entity:

Implicit Interrunnable Variable	Access
wls_status	Implicit Write

Port Element	Access
wls – value	Receive by argument

- TssPreprocessing runnable entity:

Implicit Interrunnable Variable	Access
tss_status	Implicit Write

Port Element	Access
tss – value	Read

Result

You have specified data access for the runnable entities of the IndicatorLogic SWC. You have finished modeling the SWC internal behavior.

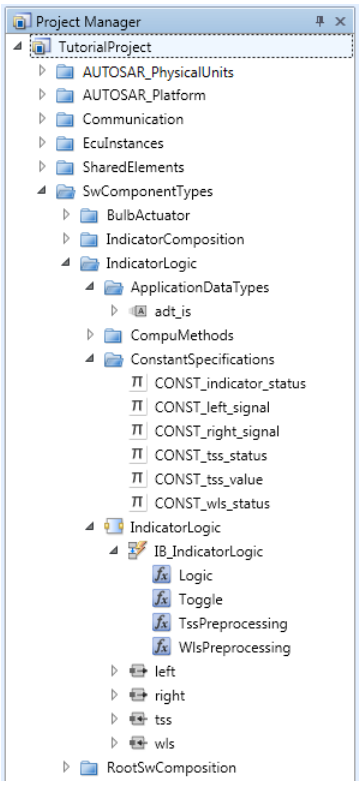
What's next

In this step, you completed the modeling of the SWC internal behavior of the IndicatorLogic SWC. Now check your work against the result you should have achieved. Refer to Result of Lesson 5 on page 138.

Result of Lesson 5

Summary

If you have done all the steps in this lesson, your IndicatorLogic SWC should look like this:

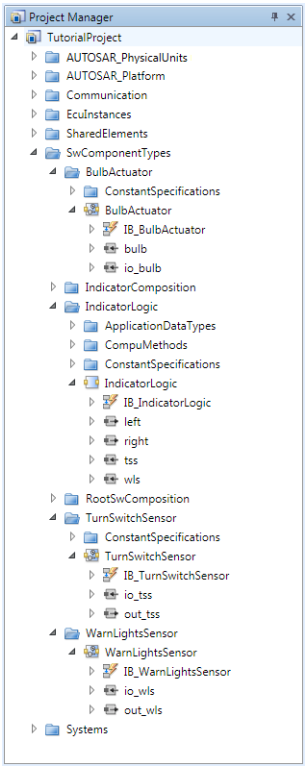


In this lesson, you learned how to model the SWC internal behavior of atomic software components. You added an SWC internal behavior to the IndicatorLogic SWC. You created runnable entities and learned how to trigger them with RTE events. To specify communication between runnable entities, you defined interrunnable variables and data accesses by runnable entities to the variable data prototypes.

Demo

You can reproduce the result of this lesson with the demo script `Lesson05_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

The **Lesson05_complete.py** demo script also adds SWC internal behaviors to the BulbActuator, TurnSwitchSensor, and WarnLightsSensor SWCs, as shown in the following illustration.



Further information

For detailed information on working with AR elements like SWC internal behaviors, refer to Working with AUTOSAR Elements (🕮 SystemDesk Manual).

What's next

The next lesson shows you how to model measurement and calibration access.

Lesson 6: Modeling Measurements and Calibration Access

Where to go from here

Information in this section

Overview of Lesson 6 141
In this lesson, you will learn how to model measurement and calibration access.

Step 1: How to Create Measurements 142
You will now specify read-only measurements for selected data elements.

Step 2: How to Create Calibration Parameters 143
You will now specify a shared parameter for an SWC internal behavior.

Result of Lesson 6 151
In this lesson, you learned how to create measurements and calibration parameters.

Overview of Lesson 6

Starting point

In the previous lessons, you learned how to model software components with their interactions and SWC internal behaviors.

What will you learn?

In this lesson, you will learn how to model measurement and calibration access.

With SystemDesk, you can specify measurement access for data elements, operation arguments, and interrunnable variables. You can also specify calibration parameters at the SWC internal behavior of an atomic software component. The RTE generation process creates measurement variables for the elements with measurement access and calibration parameters for the shared

parameters of SWC internal behaviors. This allows measurement and calibration (MC) systems such as dSPACE ControlDesk to access these variables while the ECU application is running.

In this lesson:

- You will specify read-only measurement access to data elements.
- You will specify a shared parameter for the SWC internal behavior of an atomic software component.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson05_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

It contains the following steps:

- Step 1: How to Create Measurements on page 142
- Step 2: How to Create Calibration Parameters on page 143

Summary

To check if the steps you executed are correct, refer to Result of Lesson 6 on page 151.

Step 1: How to Create Measurements

Objective

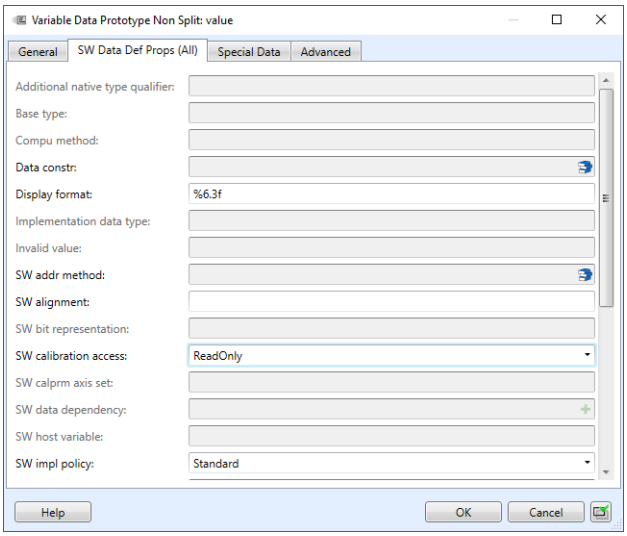
To allow measurement and calibration (MC) systems such as dSPACE ControlDesk to access data elements, you will now specify read-only measurements for selected data elements.

Method

To create measurements

1. In the Project Manager, double-click the value data element of the if_tss sender receiver interface in the SharedElements/PortInterfaces package.
   The Variable Data Prototype Non Split Properties dialog opens.
2. In the dialog, select the SW Data Def Props (All) page.
3. In the Display format edit field, enter `%6.3f`.

4 From the SW calibration access list, select `ReadOnly`.



5 Click OK to close the Variable Data Prototype Non Split Properties dialog.

6 Repeat steps 1 … 5 to create the following measurements:

Sender Receiver Interface	Data Element	Property	Value
if_wls	value	Display format	%1d
		SW calibration access	ReadOnly
if_bulb	signal	Display format	%1d
		SW calibration access	ReadOnly

Result

You have created measurements for selected data elements of sender receiver interfaces.

What's next

You will create calibration parameters in the next step (refer to Step 2: How to Create Calibration Parameters on page 143).

Step 2: How to Create Calibration Parameters

Objective

So that measurement and calibration (MC) systems such as dSPACE ControlDesk can modify atomic software components, you will now specify a shared parameter for an SWC internal behavior.

Specifying calibration parameters

To specify a calibration parameter, you have to perform the following steps:

1. Create a compu method for the linear conversion of a calibration parameter's physical values. Refer to Part 1 on page 144.
2. Constrain the physical values of a calibration parameter with a data constr element. Refer to Part 2 on page 145.
3. Create an application data type for a calibration parameter. Refer to Part 3 on page 146.
4. Specify the init value of a calibration parameter. Refer to Part 4 on page 147.
5. Create a calibration parameter. Refer to Part 5 on page 148.
6. Specify the access to a calibration parameter. Refer to Part 6 on page 149.

Part 1

To create a compu method for linear conversion of a calibration parameter's physical values

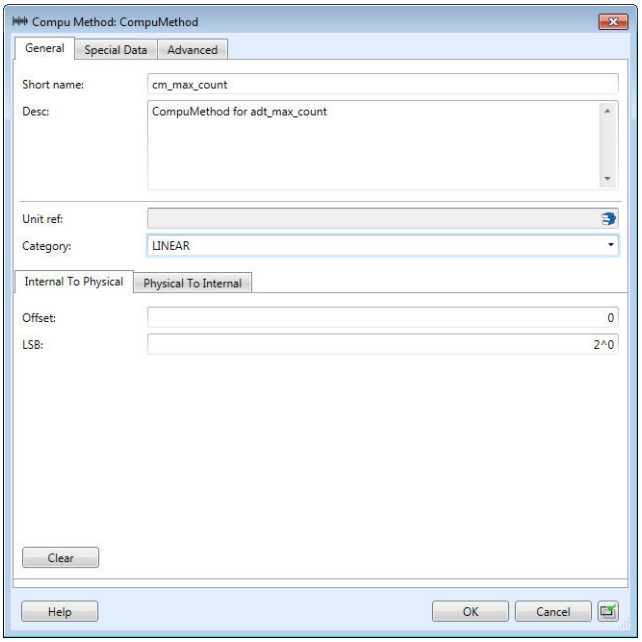
1 In the Project Manager, right-click the SwComponentTypes/IndicatorLogic/CompuMethods package.

2 From the context menu, select New – Compu Method.

SystemDesk adds a compu method to the project.

3 Open the element's Properties dialog and specify the following settings:

Property	Value
Short name	`cm_max_count`
Desc	`CompuMethod for adt_max_count`
Category	`LINEAR`
Internal To Physical	
Offset	0
LSB	2^0



4 Click OK to close the dialog.

Interim result

You have created a compu method for linear conversion of physical values.

Part 2

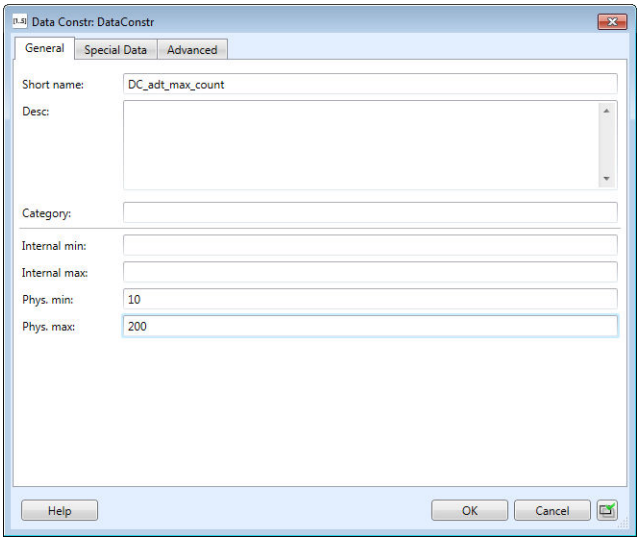
To constrain the physical values of a calibration parameter with a data constr element

1 In the Project Manager, right-click the SwComponentTypes/IndicatorLogic/ApplicationDataTypes package.

2 From the context menu, select New – New SWC-T – Data Constr.
SystemDesk adds a data constr element to the project.

3 Open the element's Properties dialog and specify the following settings:

Property	Value
Short name	DC_adt_max_count
Phys. min	10
Phys. max	200



4 Click OK to close the dialog.

Interim result

You have created a data constr element to constrain physical values.

Part 3

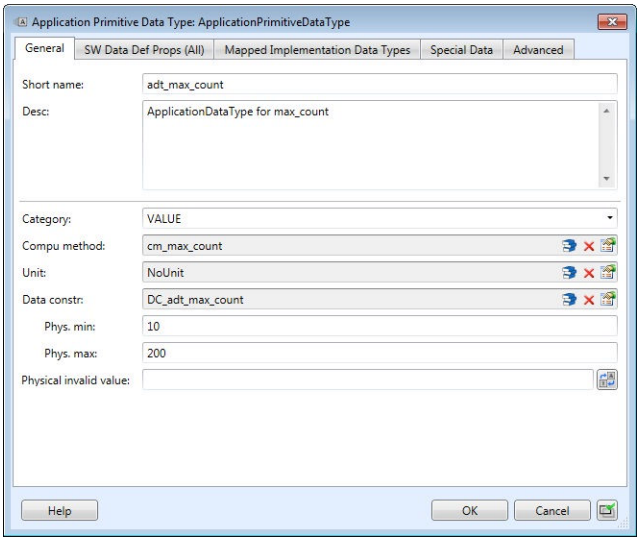
To create an application data type for the calibration parameter

1 In the Project Manager, right-click the SwComponentTypes/IndicatorLogic/ApplicationDataTypes package.

2 From the context menu, select New – Application Primitive Data Type.
SystemDesk adds a data type to the project.

3 Specify the following settings for the data type:

Property	Value
Short name	`adt_max_count`
Desc	`ApplicationDataType for max_count`
Category	`VALUE` (Default)
Compu method	`cm_max_count`
Unit	`NoUnit`
Data constr	`DC_adt_max_count`



4 Click OK to close the dialog.

Interim result

You have created an application data type that will be used for a calibration parameter.

Part 4

To specify the init value of a calibration parameter

1 In the Project Manager, right-click the SwComponentTypes/IndicatorLogic/ConstantSpecifications package.

2 From the context menu, select New – New SWC-T – Constant Specification.

SystemDesk adds a constant specification to the project.

3 Rename the constant specification to `CONST_max_count`.

4 Open the element's Properties dialog and type `Determines the duration of the on/off state of the indicator lights.` in the Desc edit field.

5 Type `VALUE` in the Category field.

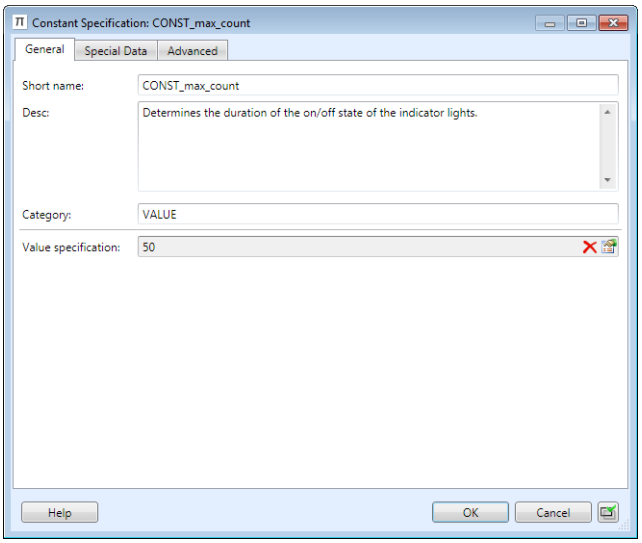
6 Click ✚ to add a Value specification.

SystemDesk opens the Select Type Ref dialog.

7 In the dialog select the adt_max_count application primitive data type and click OK.

SystemDesk shows the Application Value Specification dialog.

8 In the dialog, specify `50` in the Scalar edit field and click OK to return to the Constant Specification dialog.



9 Click OK to close the element's Properties dialog.

Interim result

You have specified an init value that will be used for a calibration parameter.

Part 5

To create a calibration parameter

1 In the Project Manager, double-click the IB_IndicatorLogic SWC internal behavior of the IndicatorLogic SWC in the SwComponentTypes/IndicatorLogic package.

The SWC Internal Behavior Properties dialog opens.

2 In the dialog, select the Parameters page.

3 In the dialog, select Shared and click New.

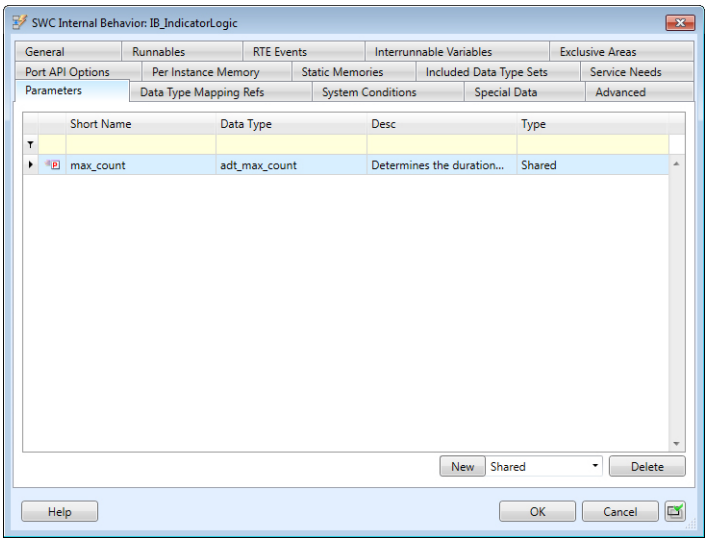
SystemDesk adds a shared calibration parameter to the list.

4 Double-click the parameter to specify the following settings in its Properties dialog:

Property	Value
General Page	
Short name	max_count
Desc	Determines the duration of the on/off state of the indicator lights.
Category	VALUE
Data type	adt_max_count
SW impl policy	Standard
Init value	CONST_max_count

Property	Value
SW Data Def Props (All) Page	
Display format	%3d
SW calibration access	ReadWrite

5 Click OK to close the dialog and return to the SWC Internal Behavior Properties dialog.



Interim result

You have created a calibration parameter that will be accessed by a runnable entity.

Part 6

To specify the access to a calibration parameter

1 In the Project Manager, double-click the SwComponentTypes/IndicatorLogic/IndicatorLogic/IB_IndicatorLogic/Toggle runnable entity.

The Runnable Properties dialog opens.

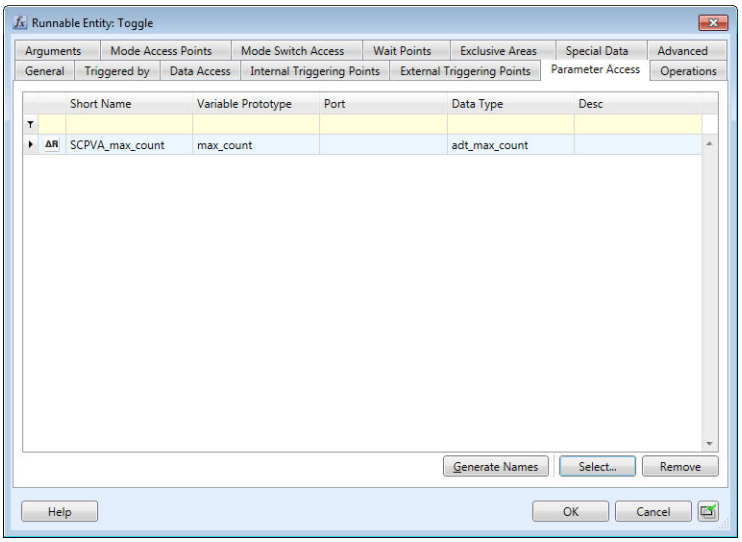
2 In the dialog, select the Parameter Access page.

3 Click Select to add a parameter access to the page.

SystemDesk opens the Select Element dialog.

4 In the dialog, select Shared/max_count from the tree and click OK.

SystemDesk adds a parameter access to the list.



5 Click **OK** to close the element's Properties dialog.

Result

You have specified a shared calibration parameter for the **Indicator Logic** SWC internal behavior. The calibration parameter can be accessed by the **Toggle** runnable entity.

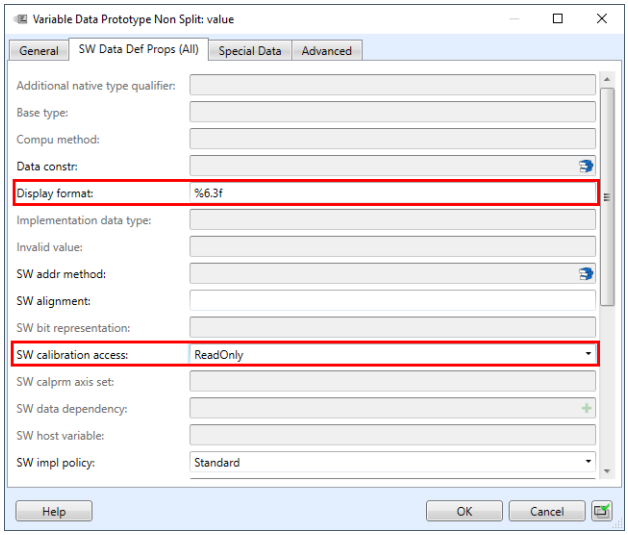
What's next

In this lesson, you created measurements and calibration parameters. Now check your work against the result you should have achieved. Refer to Result of Lesson 6 on page 151.

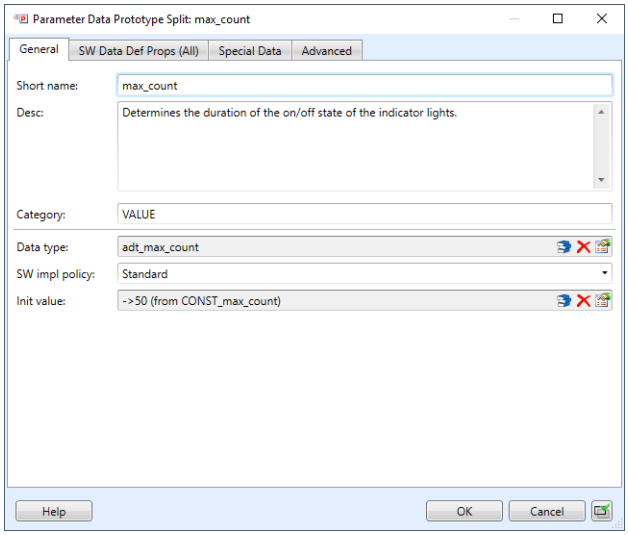
Result of Lesson 6

Summary

If you have done all the steps in this lesson, your project should contain measurements like this:



And calibration parameters like this:



In this lesson, you learned how to create measurements and calibration parameters.

Demo

You can reproduce the result of this lesson with the demo script `Lesson06_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

What's next

The next lesson shows you how to add data type mapping sets and constant specification mappings sets to the TutorialProject.

Lesson 7: Specifying Data Type Mapping Sets and Constant Specification Mapping Sets

Where to go from here

Information in this section

Overview of Lesson 7 153
In this lesson you will specify data type mapping sets and constant specification mappings sets.

Step 1: How to Map Application to Implementation Data Types 154
When you use application data types as shown in this tutorial, you have to map them to implementation data types.

Step 2: How to Map Constants 158
If you use constant specifications that have application data types, you can map them to constant specifications that have implementation data types.

Result of Lesson 7 164
In this lesson, you added data type mapping sets and constant specification mappings sets to the TutorialProject.

Overview of Lesson 7

Starting point

In the previous lesson you learned how to model measurement and calibration access.

What will you learn?

When you use application data types as explained in this tutorial, you have to map them to implementation data types for generating the RTE. However, this might not be sufficient in all cases depending on the tools which you use in later process steps. If you use constants that have application data types, you often

also have to specify a constant specification mapping, even if the application data type references a compu method. Some RTE generation tools require an explicit constant specification mapping.

In this lesson you will specify the following mappings:

- You will map application data types to implementation data types with a data type mapping set.
- You will map constant specifications that have application data types to constant specifications that have implementation types.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson06_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Map Application to Implementation Data Types on page 154
- Step 2: How to Map Constants on page 158

Summary

To check if the steps you executed are correct, refer to Result of Lesson 7 on page 164.

Step 1: How to Map Application to Implementation Data Types

Objective

When you use application data types as shown in this tutorial, you have to map them to implementation data types.

Method

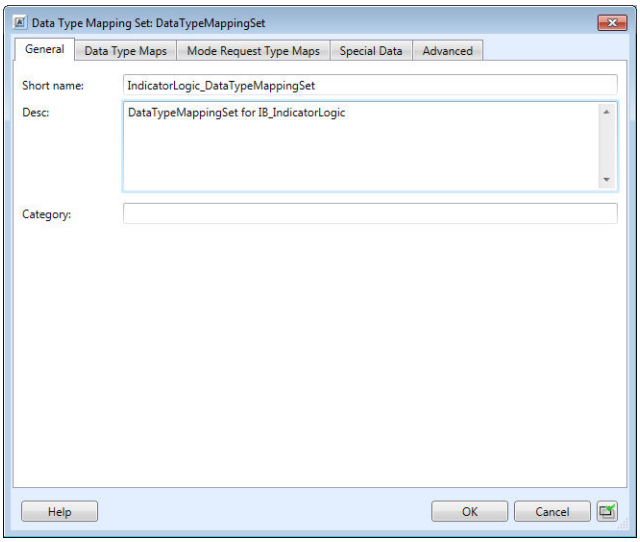
To map application to implementation data types

1. In the Project Manager, right-click the SwComponentTypes/IndicatorLogic package.
2. From the context menu, select New – Package and name the new package `DataTypeMappingSets`.
3. From the context menu of the DataTypeMappingSets package, select New – New SWC-T – Data Type Mapping Set.

   SystemDesk adds an element to the project.

4 Open the element's Properties dialog.

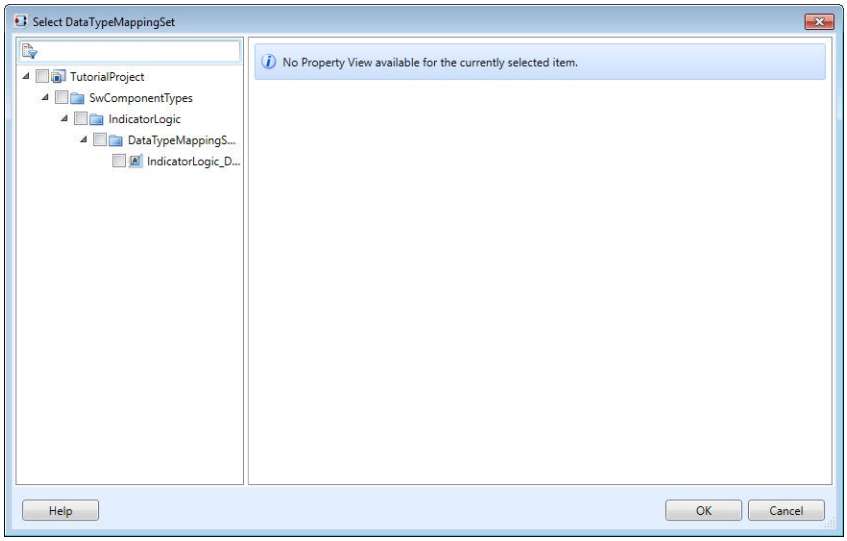
5 Rename the data type mapping set to `IndicatorLogic_DataTypeMappingSet` and type `DataTypeMappingSet for IB_IndicatorLogic` in the Desc edit field.



6 Click OK to close the Properties dialog of the data type mapping set.

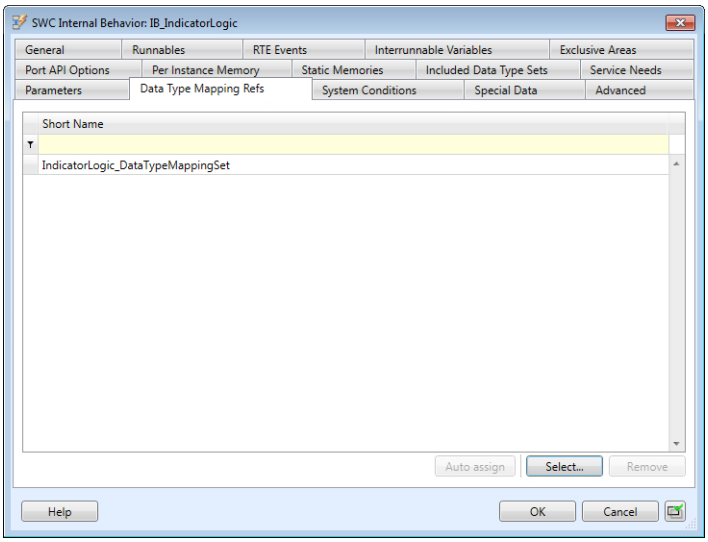
7 Open the Properties dialog of the IB_IndicatorLogic internal behavior.

8 Change to the Data Type Mappings Refs page and click Select.
SystemDesk opens the Select DataTypeMappingSet dialog.



9 In the dialog's tree view, select IndicatorLogic_DataTypeMappingSet and click OK to confirm your selection and close the dialog.

The IndicatorLogic_DataTypeMappingSet referenced is displayed on the Data Type Mappings Refs page.



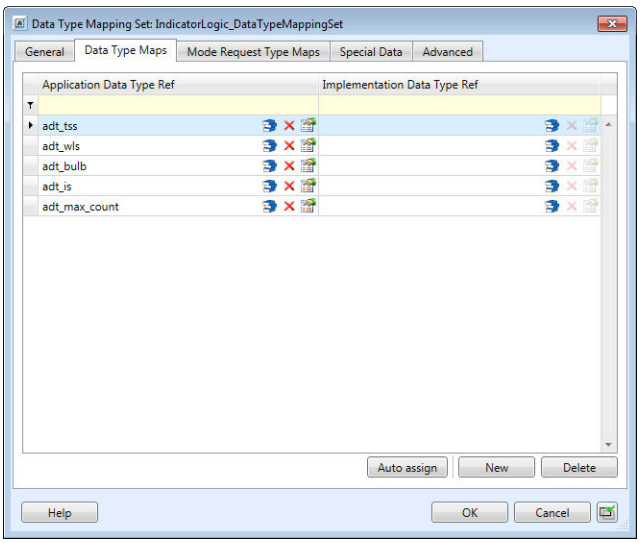
10 On the Data Type Mappings Refs page, click the IndicatorLogic_DataTypeMappingSet first and then Auto assign.

SystemDesk automatically adds to the selected data type mapping set all the application data types that are used in the IndicatorLogic SWC.

11 Click OK to close the Properties dialog of the IB_IndicatorLogic internal behavior.

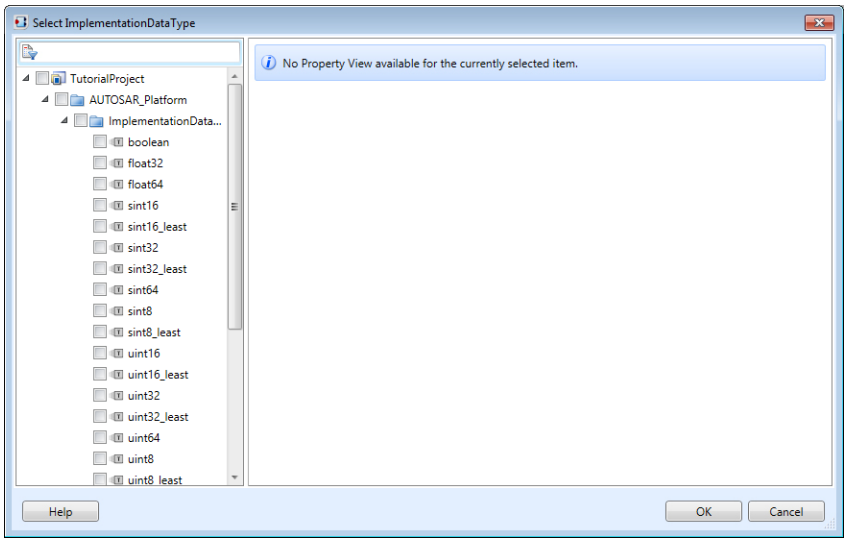
12 In the Project Manager, double-click the IndicatorLogic_DataTypeMappingSet to open its Properties dialog.

13 Change to the Data Type Maps page.



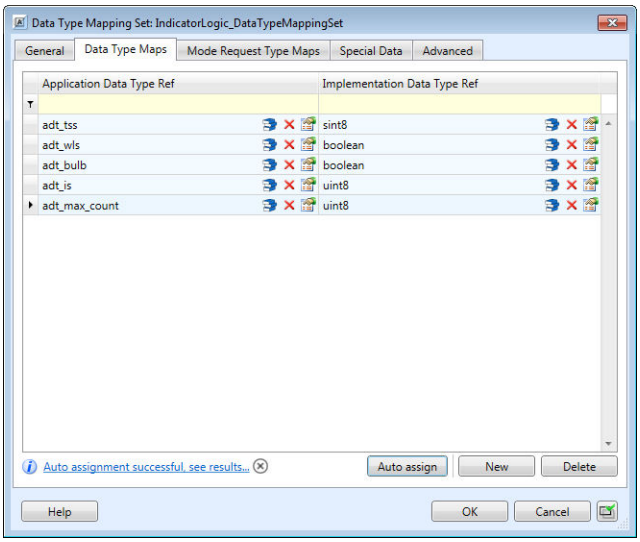
14 Multiselect all the data types and click **Auto assign**.

SystemDesk opens the **Select ImplementationDataType** dialog.



15 Select all the implementation data types. To avoid ambiguities during auto assignment, all selected implementation data types must have unique base types. Therefore, clear the **..._least** types checkboxes and click **OK**.

SystemDesk automatically maps the implementation data types to the related application data types.



16 Click **OK** to close the Properties dialog of the **IndicatorLogic_DataTypeMappingSet**.

Result

You have mapped application to implementation data types.

What's next

You will learn how to create and reference constant specification mapping sets in the next step (refer to Step 2: How to Map Constants on page 158).

Step 2: How to Map Constants

Objective

If you use constant specifications that have application data types, you can map them to constant specifications that have implementation data types. When you exchange data with basic software generation tools, the mapping is sometimes required for generating the RTE with the basic software generation tools.

Tip

For dSPACE TargetLink it is not necessary to create the constant specification mapping sets in SystemDesk. TargetLink is able to create constant specifications on implementation level automatically using the given constant specifications on application level and the implementation data types.

Mapping constants

To map constant specifications with application data types to constant specifications with implementation data types, you have to perform the following steps:

1. Create constant specifications that have implementation data types. Refer to Part 1 on page 158.
2. Specify constant specification mappings in a constant specification mapping set. Refer to Part 2 on page 161.
3. Reference constant specification mapping sets at an SWC internal behavior to use them when generating the RTE. Refer to Part 3 on page 163.

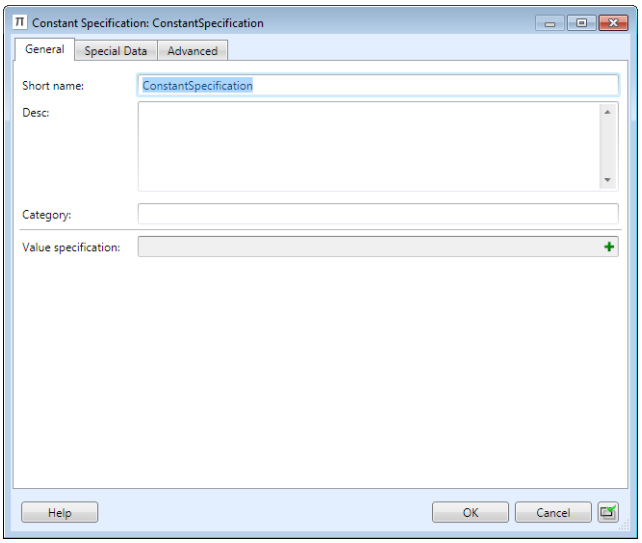
Part 1

To create constant specifications that have implementation data types

1. In the Project Manager, right-click the SwComponentTypes/IndicatorLogic/ConstantSpecifications package.
2. From the context menu, select New – New SWC-T – Constant Specification.

   SystemDesk adds an element to the project.

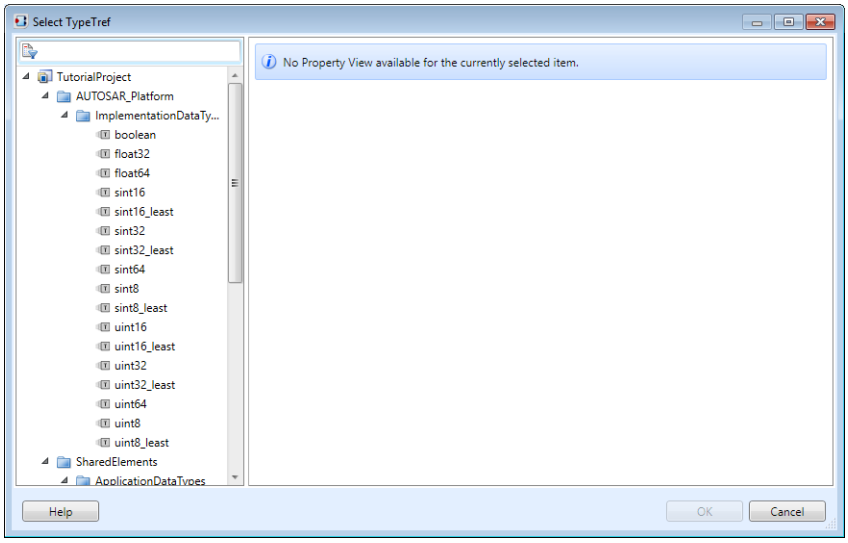
3 Double-click the new constant specification to open its Properties dialog.



4 Change the Short name to CONST_indicator_status_IDT and enter `Blink status` in the Desc field.

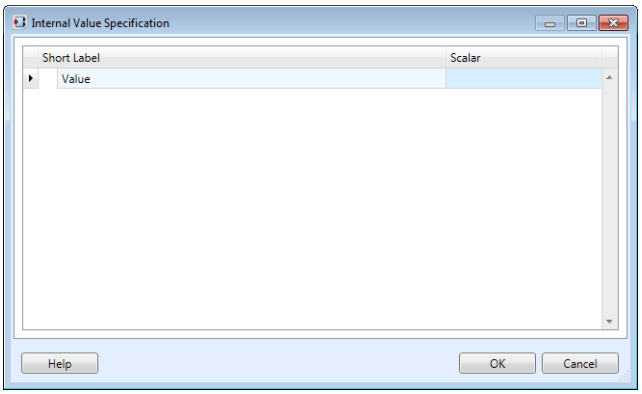
5 In the Category field, enter `VALUE` and click ✚ in the Value specification field.

The Select TypeTref dialog opens.



6 From `AUTOSAR_Platform/ImplementationDataTypes`, select sint8 and click OK to close the dialog.

SystemDesk displays the Internal Value Specification dialog.



7 In the Scalar field of the Internal Value Specification dialog, type `0` and click OK to close the dialog.

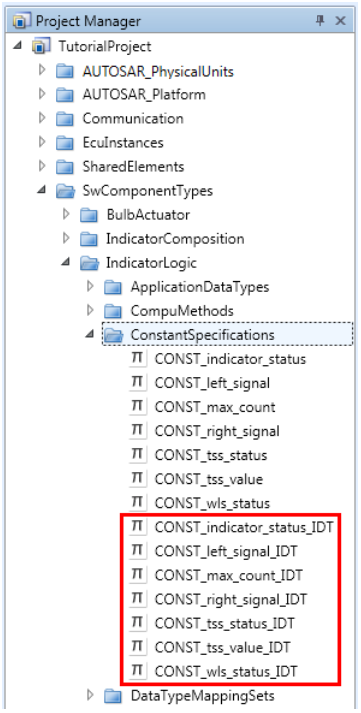
8 Click OK to close the Properties dialog of the constant specification.

9 Repeat steps 1 to 8 to specify the following constant specifications:

Short name	Desc	Category	Type Tref	Internal Value Specification
CONST_left_signal_IDT	`Initvalue constant for data element signal`	BOOLEAN	boolean	0
CONST_max_count_IDT	`Determines the duration of the on/off state of the indicator lights`	VALUE	sint8	50
CONST_right_signal_IDT	`Initvalue constant for data element signal`	BOOLEAN	boolean	0
CONST_tss_status_IDT	`Turn switch sensor status`	VALUE	sint8	0
CONST_tss_value_IDT	`Initvalue constant for data element value`	VALUE	uint8	0
CONST_wls_status_IDT	`Warn lights sensor status`	BOOLEAN	boolean	0

Interim result

You have created constant specifications that have implementation data types. Now you will specify a mapping between them and constant specifications on the application level.



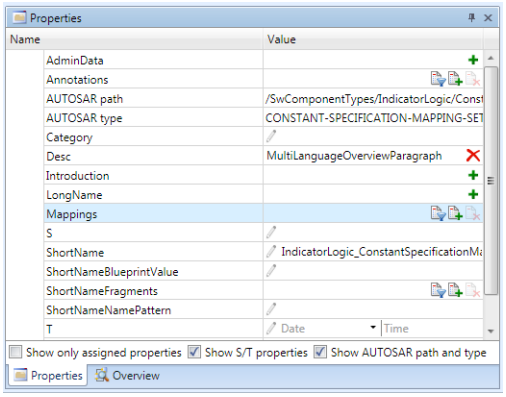
Part 2

To specify constant specification mappings

1. In the Project Manager, right-click the SwComponentTypes/IndicatorLogic package.
2. From the context menu, select New – Package and name the new package `ConstantSpecificationMappingSets`.
3. From the context menu of the ConstantSpecificationMappingSets package, select New – New SWC-T – Constant Specification Mapping Set.

   SystemDesk adds an element to the project.
4. Rename the element to `IndicatorLogic_ConstantSpecificationMappingSet`.
5. In the Properties controlbar, clear the Show only assigned properties checkbox.

6 In the Value field of Mappings, click to add a constant specification mapping.

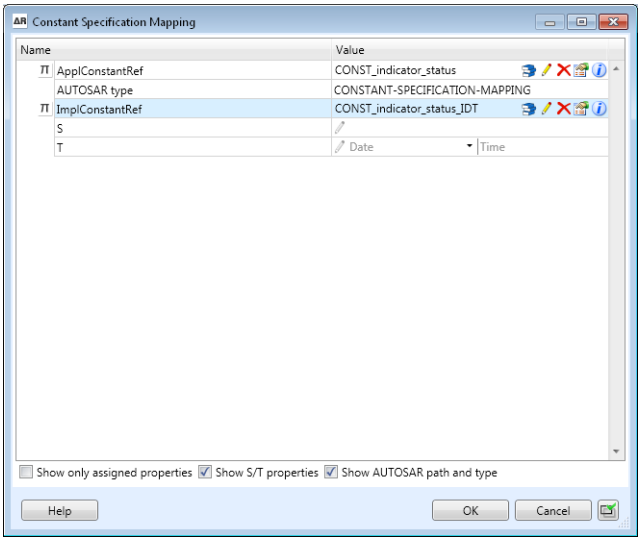


The Properties control bar displays a new constant specification mapping.

7 Double-click the new constant specification mapping to open the Constant Specification Mapping dialog.

8 In the Constant Specification Mapping dialog, click the buttons to specify the following mappings:

Name	Value
ApplConstantRef	CONST_indicator_status
ImplConstantRef	CONST_indicator_status_IDT



9 Click OK to close the Constant Specification Mapping dialog.

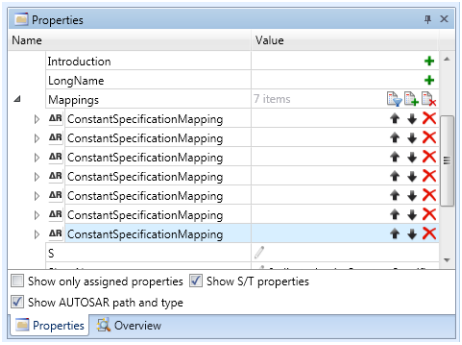
10 Repeat steps 5 … 9 to specify the following mappings:

ApplConstantRef	ImplConstantRef
CONST_left_signal	CONST_left_signal_IDT
CONST_max_count	CONST_max_count_IDT

ApplConstantRef	ImplConstantRef
CONST_right_signal	CONST_right_signal_IDT
CONST_tss_status	CONST_tss_status_IDT
CONST_tss_value	CONST_tss_value_IDT
CONST_wls_status	CONST_wls_status_IDT

Interim result

You have specified constant specification mappings in a constant specification mapping set that you will reference at an SWC internal behavior.

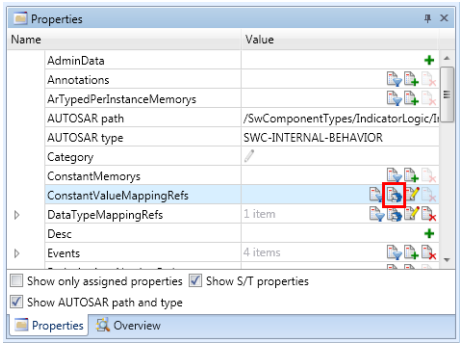


Part 3

To reference constant specification mapping sets

1 In the Project Manager, select the SwComponentTypes/IndicatorLogic/IndicatorLogic/IB_IndicatorLogic SWC internal behavior.

2 In the Properties control bar, click  in the Value field of ConstantValueMappingRefs to reference a constant specification mapping set.

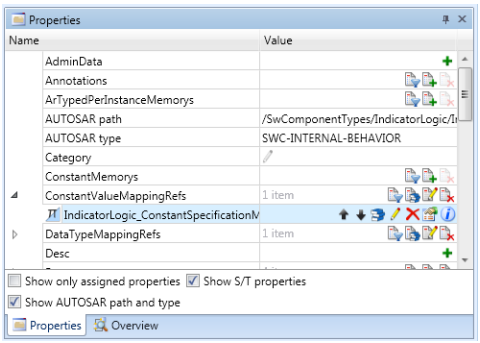


The Select ConstantSpecificationMappingSet dialog opens.

3 In the dialog, select IndicatorLogic_ConstantSpecificationMappingSet and click OK.

Result

You have mapped constant specifications that are typed by application data types to constant specifications that are typed by implementation types.



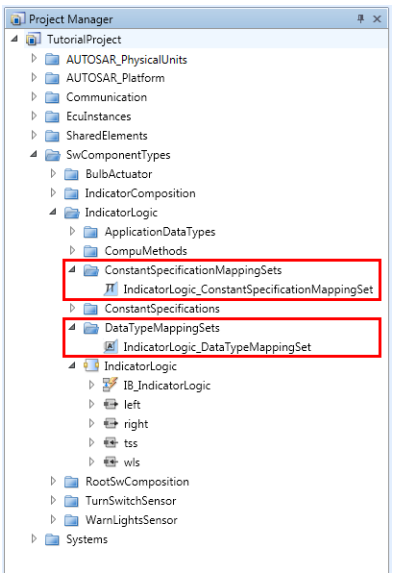
What's next

In this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 7.

Result of Lesson 7

Summary

If you have done all the steps in this lesson, your project should look like this:



In this lesson, you added data type mapping sets and constant specification mappings sets to the TutorialProject. You referenced the mapping sets at SWC internal behaviors to use the mapping sets when generating the RTE.

Demo

You can reproduce the result of this lesson with the demo script `Lesson07_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Note

The `Lesson07_complete.py` demo script also adds constant specifications and mapping sets to the other atomic software components.

Further information

For detailed information on modeling data types according to AUTOSAR, refer to Modeling Data Types (🕮 SystemDesk Manual).

What's next

The next lesson (refer to Lesson 8: Exchanging SWC Containers on page 167) shows how to map data types for generating the RTE.

Lesson 8: Exchanging SWC Containers

Where to go from here

Information in this section

Overview of Lesson 8 167
In this lesson you will learn how to exchange software components with TargetLink via SWC containers.

Step 1: How to Export SWC Containers 168
To exchange software components with TargetLink, you should use SWC containers.

Step 2: How to Integrate an SWC Implementation 171
In this step, you will learn how to specify an SWC implementation.

Result of Lesson 8 175
In this lesson, you exported SWC containers and added an SWC implementation to the TutorialProject.

Overview of Lesson 8

Starting point

In the previous lesson, you learned how to map data types for generating the RTE.

What will you learn?

In this lesson you will learn how to exchange software components with TargetLink via SWC containers.

An SWC container is a bundle of software component-related files such as AUTOSAR files (ARXML), code files (H/C), and variable description files (A2L). Using SWC containers makes exchanging data with TargetLink easier, and you can configure the data exchange. Typically, you export the software component design from SystemDesk to TargetLink. After implementing software components

and generating code with TargetLink, you import the SWC implementation back to SystemDesk and attach the generated code files to it.

In this lesson:

- You will assign AR elements to container files and export the resulting SWC containers.
- You will create an SWC implementation.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson07_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Export SWC Containers on page 168
- Step 2: How to Integrate an SWC Implementation on page 171

Summary

To check if the steps you executed are correct, refer to Result of Lesson 8 on page 175.

Step 1: How to Export SWC Containers

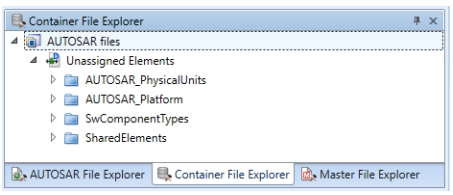
Objective

To exchange software components with TargetLink, you should use SWC containers.

Method

To export SWC containers

1 In the Container File Explorer, next to the Unassigned Elements node, click ▷ to expand it.



2 Right-click the AUTOSAR files node.

3 From the context menu, select **New – File Assignment**.

SystemDesk adds a file assignment to the tree.

4 Rename the file assignment to `AUTOSAR_Platform`.

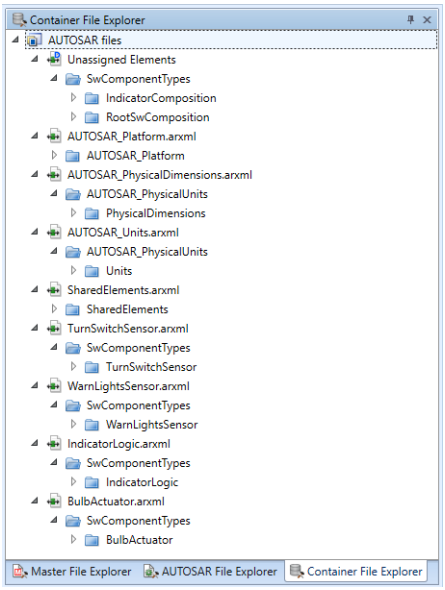
The file name extension is added automatically.

5 Drag the **AUTOSAR_Platform** package from the **Unassigned Elements** node to the **AUTOSAR_Platform** file assignment.

6 Assign the remaining packages as follows:

File Assignment	Assigned Packages
AUTOSAR_PhysicalDimensions	AUTOSAR_PhysicalUnits/PhysicalDimensions
AUTOSAR_Units	AUTOSAR_PhysicalUnits/Units
SharedElements	SharedElements
TurnSwitchSensor	SwComponentTypes/TurnSwitchSensor
WarnLightsSensor	SwComponentTypes/WarnLightsSensor
IndicatorLogic	SwComponentTypes/IndicatorLogic
BulbActuator	SwComponentTypes/BulbActuator

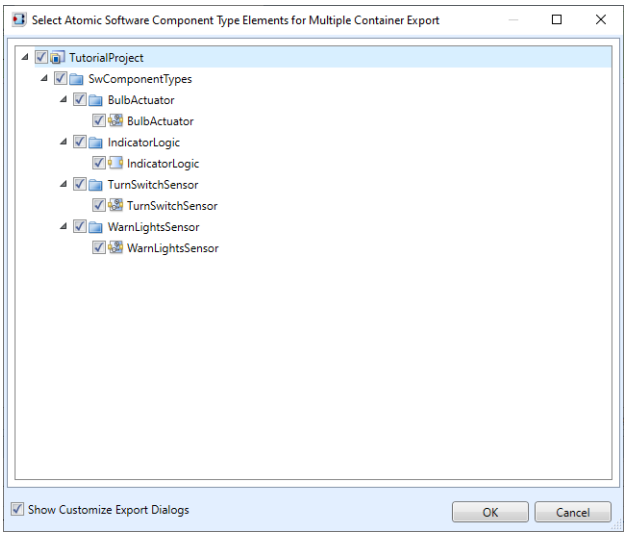
The composition packages remain unassigned, because they will not be exported.



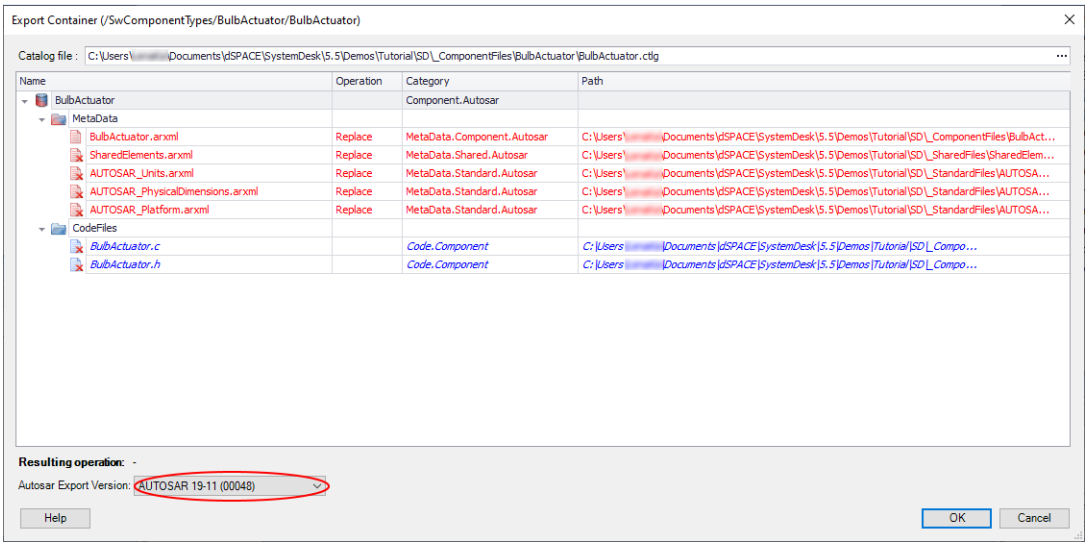
7 In the **Container Management** ribbon, click **Export – Multiple SWC Containers** to export SWC containers.

SystemDesk opens the **Select Atomic Software Component Type Elements for Multiple Container Export** dialog, which lets you select the atomic SWCs you want to export.

8 In the dialog, select all software components and click **OK**.



SystemDesk iterates over all the selected software components. It first validates a software component. Then it determines all AUTOSAR files containing references to the selected software component and opens the related **Export Container** dialog, as shown in the following illustration.



9 In the **Autosar Export Version** drop-down list, select the latest AUTOSAR release that is supported by the TargetLink version for which you export the SWC container. Click **OK** to close the **Export Container** dialog.

Repeat this action for each software component.

Result

You have exported SWC containers for the software component of the TutorialProject.

What's next

You can now integrate the SWC implementation in the next step (refer to Step 2: How to Integrate an SWC Implementation on page 171).

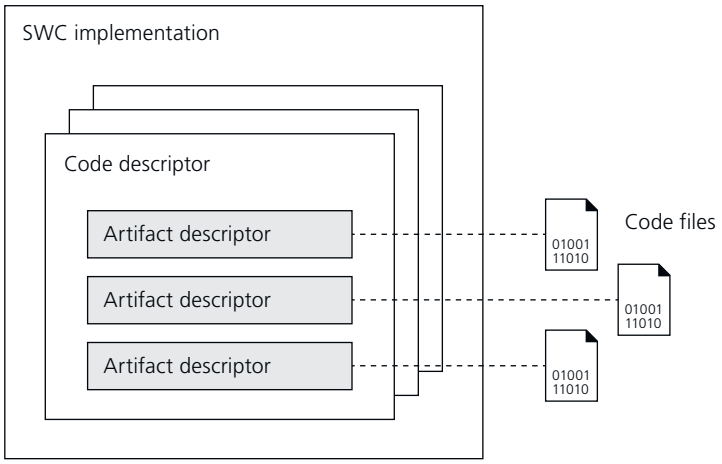
Step 2: How to Integrate an SWC Implementation

Objective

In the previous step, you exported SWC containers.

Typically, an SWC container will be imported by TargetLink and a behavior model will be designed in Simulink. TargetLink generates code for the behavior model and exports the SWC implementation to an SWC container. This container can then be imported back to SystemDesk. For the purpose of this tutorial, the SWC implementation is created manually to show the relevant elements.

In this step, you will learn how to specify an SWC implementation SWC Implementation. An SWC implementation describes how a specific SWC internal behavior is realized. An SWC implementation mainly consists of a list of source files and/or object files. It can also describe compiler attributes and build actions.



The illustration above shows the relations of an SWC implementation to the code files. An SWC implementation contains one or more code descriptors, which are used to refer to code files. Each code descriptor contains one artifact descriptor per code file with attributes such as category, domain, and short label. If you do not work with SWC containers, you also have to specify the file path.

You will add an SWC implementation to the **IndicatorLogic** SWC. You will then add one code descriptor with one artifact descriptor referencing the C code and another artifact descriptor referencing the header file. You will specify the category as *SWSRC* for the source code and *SWHDR* for the header.

Part 1

To add an implementation

1 In the Project Manager, right-click the IB_IndicatorLogic SWC internal behavior.

2 From the context menu, select New – SWC Implementation.

The Project Manager displays a new SWC implementation in the IndicatorLogic package.

3 Rename the SWC implementation to `IMPL_IndicatorLogic`.

Interim result

You added an empty SWC implementation. You can now reference the code files.

Part 2

To refer to code files

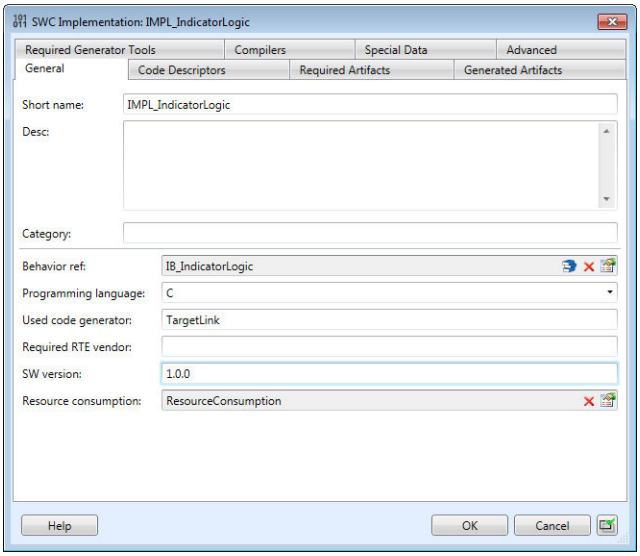
1 In the Project Manager, double-click the IMPL_IndicatorLogic implementation.

The SWC Implementation Properties dialog opens.

2 On the General page, select C from the Programming language list.

3 In the Used Code Generator field, enter `TargetLink`.

4 In the SW version field, enter `1.0.0`.



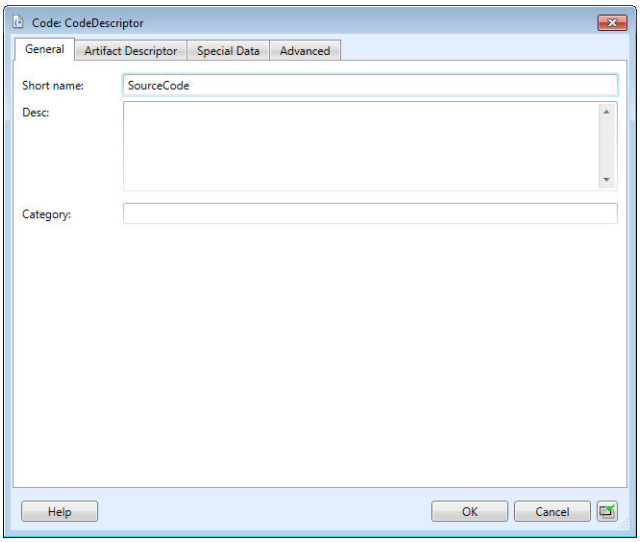
5 On the Code Descriptors page, click New.

SystemDesk creates a new code descriptor on the page.

6 Double-click the new code descriptor.

The Code Properties dialog opens.

7 On the General page, change the Short name to `SourceCode`.

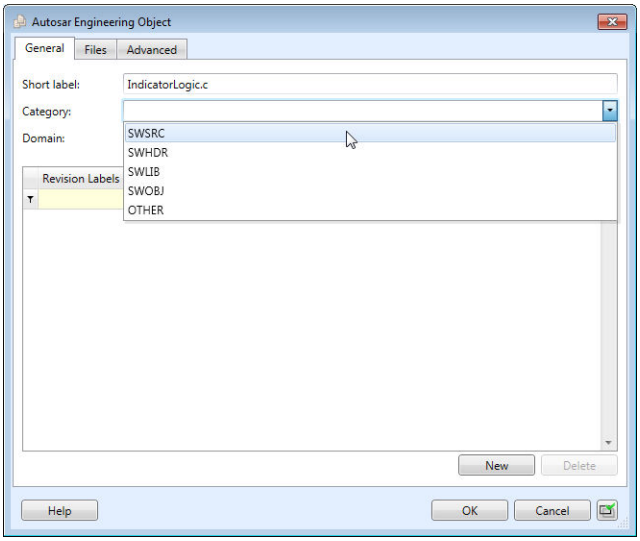


8 On the Artifact Descriptor page, click New.
SystemDesk creates a new AUTOSAR engineering object on the page.

9 Double-click the new AUTOSAR engineering object.
The AUTOSAR Engineering Object dialog opens.

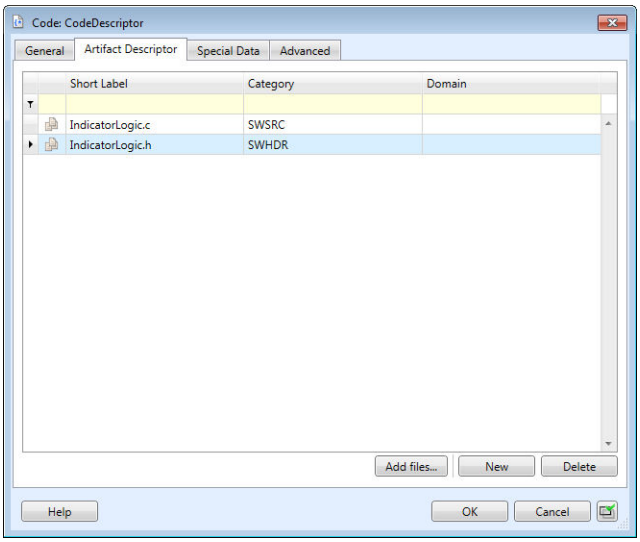
10 In the Short label edit field, type `IndicatorLogic.c`.

11 In the Category list, select `SWSRC`.



12 Click OK to close the AUTOSAR Engineering Object dialog and return to the Code Properties dialog.

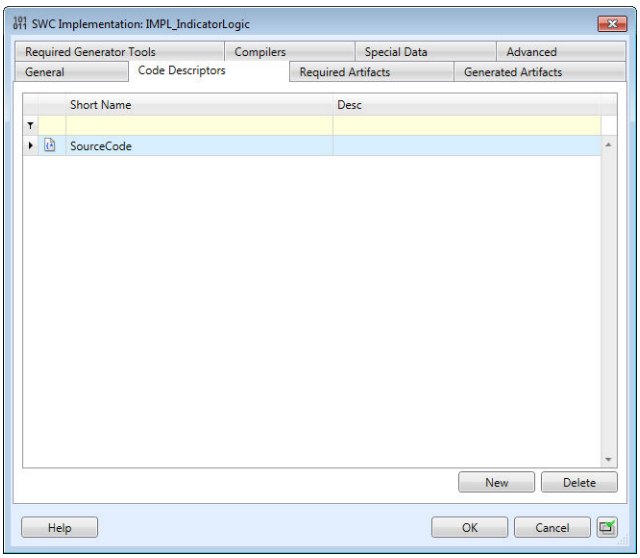
13 Repeat steps 8 … 12 to create another artifact descriptor with category `SWHDR` and short label `IndicatorLogic.h`.



Tip

The Add files button lets you add existing files directly from the file system.

14 Click OK to close the Code Properties dialog.



15 Click OK to close the SWC Implementation Properties dialog.

Result

You added an SWC implementation to the IndicatorLogic SWC.

What's next

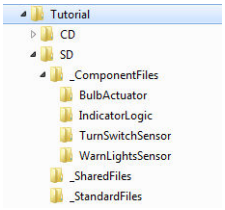
In this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 8 on page 175.

Result of Lesson 8

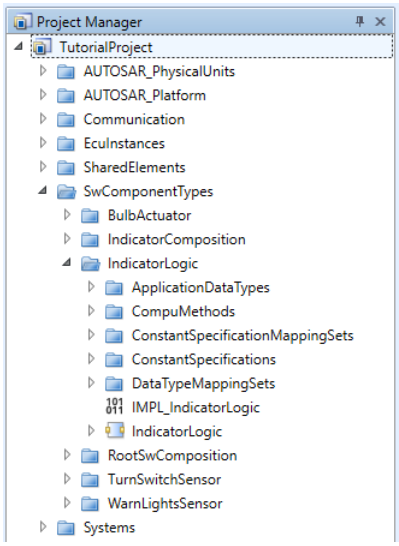
Summary

If you have done all the steps in this lesson:

- You will find the exported SWC containers in the `_ComponentFiles`, `_SharedFiles`, and `_StandardFiles` folders of your file system.



- Your SystemDesk project contains an SWC implementation for the IndicatorLogic SWC.



Demo

You can reproduce the result of this lesson with the demo script `Lesson08_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Note

The `Lesson08_complete.py` demo script also adds SWC implementations to the other atomic software components.

Further information

For detailed information on exchanging SWC containers, refer to Exchanging SWC Containers Between SystemDesk and TargetLink (📖 SystemDesk Manual).

What's next

The next lesson shows you how to work with ECU instances.

Lesson 9: Creating and Configuring an ECU Instance

Where to go from here

Information in this section

Overview of Lesson 9 177
In this lesson you will learn how to work with ECU instances.

Step 1: How to Create an ECU Instance 178
ECU instances are used to define the ECUs used in the network topology of a system.

Step 2: How to Configure ECU Instances 179
ECU instances have communication controllers and connectors that can access a physical channel of a communication cluster in the context of a system.

Result of Lesson 9 182
In this lesson, you learned how to add and configure an ECU instance.

Overview of Lesson 9

Starting point

In the previous lesson you learned how to exchange software components with TargetLink.

What will you learn?

In this lesson you will learn how to work with ECU instances.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson08_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create an ECU Instance on page 178
- Step 2: How to Configure ECU Instances on page 179

Summary

To check if the steps you executed are correct, refer to Result of Lesson 9 on page 182.

Step 1: How to Create an ECU Instance

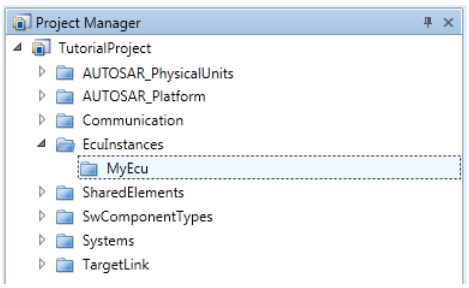
Objective

ECU instances are used to define the ECUs used in the network topology of a system.

Method

To create an ECU instance

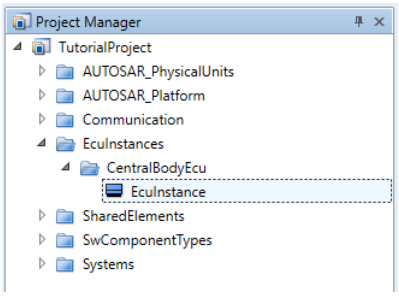
1 In the Project Manager, change to the MyEcu package.



2 Change the name of the MyEcu package to `CentralBodyEcu`.

3 From the context menu of the CentralBodyEcu package, select New – New System-T – ECU Instance.

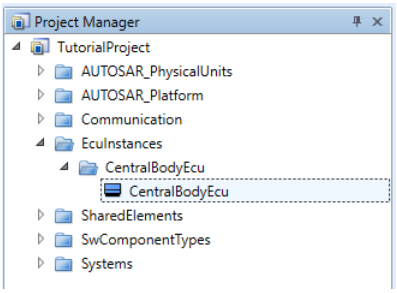
An ECU instance named EcuInstance is created in the CentralBodyEcu package.



4 Change the name of the EcuInstance ECU instance to `CentralBodyEcu`.

Result

You created the CentralBodyEcu ECU instance in the CentralBodyEcu package.



What's next

You will configure ECU instances in the next step (refer to Step 2: How to Configure ECU Instances on page 179).

Step 2: How to Configure ECU Instances

Objective

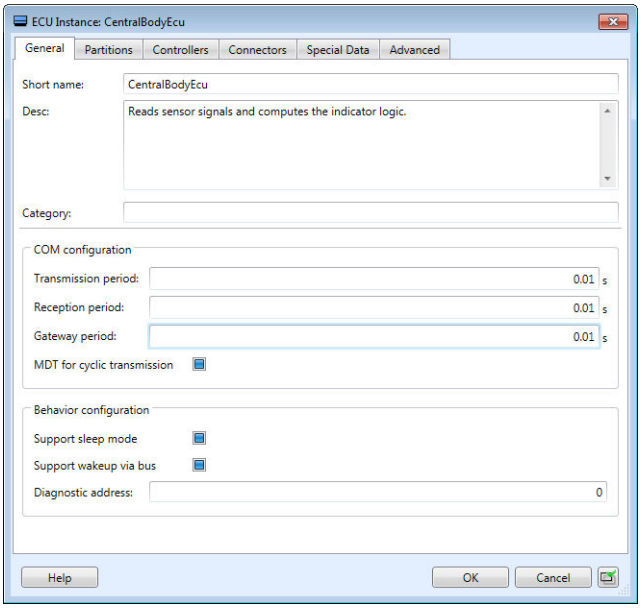
ECU instances have communication controllers and connectors that can access a physical channel of a communication cluster in the context of a system.

Method

To configure ECU instances

1 In the Project Manager, double-click the CentralBodyEcu ECU instance.
The ECU instance's Properties dialog opens.

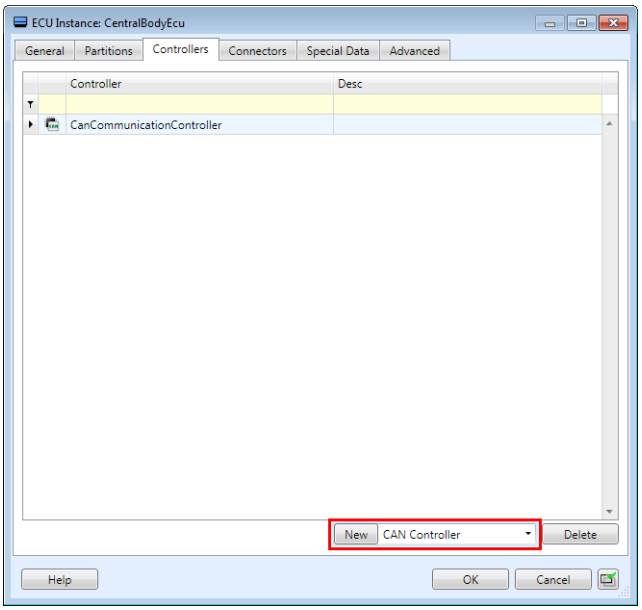
2 Type `Reads sensor signals and computes the indicator logic.` in the Desc edit field and specify the Transmission period, Reception period, and Gateway period as `0.01` seconds.



3 Change to the **Controllers** page of the dialog.

4 Select **CAN Controller** and click **New**.

A new CAN communication controller is displayed on the **Controllers** page.



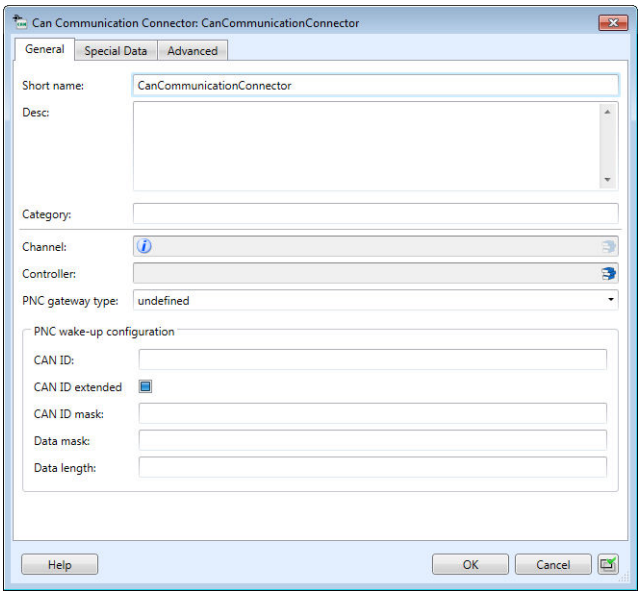
5 Change to the dialog's **Connectors** page.

6 Select **CAN Connector** and click **New**.

A new CAN communication connector is displayed on the **Connectors** page.

7 Double-click the new CAN communication connector.

The Properties dialog of the new CAN communication connector named CanCommunicationConnector opens.



8 Change the Short name to `CanBodyConnector`.

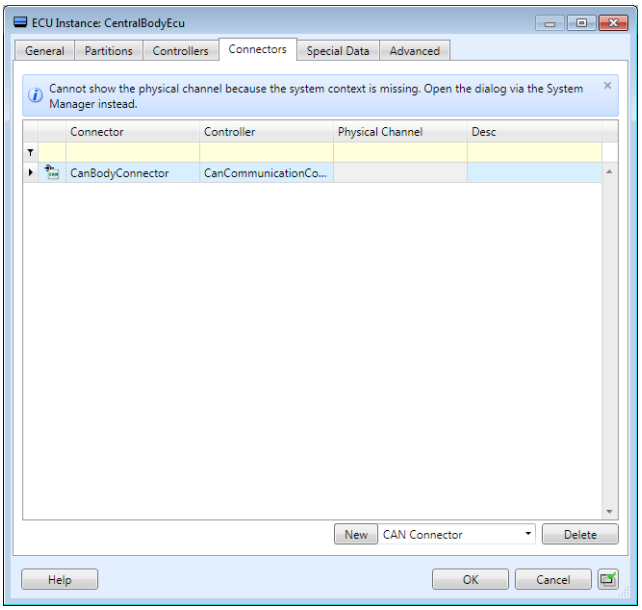
9 Click the icon in the Controller edit field to map the connector to a controller.

The Select Element dialog opens.

10 Select the Can CommunicationController you created before. Click OK to confirm your selection and return to the Properties dialog of the CAN communication connector.

11 Click OK to close the Properties dialog of the CAN communication connector and to return to the ECU instance's Properties dialog.

The new CAN communication connector and its mapping is displayed on the **Connectors** page (ECU instance).



12 Click **OK** to close the dialog.

Result

You have specified the topology of an ECU instance by adding controllers and connectors to it.

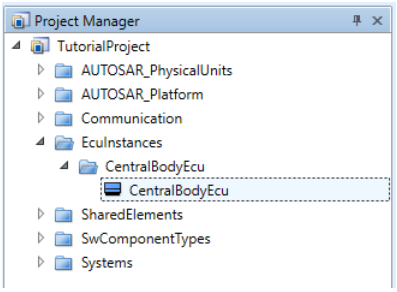
What's next

In this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 9 on page 182.

Result of Lesson 9

Summary

If you have done all the steps in this lesson, your project should look like this:



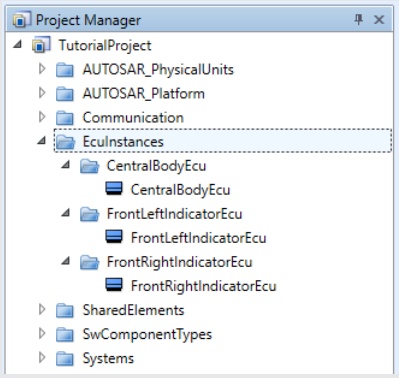
In this lesson, you learned how to add and configure an ECU instance.

Demo

You can reproduce the result of this lesson with the demo script `Lesson9_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Note

To facilitate your work, the demo script `Lesson9_complete.py` script will extend your project by two more packages with two ECU instances (FrontLeftIndicatorEcu and FrontRightIndicatorEcu) including their related CAN communication controllers and connectors.



Further information

For detailed information on modeling systems, refer to Modeling Systems (🕮 SystemDesk Manual).

What's next

The next lesson shows you how to import network communication elements with the AUTOSAR File Explorer control bar.

Lesson 10: Importing Network Communication Elements

Where to go from here

Information in this section

Overview of Lesson 10 185
In this lesson you will import network communication elements such as system signals, ISignal-IPdus, and frames.

Step 1: How to Import Network Communication Elements 186
You can use AUTOSAR files to exchange AUTOSAR elements such as network communication elements between project partners.

Result of Lesson 10 188
In this lesson, you learned how to perform data exchange with AUTOSAR files.

Overview of Lesson 10

Starting point

In the previous lesson you learned how to add and configure ECU instances.

What will you learn?

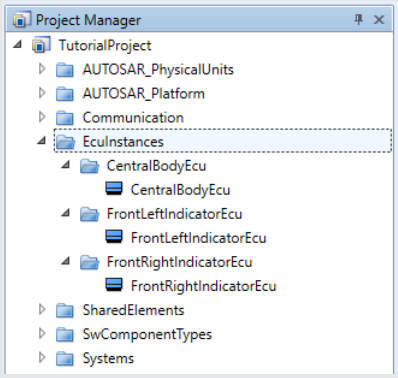
In this lesson you will import network communication elements such as system signals, ISignal-IPdus, and frames. Since network communication and ECU instances are often specified together, you will import both ECU instances and a CAN network cluster with its communication elements.

Before you begin

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons.

Note

The starting point of this lesson is the `Lesson09_complete.py` demo script. The script reproduces the results of the previous lessons and extends your project by adding and configuring an ECU instance not only for the CentralBodyEcu, but for all the ECUs of the tutorial project.



Run the `Lesson09_complete.py` demo script. You can find it in the `Tutorial\SD\Scripts` folder of your working directory.

Steps

This lesson contains the following steps:

- Step 1: How to Import Network Communication Elements on page 186

Summary

To check if the steps you executed are correct, refer to Result of Lesson 10 on page 188.

Step 1: How to Import Network Communication Elements

Objective

You can use *AUTOSAR files* to exchange AUTOSAR elements such as network communication elements between project partners.

The AUTOSAR File Explorer control bar allows you to add and import existing files to a project, or to create new AUTOSAR files, assign AR elements to them and export them. For each file, you can specify import/export options that are used when you import or export the file.

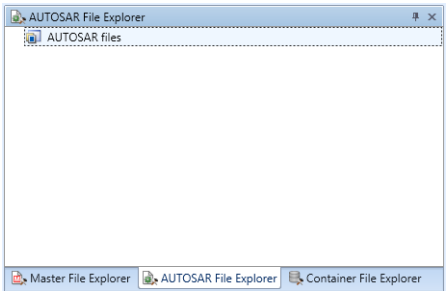
AR elements are those AUTOSAR elements that you can create directly in a package. AR elements are identified by the package they are contained in and

their short name. Examples for AR elements are ECU instances, communication clusters, or frames.

Method

To import network communication elements

1 Open the AUTOSAR File Explorer control bar.



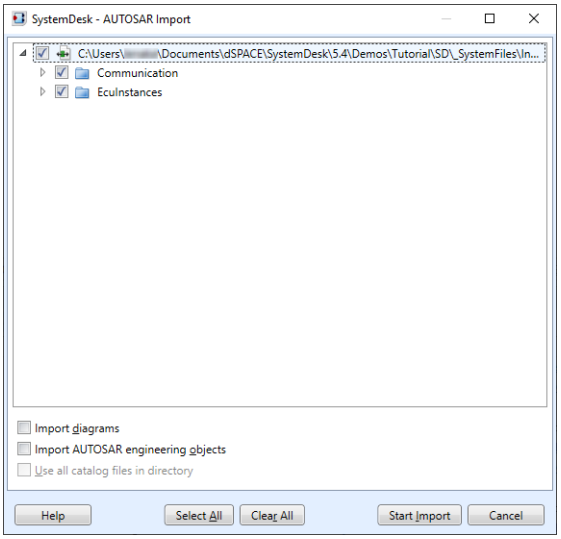
2 On the AUTOSAR File ribbon, click AUTOSAR File - Add Existing to add an existing AUTOSAR file to your project.

A standard file dialog opens.

3 Select the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\_SystemFiles\IndicatorSystem\CanBodyCluster.arxml` file. Click Open to confirm your selection.

The AUTOSAR Import dialog opens.

4 In the dialog, select the file contents.



5 Click Start Import.

Result

You have added an AUTOSAR file to the project and imported network communication elements from the file.

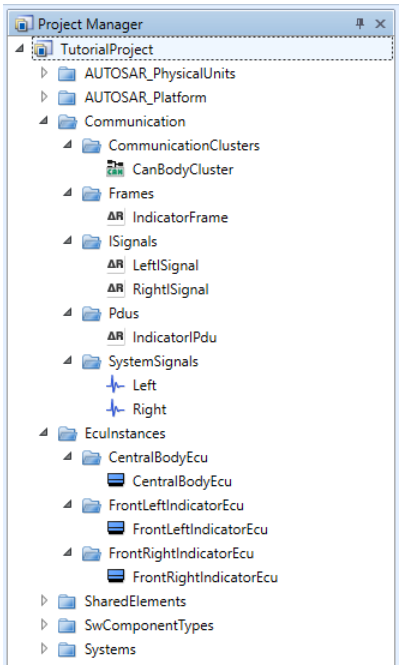
What's next

In this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 10 on page 188.

Result of Lesson 10

Summary

If you have done all the steps in this lesson, your project should look like this:



In this lesson, you learned how to perform a data exchange with the AUTOSAR File Explorer control bar by importing network communication elements.

Demo

You can reproduce the result of this lesson with the demo script `Lesson10_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information

For detailed information on modeling systems, refer to Modeling Systems (📖 SystemDesk Manual).

What's next

The next lesson shows you how to model the system.

Lesson 11: Modeling the System

Where to go from here

Information in this section

Overview of Lesson 11 190
In this lesson, you will model a system.

Step 1: How to Create a System 191
In this step you will learn how to create a system.

Step 2: How to Map Software Components to ECU Instances 195
In this step you will learn how to map software components to ECU instances.

Step 3: How to Select SWC Implementations 198
In this step you will learn how to select an SWC implementation for the atomic software components mapped to ECU instances of a system.

Step 4: How to Map SWC Communication to Network Communication 200
Communication between software components on different ECUs has to be mapped to system signals.

Result of Lesson 11 202
In this lesson you learned how to model a system.

Overview of Lesson 11

Starting point

In the previous lesson you imported network communication elements from an AUTOSAR file.

What will you learn?

In this lesson you will:

- Create a system and add a root software composition, a communication cluster, and ECU instances to it.
- Map software components to ECUs.
- Select SWC implementations for the software components.
- Map SWC communication to network communication.

Before you begin

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons.

Note

The starting point of this lesson is the demo script `Lesson10_complete.py`. The script reproduces the results of the previous lessons, in particular the addition of the internal behaviors to the SWCs, which you will need in the course of this lesson.

Run the demo script `Lesson10_complete.py`. You can find it in the `Tutorial\SD\Scripts` folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create a System on page 191
- Step 2: How to Map Software Components to ECU Instances on page 195
- Step 3: How to Select SWC Implementations on page 198
- Step 4: How to Map SWC Communication to Network Communication on page 200

Summary

To check if the steps you executed are correct, refer to Result of Lesson 11 on page 202.

Step 1: How to Create a System

Objective

In this step you will learn how to perform the following tasks:

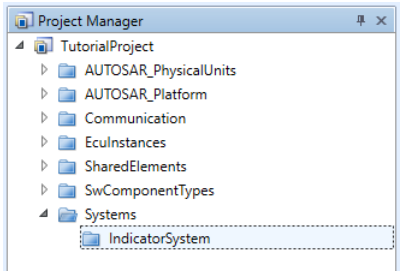
1. Create a new system in SystemDesk. Refer to Part 1 on page 191.
2. Select a composition SW component as the type for the system's root SW composition prototype. Refer to Part 2 on page 193.
3. Add a communication cluster to the system. Refer to Part 3 on page 193.

Part 1

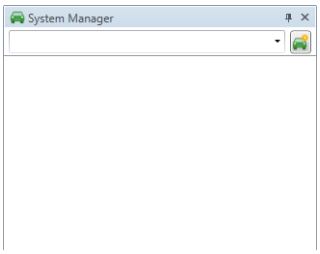
To create a system

1 In the Project Manager, change to the Systems package.

2 Change the name of the MySystem package to `IndicatorSystem`.



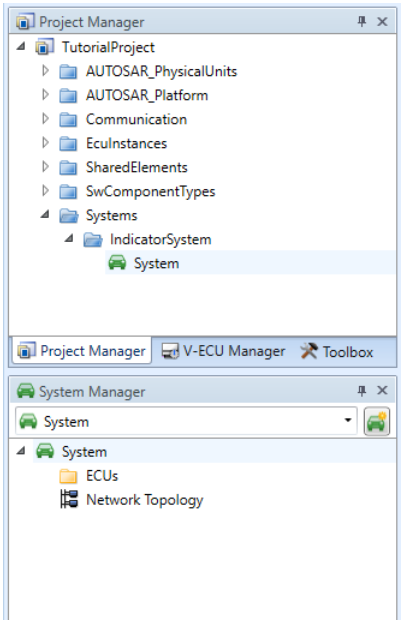
3 Select the System Manager.



4 In the System Manager, click the icon to create a new system.
The Select Package for new System dialog opens.

5 In the dialog, select the IndicatorSystem package and click OK.

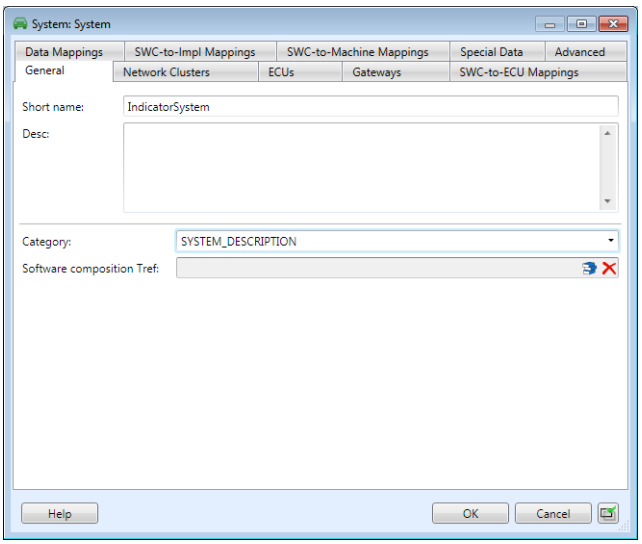
A new system is displayed in the System Manager and added to the IndicatorSystem package in the Project Manager.



6 In the System Manager, double-click the system.

The System Properties dialog opens.

7 On the dialog's General page, change the system's short name to `IndicatorSystem` and select the Category `SYSTEM_DESCRIPTION`.



Interim result

Via the System Manager, you created a new system in the IndicatorSystem package and named it IndicatorSystem.

You can now to configure the system.

Part 2

To select a composition SW component type for the system's root SW composition prototype

1 On the General page, click the icon in the Software composition Tref edit field to select a composition SW component type.

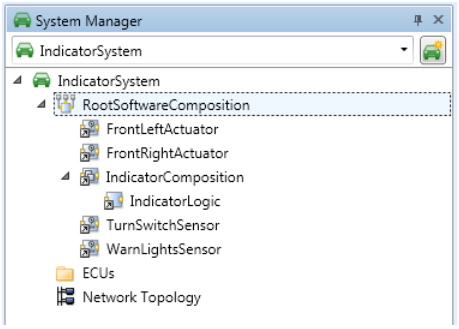
The Select CompositionSwComponentType dialog opens.

2 In the Select CompositionSwComponentType dialog, select RootSwComposition for the root SW composition prototype of the system. Click OK to confirm your selection and close the dialog.

3 Click OK to close the system's Properties dialog.

Interim result

The RootSoftwareComposition with its child elements is displayed in the System Manager.



You can now add a communication cluster to the system.

Part 3

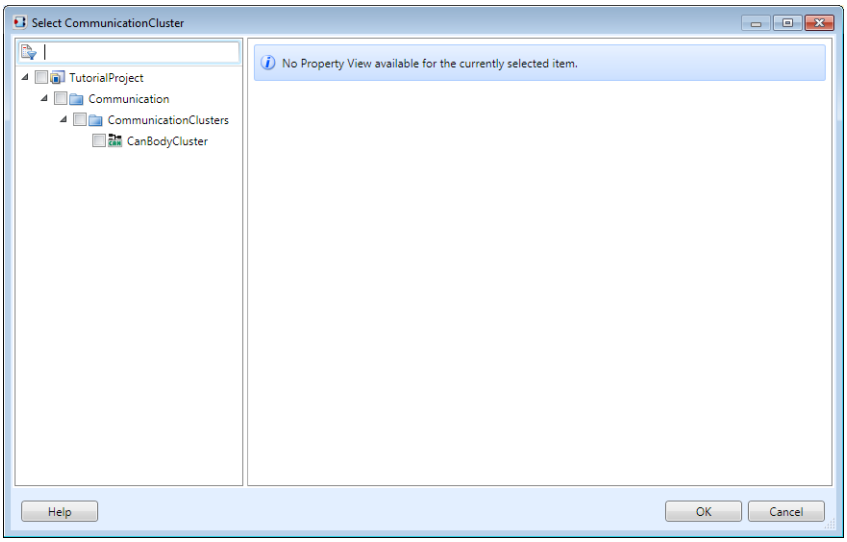
To add a communication cluster to the system

1 Reopen the IndicatorSystem Properties dialog.

2 Change to the Network Clusters page.

3 Click Select.

4 The Select CommunicationCluster dialog opens.



5 In the dialog's tree view, select the CanBodyCluster. Click OK to confirm your selection and close the dialog.

The CanBodyCluster is added to the system.

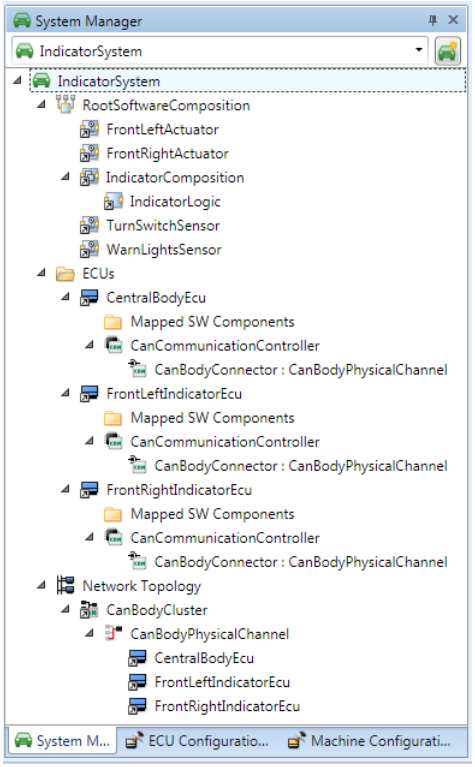
6 Click OK to close the system's Properties dialog.

Result

In this step you

- Created a new system in SystemDesk.
- Selected a composition SW component as the type for the system's root SW composition prototype.
- Added a communication cluster to the system.

The new system and its child elements are displayed in the System Manager.



What's next

You will learn how to map software components to ECU instances in the next step (refer to Step 2: How to Map Software Components to ECU Instances on page 195).

Step 2: How to Map Software Components to ECU Instances

Objective

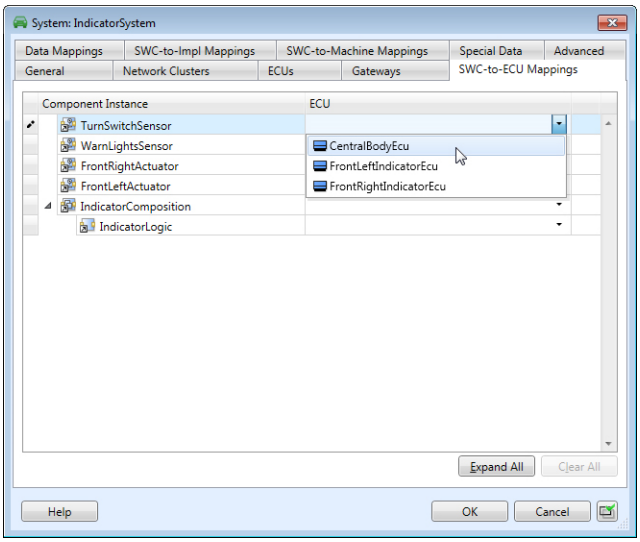
In this step you will learn how to map software components to ECU instances.

Method

To map software components to ECU Instances

1. From the context menu of the IndicatorSystem, select Properties.
   SystemDesk opens the General page of the Properties dialog.
2. Change to the SWC-to-ECU Mappings page.
   The page displays all the software components that are available in the system software architecture for you to map to the ECUs of the system.

3 In the ECU column, select the CentralBodyEcu for the TurnSwitchSensor component instance to map it to this ECU.



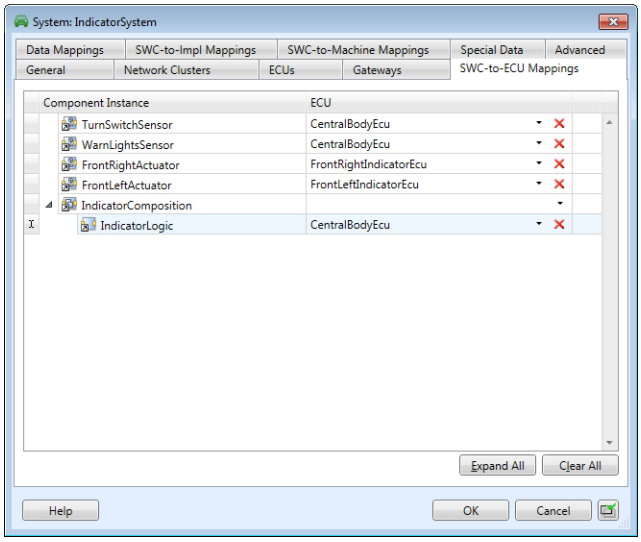
4 Repeat step 3 to map all the remaining software component instances as follows:

Component Instance	ECU Instance
WarnLightsSensor	CentralBodyEcu
FrontRightActuator	FrontRightIndicatorEcu
FrontLeftActuator	FrontLeftIndicatorEcu
IndicatorLogic	CentralBodyEcu

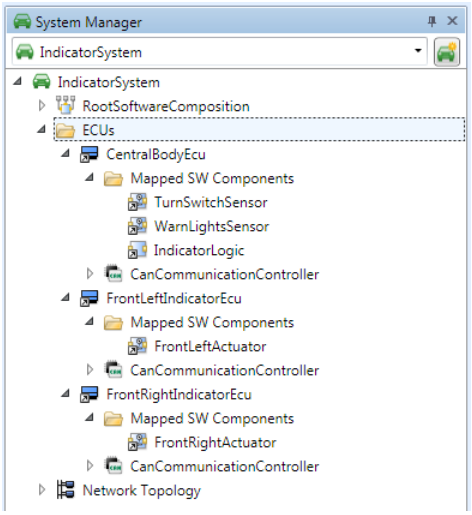
5 Click OK to close the dialog.

Result

You performed an SWC-to-ECU mapping by means of the **SWC-to-ECU Mappings** page. The mapping is displayed on the page as follows.

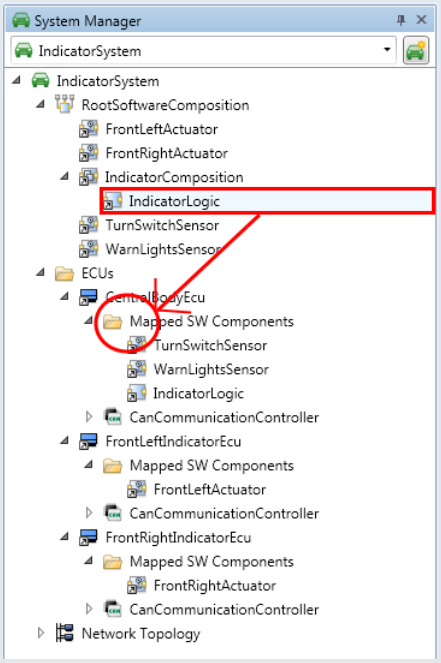


The SWC-to-ECU mapping is also displayed in the **System Manager**. Each ECU instance has its own **Mapped SwComponents** folder that contains the software components that are mapped to the corresponding ECU.



Tip

You can also perform an SWC-to-ECU mapping via drag & drop in the System Manager as follows.
Drag a software component to an ECU instance or its Mapped SwComponents folder to map it to the corresponding ECU.



What's next

You will learn how to select SWC implementations for atomic software components in the next step (refer to Step 3: How to Select SWC Implementations on page 198).

Step 3: How to Select SWC Implementations

Objective

In this step you will learn how to select an SWC implementation for each of the atomic software components mapped to the ECU instances of the IndicatorSystem.

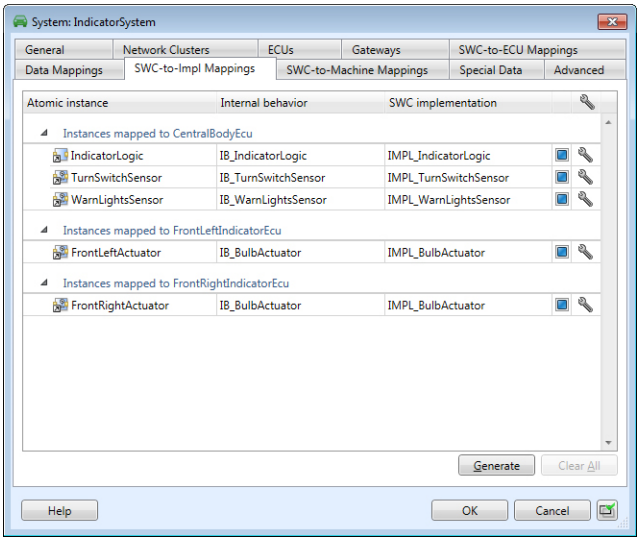
Method

To select SWC implementations

1 From the context menu of the IndicatorSystem, select Properties.
SystemDesk opens the General page.

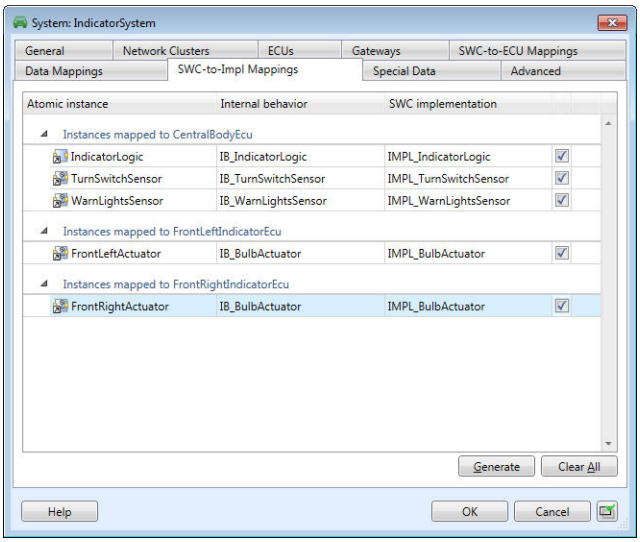
2 Change to the SWC-to-Impl Mappings page.

The page displays all the instances of software components that are available for the single ECU instances of the system and all the available SWC implementations for these software components.



The icons on the page indicate that you do not need to explicitly select an SWC implementation because only one is available. icons would indicate that several SWC implementation are available and you must select one of them.

3 Use the page's checkboxes to select the SWC implementations for the SWC internal behaviors of the software components as shown below.



4 Click OK to close the dialog.

Result

You selected one SWC implementation for each of the atomic software components mapped to the ECU instances of the system. The SWC implementations are used together with the generated RTE Code and the basic software to build the simulation application, which is a later development step.

Tip

To select SWC implementations, you can also use the **Generate** command on the **SWC-to-Impl Mappings** page. It performs an automatic mapping that selects the first available SWC implementation for each SWC internal behavior.

What's next

You will learn how to map SWC communication to network communication in the next step.

Step 4: How to Map SWC Communication to Network Communication

Objective

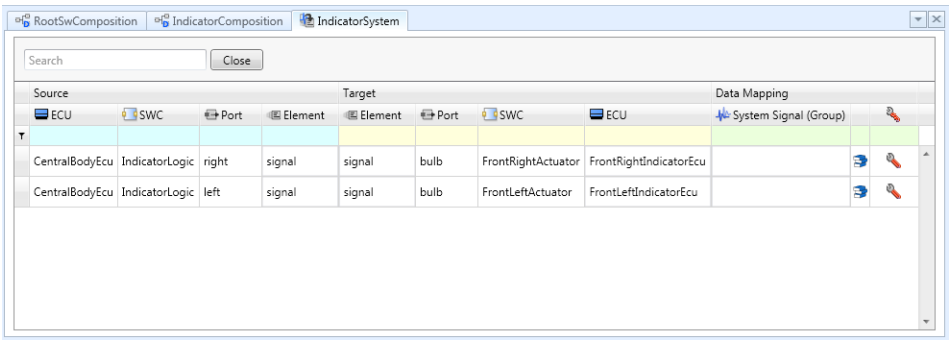
Communication between software components on different ECUs has to be mapped to system signals. This means mapping data elements to be transmitted between SWCs on different ECUs to system signals.

A system signal represents a data element in the communication domain and is required to define network communication.

Method

To map SWC communication to network communication

1 From the context menu of the **IndicatorSystem**, select **Edit Data Mapping**. SystemDesk opens the **Data Mapping Editor** in the working area.



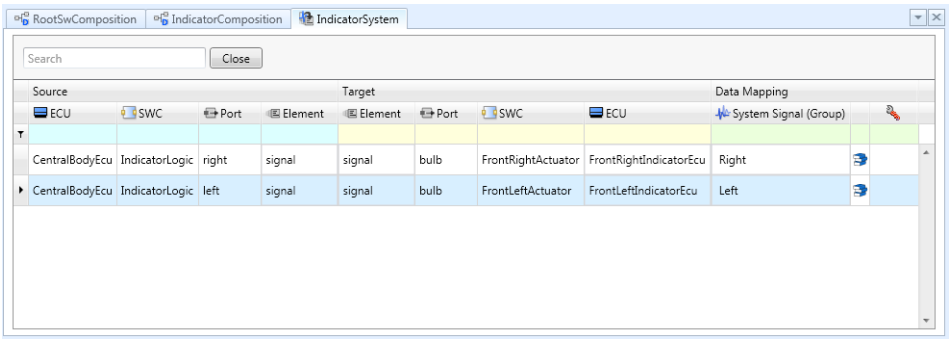
The **Data Mapping Editor** displays all the inter-ECU connections between atomic software components (SWCs). The editor also displays the data elements assigned to each of these software components and the data elements of the connected components. A icon in a row indicates that the related mapping is incomplete.

2 In the row of the **signal** data element of the **right** port, click the icon. The **Select SystemSignal** dialog opens.

3 In the dialog, select the **Right** signal.

4 Click **OK** to confirm your selection and close the **Select SystemSignal** dialog. The **signal** data element of the **right** port is now mapped to the **Right** signal. This mapping is displayed in the **Data Mapping Editor**.

5 Repeat steps 2 … 4 for the **signal** data element of the **left** port to map it to the **Left** signal.



Result

You have mapped SWC communication to network communication.

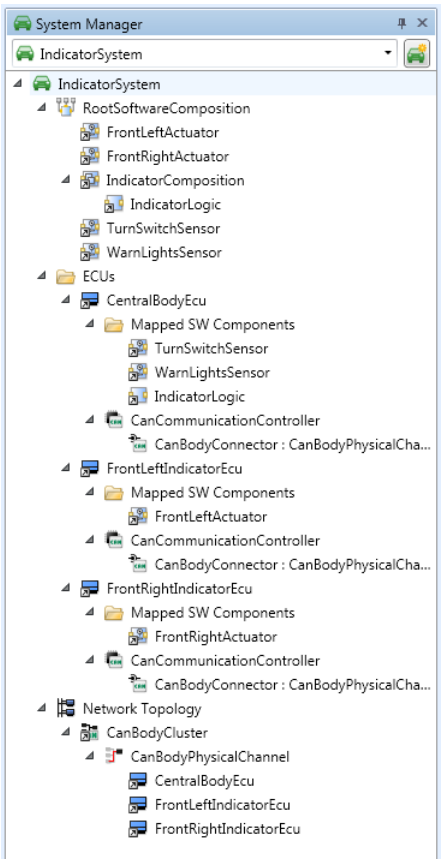
What's next

In this step, you completed the modeling and configuration of the system. Now check your work against the result you should have achieved. Refer to Result of Lesson 11 on page 202.

Result of Lesson 11

Summary

If you have done all the steps in this lesson, your system should look like this:



The **System Manager** also displays the elements and mappings for the network communication you imported in Lesson 10.

In this lesson you learned

- How to create and configure a system.
- How to map software component to ECUs.
- How to select SWC implementations for the mapped software components.
- How to map SWC communication to the network communication of the system.

Demo

You can reproduce the result of this lesson with the demo script `Lesson11_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information	For detailed information on modeling systems, refer to Modeling Systems (🕮 SystemDesk Manual).
What's next	The next lesson shows you how to exchange data.

Lesson 12: Importing and Exporting Data

Where to go from here

Information in this section

Overview of Lesson 12 205
In this lesson you will learn how to exchange data via AUTOSAR master files.

Step 1: How to Use AUTOSAR Master Files 206
In this step, you will create one AUTOSAR master file for each atomic software component and each composition SW component type, and an AUTOSAR master file containing the elements shared by the software components. You will also split a composition SW component type across two AUTOSAR master files to separate its internal and external parts.

Step 2: How to Export System Extracts 212
A system extract describes a subsystem on the basis of the system element.

Step 3: How to Import AUTOSAR Master Files 214
In this step you will import AUTOSAR master files to the project.

Result of Lesson 12 215
In this lesson you learned how to work with AUTOSAR master files and export a system extract file.

Overview of Lesson 12

Starting point

In the previous lesson, you learned how to model systems.

What will you learn?

In this lesson you will learn how to exchange data via AUTOSAR master files.

In this lesson:

- You will create AUTOSAR master files for storing AR elements and split some AR elements across AUTOSAR master files to separate the description of a composition SWC type into internal and external parts.
- You will export a system extract file, which is required to work with basic software tools such as EB tresos® Studio.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson11_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

It contains the following steps:

- Step 1: How to Use AUTOSAR Master Files on page 206
- Step 2: How to Export System Extracts on page 212
- Step 3: How to Import AUTOSAR Master Files on page 214 – addressed to users who have only SystemDesk's V-ECU Generation module.

Summary

To check if the steps you executed are correct, refer to Result of Lesson 12 on page 215.

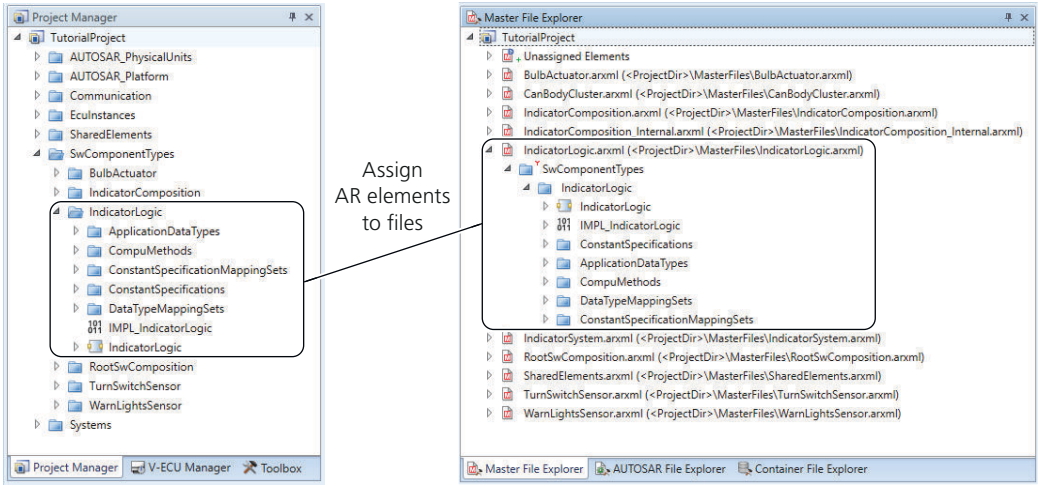
Step 1: How to Use AUTOSAR Master Files

Objective

AUTOSAR has defined the splittable mechanism for distributing elements across several AUTOSAR files. By supporting the splittable mechanism, SystemDesk lets you perform tasks such as separating the description of a composition SWC type into internal and external parts. This is useful for distributing the development of a software component between different parties involved in the development process, such as OEMs and suppliers.

You can read AUTOSAR master files and import the contained AR elements to your SystemDesk project. AR elements in the project that are assigned to an AUTOSAR master file but are not contained in the file are deleted from the project.

You can write AR elements from your SystemDesk project to AUTOSAR master files. The AUTOSAR master files then contain exactly the AR elements you assigned to them in your SystemDesk project.



In this step, you will first create one AUTOSAR master file for each atomic software component and each composition SW component type, and an AUTOSAR master file containing the elements shared by the software components. You will then assign the AR elements of the TutorialProject such as interfaces, data types and compu methods to the AUTOSAR master files.

To separate the external interface, i.e., the delegation ports, from the internal parts of the IndicatorComposition composition SWC type, you will split the IndicatorComposition composition SWC type across two AUTOSAR master files.

Finally, you will write the AUTOSAR master files to disk.

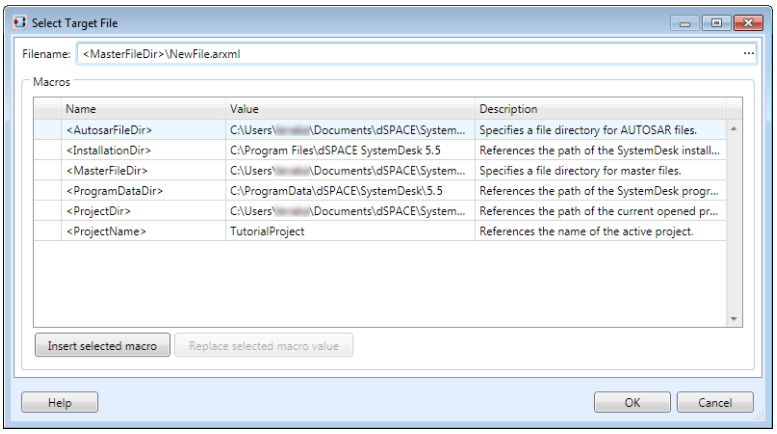
Part 1

To create AUTOSAR master files

1. In the Master File Explorer, right-click the TutorialProject node.
2. From the context menu, select New Master File.
   The Master File dialog opens.

3 Click the Browse button.

The Select Target File dialog opens.



4 In the Filename edit field, change to an appropriate folder in the file system, for example, the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\MasterFiles` folder and name the AUTOSAR file `SharedElements.arxml`.

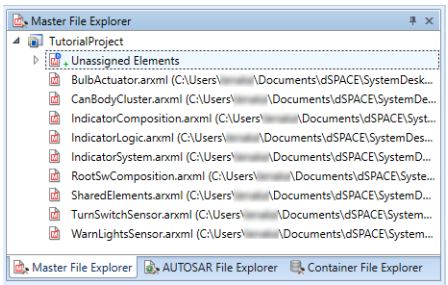
Click OK to confirm and close the dialog.

5 In the Schema version drop-down list of the Master File dialog, select the AUTOSAR release with which you are working.

6 Click OK to create the new AUTOSAR XML file.

7 Repeat steps 1 … 6 to create the following AUTOSAR XML files:

- `TurnSwitchSensor.arxml`
- `WarnLightsSensor.arxml`
- `BulbActuator.arxml`
- `IndicatorLogic.arxml`
- `RootSwComposition.arxml`
- `IndicatorComposition.arxml`
- `CanBodyCluster.arxml`
- `IndicatorSystem.arxml`



Interim result

You created empty AUTOSAR master files for your project. You can now assign the AR elements to the AUTOSAR master files.

Part 2

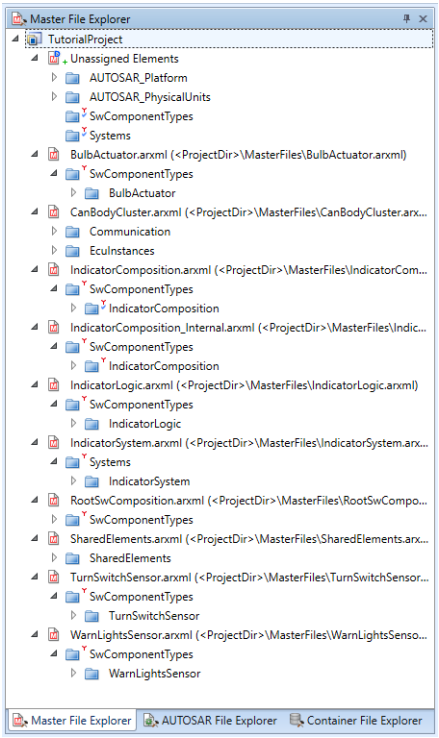
To assign AR elements to AUTOSAR master files

1 In the Master File Explorer, click ▷ next to the Unassigned Elements node to show its contents.

2 Below the Unassigned Elements node, select the SharedElements package and drag it to the SharedElements file.

3 Repeat step 2 to assign the remaining elements as follows:

Package	AUTOSAR Master File
SwComponentTypes/TurnSwitchSensor	TurnSwitchSensor
SwComponentTypes/WarnLightsSensor	WarnLightsSensor
SwComponentTypes/BulbActuator	BulbActuator
SwComponentTypes/IndicatorLogic	IndicatorLogic
SwComponentTypes/RootSwComposition	RootSwComposition
SwComponentTypes/IndicatorComposition	IndicatorComposition
Communication	CanBodyCluster
EcuInstances	CanBodyCluster
Systems/IndicatorSystem	IndicatorSystem



Interim result

You assigned AR elements to AUTOSAR master files. It is not necessary to assign the AUTOSAR templates (AUTOSAR_Platform, AUTOSAR_PhysicalDimensions, and AUTOSAR_PhysicalUnits) you imported in step 4 of lesson 1 to AUTOSAR master files.

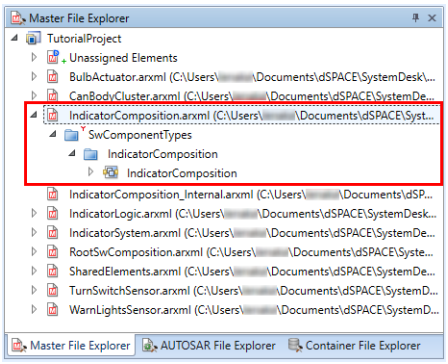
You can now split the IndicatorComposition composition SWC type between two AUTOSAR master files.

Part 3

To split AR elements between AUTOSAR master files

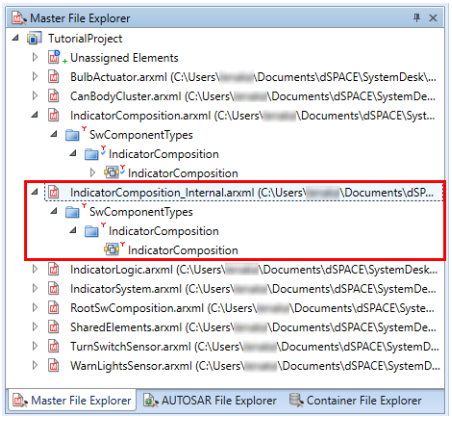
1 In the Master File Explorer, create a new AUTOSAR master file as described in part 1 and name it `IndicatorComposition_Internal.arxml`.

2 Expand the `IndicatorComposition.arxml` node and its assigned packages. Refer to the following illustration.



3 Drag the IndicatorComposition composition SWC type to the `IndicatorComposition_Internal.arxml` node and drop it while pressing the Alt key.

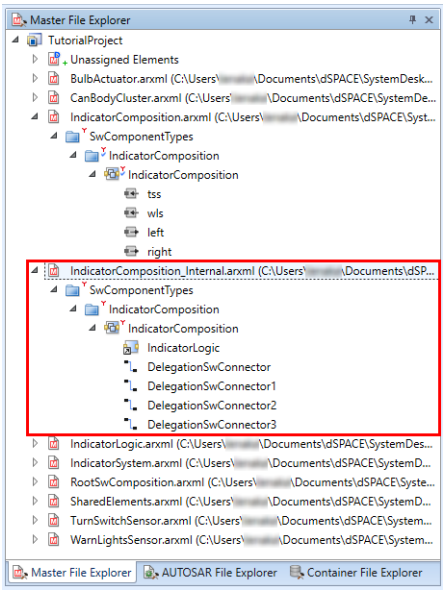
SystemDesk creates a split of the IndicatorComposition composition SWC type, which includes the overlying package structure, and marks the original SWC type and its containing package as the main split.



4 Expand the IndicatorComposition composition SWC type below the `IndicatorComposition.arxml` node and select the IndicatorLogic SWC

prototype and all the delegation SW connector types. On the context menu, select Move Split To – IndicatorComposition_Internal.arxml.

SystemDesk moves the IndicatorLogic SWC prototype and the associated delegation SW connector types to the IndicatorComposition split below the `IndicatorComposition_Internal.arxml` node.



5 On the context menu of the IndicatorComposition composition SWC type below the `IndicatorComposition_Internal.arxml` node, select Assign New Child Splittables. This ensures that splittable elements that are created below the IndicatorComposition composition are automatically assigned to the `IndicatorComposition_Internal.arxml` AUTOSAR master file.

Interim result

You created a split of the IndicatorComposition composition SWC type in the `IndicatorComposition_Internal.arxml` AUTOSAR master file that contains the internal parts of the composition.

You also ensured that new splittable elements of the IndicatorComposition composition are automatically assigned to the `IndicatorComposition_Internal.arxml` AUTOSAR master file.

You can now write the AR elements to the AUTOSAR master files.

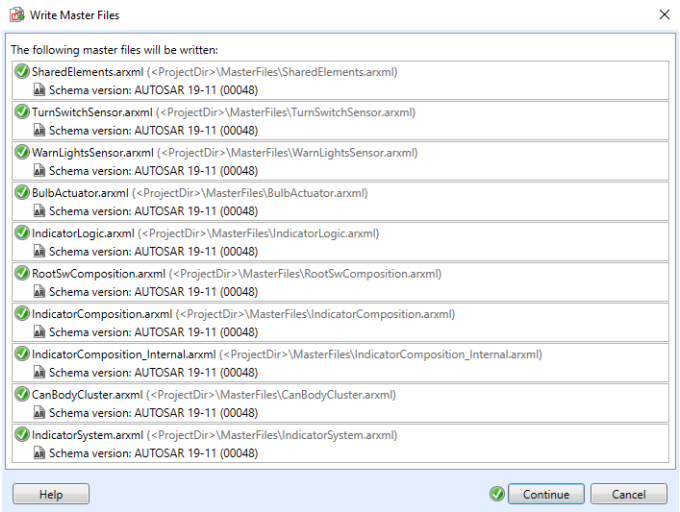
Part 4

To read/write AUTOSAR master files

1 In the Master File Explorer, right-click the TurorialProject node.

2 From the context menu, select Write All Master Files.

3 In the Write Master Files dialog, click Continue.



AR elements are written to the files to which they are assigned. The files can now be added and synchronized in a version control system, for example.

Result

You created AUTOSAR master files and assigned and wrote your AR elements to them.

You also created a split of the **IndicatorComposition** composition SWC type that contains only the internal parts of the composition and configured the automatic assignment of new child splittables to the AUTOSAR master file of the internal composition parts.

You can look at the AUTOSAR master files in the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\Master Files` folder.

What's next

You will export a system extract file in the next step.

Step 2: How to Export System Extracts

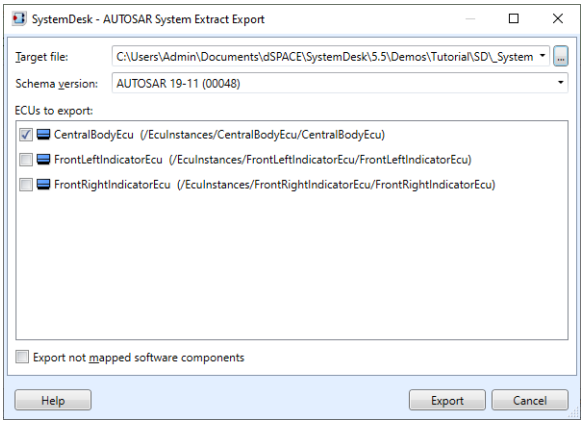
Objective

A *system extract* describes a subsystem on the basis of the system element. You can select one or more ECU elements of a system to define a system extract for SystemDesk to export.

Method

To export system extracts

1. In the Project Manager, right-click the IndicatorSystem system.
2. From the context menu, select Export – AUTOSAR System Extract.
   The SystemDesk – AUTOSAR System Extract Export dialog opens.
3. In the dialog, specify the `<My Documents>\dSPACE\SystemDesk\5.4\Demos\Tutorial\SD\_SystemFiles\SystemExtracts\CentralBodyEcuSystemExtract.arxml` file.
4. Select the AUTOSAR release with which you are working from the Schema version drop-down list.
5. Select the CentralBodyEcu.



6. Click Export to write a system extract file to the specified location.

Result

You have exported a system extract.

What's next

Note

Users with the Modeling module who worked through this tutorial completed the lesson with this step and can skip the next step.

Now check your work against the result you should have achieved. Refer to Result of Lesson 12 on page 215.

Step 3: How to Import AUTOSAR Master Files

Objective

In this step you will import AUTOSAR master files to the project.

Note

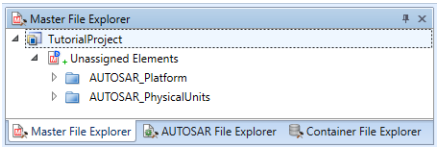
You have to proceed with this step only if you have only the V-ECU Generation module.

Available Modules	Action
Modeling and V-ECU Generation	Skip this step.
Modeling	None - You have finished the tutorial with step 2.
V-ECU Generation	Proceed with this step.

Method

To import AUTOSAR master files

1 In the Master File Explorer, right-click the TutorialProject node.



2 From the context menu, select Add Existing File to Project.

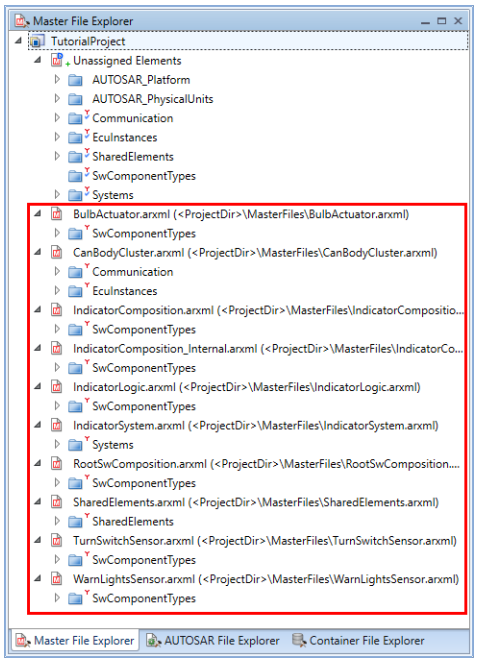
SystemDesk opens a standard Open dialog for you to select the AUTOSAR files to be added to the project as AUTOSAR master files.

3 Change to the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\MasterFiles` folder and open all the ARXML files contained in the folder.

All the AR elements that are contained in the files are added to the project.

Result

You have imported AUTOSAR master files to the project.



What's next

In this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 12 on page 215.

Result of Lesson 12

Summary

In this lesson you learned how to work with AUTOSAR master files and export a system extract file.

Note

If you have only SystemDesk's modeling module, you have now successfully completed this tutorial.

Demo

You can reproduce the result of this lesson with the demo script `Lesson12_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Further information

For more information on AUTOSAR master files and the splittable mechanism, refer to:

- Basics on AUTOSAR Master Files (🕮 SystemDesk Manual)
- Basics on Assigning Elements (🕮 SystemDesk Manual)

What's next

The next lesson shows you how to configure an ECU and model basic software components.

Lesson 13: Modeling ECU Configurations and Basic Software Components

Where to go from here

Information in this section

Overview of Lesson 13 218
In this lesson you will learn how to configure ECUs.

Step 1: How to Create an ECU Configuration 219
You will now add an ECU configuration to an ECU.

Step 2: How to Create the I/O Hardware Abstraction 223
In this step, you will create an I/O hardware abstraction component.

Step 3: How to Configure the ECU State Manager and the COM Stack 225
You will now configure the ECU state manager (EcuM) and the COM stack (COM).

Step 4: How to Map Runnables to OS Tasks 228
You will now create OS tasks and map the runnables of the software components via RTE events.

Step 5: How to Complete the OS Configuration 231
Completing the OS configuration mainly means working with OS tasks, OS events, OS alarms, OS application modes, and OS counters.

Step 6: How to Generate the RTE 234
In this step, you will now learn how to configure RTE code generation and generate C code for the RTE and all the other BSW modules of an ECU configuration.

Result of Lesson 13 237
In this lesson, you learned how to configure an ECU.

Overview of Lesson 13

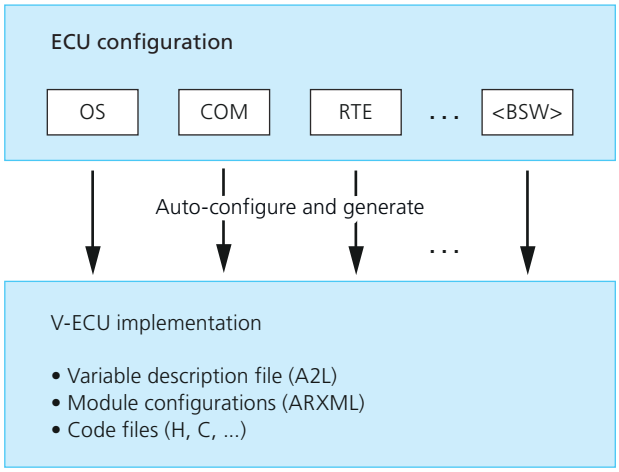
Starting point

In the previous lesson, you learned how to specify and write AUTOSAR master files and export a system extract file.

What will you learn?

Configuring an ECU is a precondition for generating run-time environment (RTE) and BSW code for a virtual ECU.

The schematic below illustrates how the files that are needed for a V-ECU implementation are generated from an ECU configuration.



RTE and BSW code SystemDesk lets you generate code for the RTE and BSW modules according to the related module configurations for the purpose of simulation. You will configure RTE and BSW modules and generate code in this lesson.

V-ECU implementation A V-ECU implementation is the implementation of a virtual ECU for the purpose of simulation. The V-ECU implementation contains code files of application software and basic software as well as additional configuration parameters and an A2L variable description file. The V-ECU implementation is needed for building a virtual ECU (V-ECU) for simulation.

Note

To simplify matters, you will configure only one ECU.
If you run the demo scripts belonging to this lesson, these will configure all the ECUs.

In this lesson:

- You will add an ECU configuration to one ECU in your project.
- You will configure the I/O hardware abstraction that is a basic software module for accessing IO signals. In an offline simulation scenario, the IO signals of the V-ECU are connected to an environment model.

- You will configure the COM stack and the ECU state manager.
- You will create and configure tasks, and map the runnables to them.
- You will configure the OS.
- You will generate code for the run-time environment (RTE) and basic software modules of one ECU.

Before you begin

The TutorialProject you saved in the last lesson must be open.

Tip

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons. To reproduce the result of the previous lessons, run the demo script `Lesson12_complete.py` from the `Tutorial\SD\Scripts` folder in your working folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create an ECU Configuration on page 219
- Step 2: How to Create the I/O Hardware Abstraction on page 223
- Step 3: How to Configure the ECU State Manager and the COM Stack on page 225
- Step 4: How to Map Runnables to OS Tasks on page 228
- Step 5: How to Complete the OS Configuration on page 231
- Step 6: How to Generate the RTE on page 234

Summary

To check if the steps you executed are correct, refer to Result of Lesson 13 on page 237.

Step 1: How to Create an ECU Configuration

Objective

Before starting RTE and BSW code generation, you have to add an ECU configuration to the ECU whose RTE and BSW code you want to generate. You will now add a predefined ECU configuration to the CentralBodyEcu and extend it by the IoHwAb and Dap module configuration elements. The Dap (Data Access Points) module configuration describes the points where a V-ECU accesses IO signals and stimulus signals. Each data access point corresponds to a VPU port in VEOS that can be connected to other V-ECUs or environment models.

The IoHwAb module configuration element is described in more detail in step 2 of this lesson.

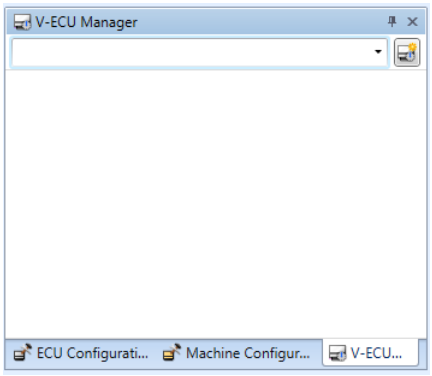
Method

To create an ECU configuration

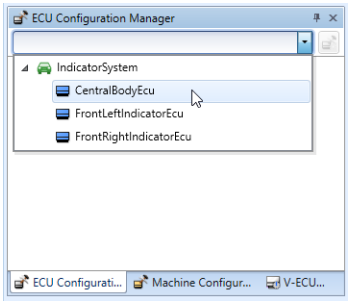
1 On the View ribbon, click Layout – Switch Layout – Integration.

The layout of SystemDesk changes to the integration layout. This layout brings attention to the control bars and managers that you need when you configure and create virtual ECUs (V-ECUs).

2 Select the ECU Configuration Manager.



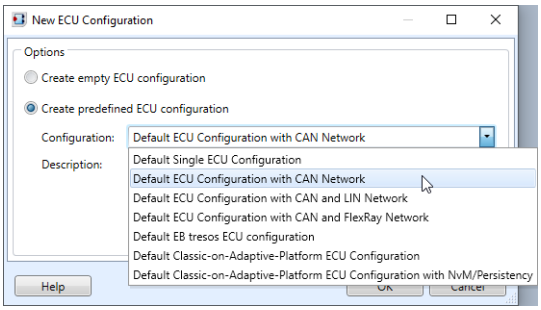
3 In the list of the ECU Configuration Manager, select the CentralBodyEcu.



4 Click , to create a new ECU configuration for the CentralBodyEcu.

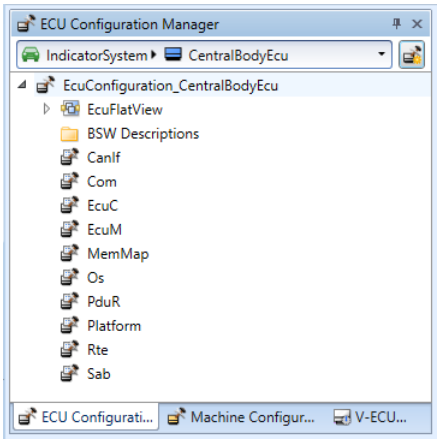
The New ECU Configuration dialog opens.

5 Select Create predefined ECU configuration to use the Default ECU Configuration with CAN Network and click OK to close the dialog.



SystemDesk creates a new ECU configuration in the ECU Configuration Manager. The ECU configuration element contains the minimal set of module configurations that are required for simulation.

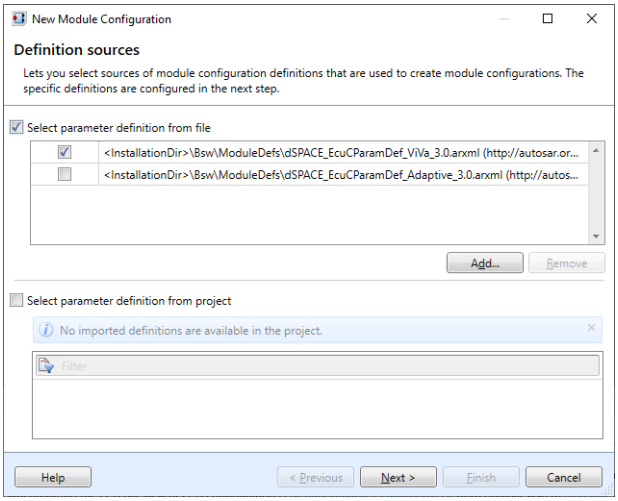
Rename the new ECU configuration element `CentralBodyEcuConfig`.



6 To prepare the following steps of this lesson, you have to add further module configuration elements to the ECU configuration created in step 5.

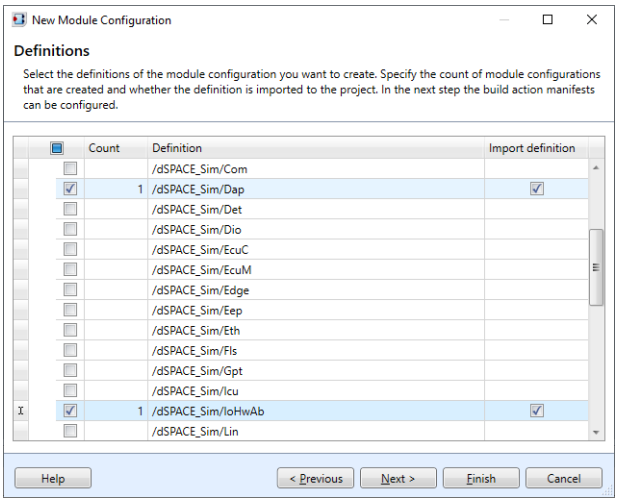
Right-click the CentralBodyEcuConfig and select New – Module Configuration.

7 On the Definition sources page of the New Module Configuration dialog, click Next.



8 On the Definitions page of the New Module Configuration dialog, select `/dSPACE_Sim/Dap` and `/dSPACE_Sim/IoHwAb`.

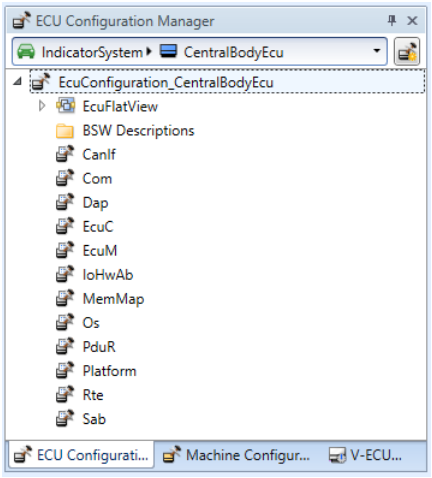
Select **Import Definition** for each module configuration and click **Finish**.



9 SystemDesk adds the **Dap** and **IoHwAb** module configuration elements to the **CentralBodyEcu** ECU configuration.

Result

You have created the initial ECU Configuration for the **CentralBodyEcu**.



Tip

You can define new ECU configurations on the **ECU Configurations page (preferences)** dialog page. You can then access the new ECU configurations from the **ECU Configuration Manager** as shown in steps 4 and 5.

What's next

You can now create the I/O hardware abstraction component in the next step.

Step 2: How to Create the I/O Hardware Abstraction

Objective

Having created an ECU configuration, you can add basic software modules. A basic software component is a component that has ports and interfaces on the RTE level, which can be used by an application software component to access the C-API of a BSW module.

In this step, you will create an *I/O hardware abstraction component*. The component provides I/O driver access to application software components via ports and interfaces.

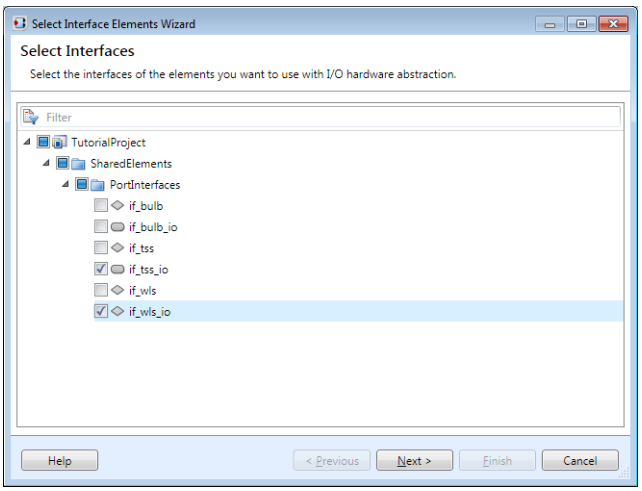
Method

To create the IO hardware abstraction

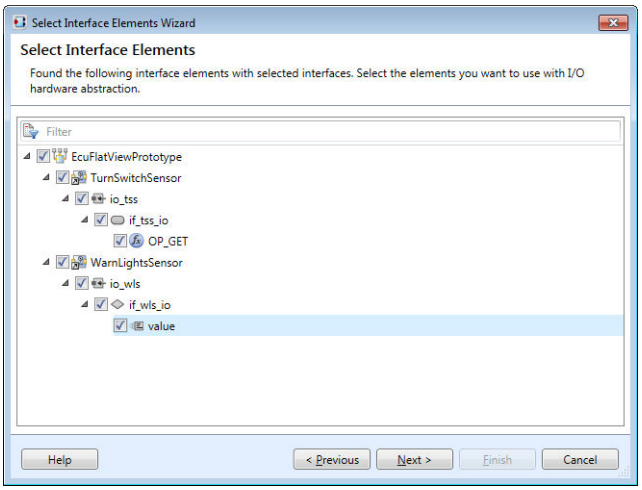
1 In the ECU Configuration Manager, right-click the IoHwAb module configuration and select Select Interface Elements from its context menu.

The Select Interface Elements Wizard opens.

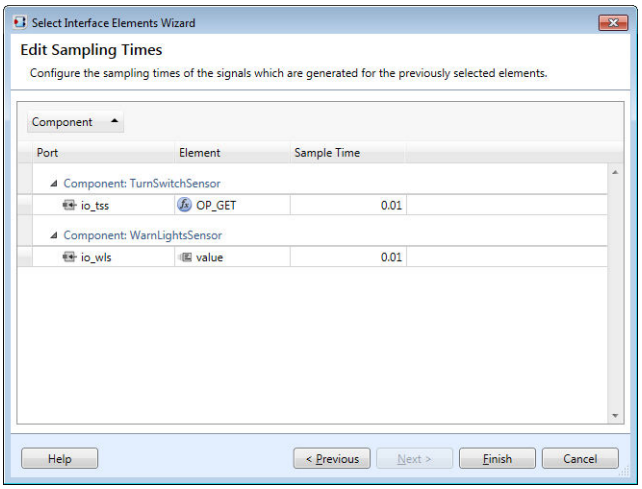
2 In the Select Interface Elements Wizard select the if_tss_io and if_wls_io I/O interfaces belonging to the central body ECU.



3 Click **Next** and select the **TurnSwitchSensor** and **WarnLightsSensor** interface elements.



4 Click **Next** to change to the **Edit Sampling Times** page.

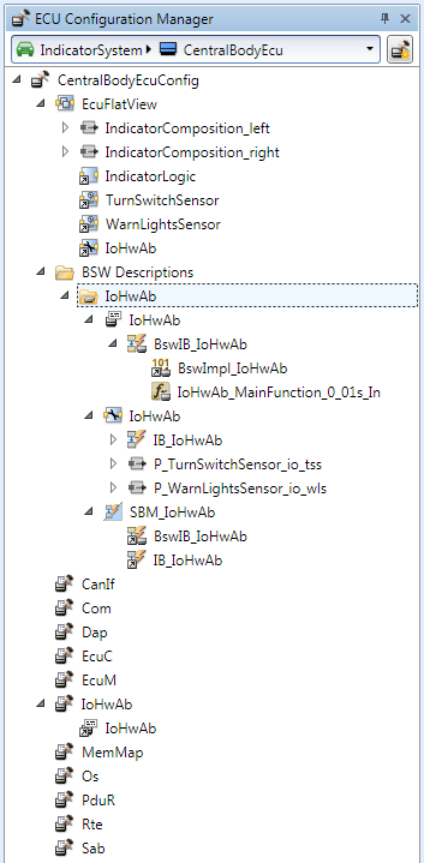


5 Click **Finish** to confirm your settings and close the **Select Interface Elements Wizard**.

6 In the **ECU Configuration Manager**, right-click the **IoHwAb** module configuration and select **Generate IoHwAb Component** from its context menu.

SystemDesk:

- Generates a basic software component representation with ports and interfaces for the I/O hardware abstraction from the **IoHwAb** configuration. SystemDesk also generates an internal behavior and an implementation for it.



- Connects basic SWC ports to application SWC ports.

7 In the **ECU Configuration Manager**, right-click the **CentralBodyEcuConfig** ECU configuration and select **Auto Configure and Generate - 1 Update Configurations** from its context menu.

SystemDesk automatically configures and updates the parameters of all the BSW modules in the ECU configuration.

Result

You created the **IoHwAbSwc** basic software component by specifying the **IoHwAb** module configuration and using its plug-in methods.

What's next

You can now specify the ECU state manager in the next step.

Step 3: How to Configure the ECU State Manager and the COM Stack

Objective

In the last step you used the plug-in methods for the **IoHwAb** module configuration to generate the **IoHwAbSwc** (IoHwAb software component).

You will now configure the ECU state manager (EcuM) and the COM stack (Com) by using SystemDesk's Auto Configure and Generate feature that lets SystemDesk automatically configure module configurations. SystemDesk executes the required plug-ins and follows the BSW module interdependencies. This lets you automatically configure module configurations and update their interdependencies.

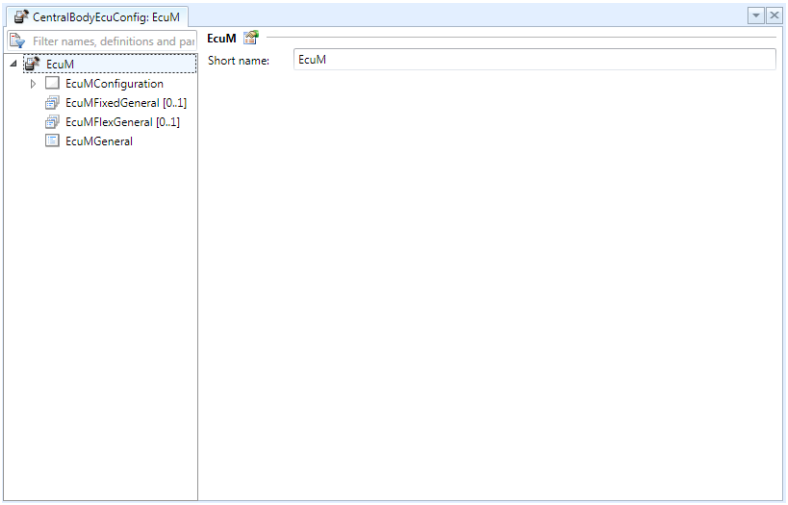
Configuring the EcuM is a precondition for ECU startup. Configuring the communication is a precondition for generating code for the RTE. SystemDesk uses the COM stack for inter-ECU communication. SystemDesk generates COM IPDUs from the data-element-to-system-signal mapping that you specified during system modeling.

Method

To configure the ECU state manager

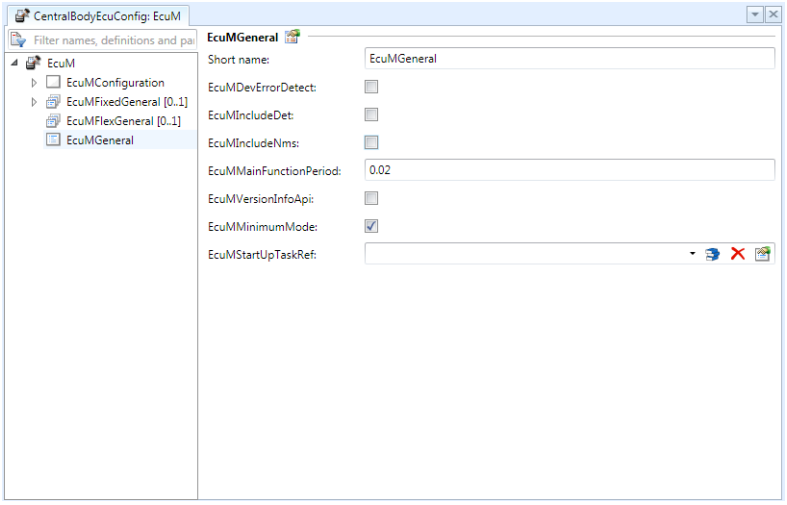
1 In the ECU Configuration Manager, right-click the EcuM module configuration element and select Configure BSW Module from its context menu.

The BSW Module Editor opens.



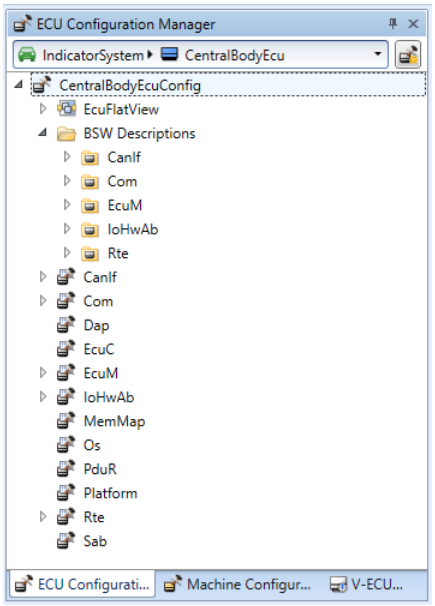
2 In the editor's tree view, specify the following EcuMGeneral properties:

EcuMGeneral Property	Value	Purpose
EcuMMainFunctionPeriod	0.02	To specify the period of the EcuM main function.
EcuMMinimumMode	True	To generate a minimized implementation of the EcuM component that requires no further configuration except for the EcuMMainFunctionPeriod.



3 From the context menu of the CentralBodyEcuConfig ECU configuration, select Auto Configure and Generate - 2 Update Components to configure the ECU state manager and all the other BSW components.

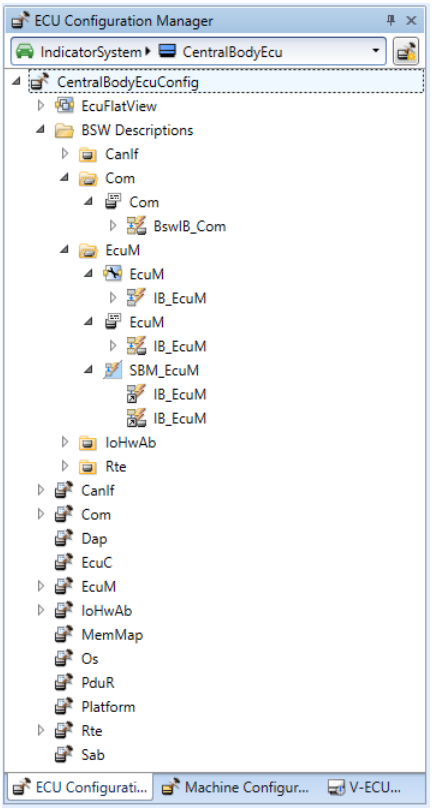
SystemDesk automatically adds all the required BSW modules, module descriptions, and their internal behaviors to the CentralBodyEcuConfig.



Result

You used SystemDesk's Auto Configure and Generate feature to configure the COM stack and ECU state manager of the CentralBodyEcu.

For the Com and EcuM basic software components, SystemDesk created the required internal behaviors.



What's next

You can now map runnables to tasks in the next step.

Step 4: How to Map Runnables to OS Tasks

Objective

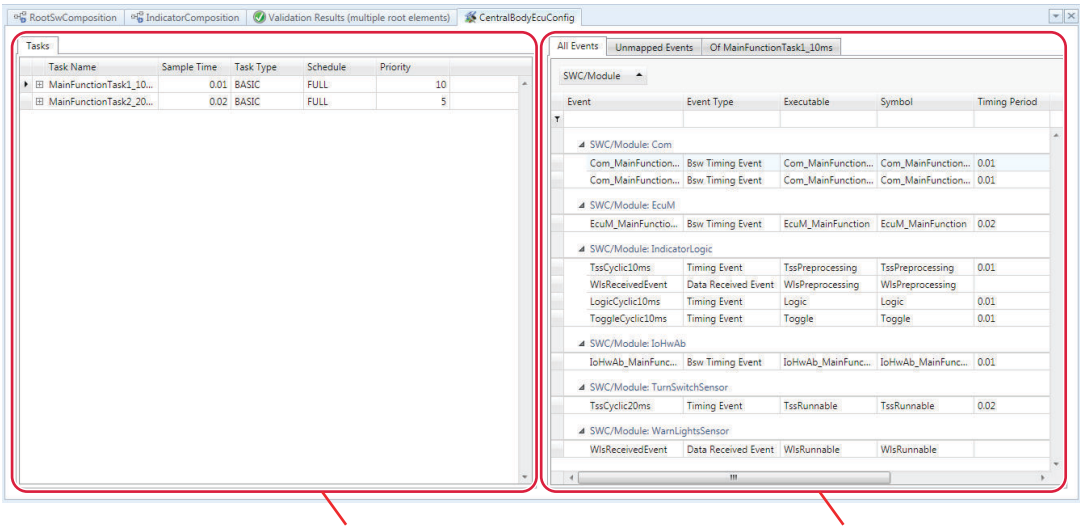
You will now learn how to create OS tasks and to map runnables to them. Runnable-to-OS task mapping defines the execution context of runnables. If an OS task has more than one runnable, you have to specify the execution order of the runnables. You will learn how to work with SystemDesk´s *Runnable Mapping Editor*. You will now create OS tasks and map the runnables of the software components via RTE events. You will also define the execution order in the Task_10ms OS task.

Method

To map runnables to OS tasks

1 In the ECU Configuration Manager, right-click the CentralBodyECUConfig ECU configuration and select Edit Runnable Mapping from its context menu.

The Runnable Mapping Editor opens.



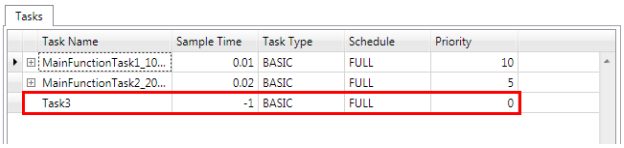
Tasks of the OS configuration

Triggering events of all the runnable entities assigned to the ECU configuration

The Runnable Mapping Editor displays all the tasks of the OS configuration, and all the runnables of the software components mapped to the CentralBodyEcu. Each runnable (executable entity) is triggered by a related RTE event.

2 In the Tasks grid of the Runnable Mapping Editor, right-click the free space and select New task.

SystemDesk adds a new OS task to the Tasks grid.



3 Rename the OS task to `Task_10ms` and set its Sample Time to `0.01`.

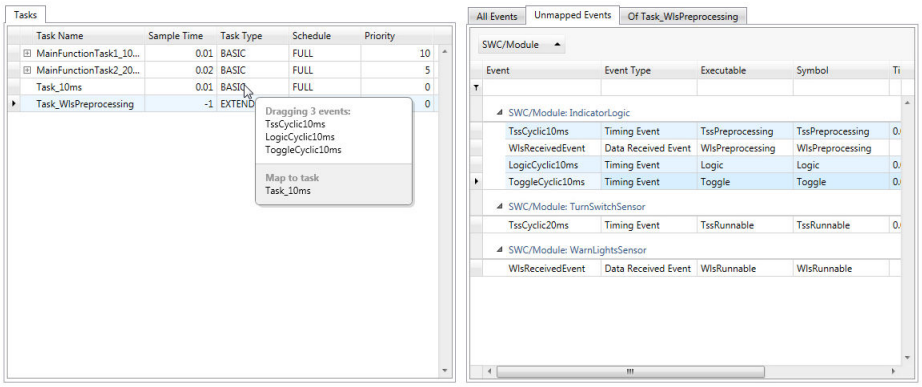
4 Add another OS task and name it `Task_WlsPreprocessing`.

5 Change the Task Type of the Task_WlsPreprocessing task to `Extended` and set its Sample Time to `-1` to specify a non-cyclic task.

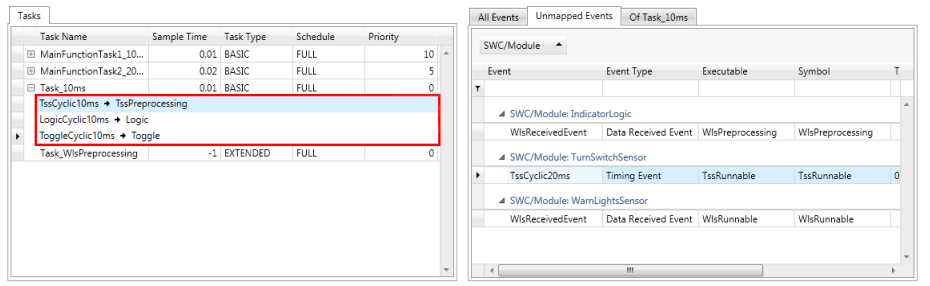
This task will wait for OS events which can be triggered, for example, when a data element is received.

6 In the Events grid of the Runnable Mapping Editor, go to the Unmapped Events page.

Multiselect all runnables having a timing period of **0.01** seconds and drag them to the Task_10ms OS task.



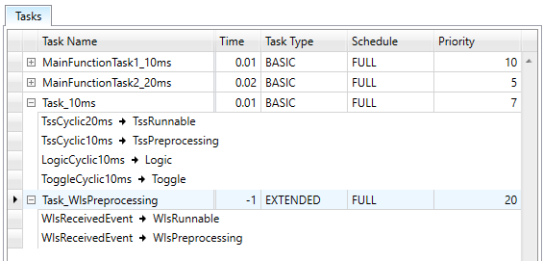
The runnables are mapped to the Task_10ms OS task.



7 Repeat the last step for the TssRunnable. This is possible, because its sample time is a multiple of **0.01**. It will be invoked every second cycle of the Task_10ms OS task.

8 Map the WlsRunnable and WlsPreprocessing runnables to the Task_WlsPreprocessing task.

9 Set the Schedule and Priority properties, and execution order as shown below. You can change the execution order by left-clicking a runnable and using the arrow buttons on the Edit (Runnable Mapping context) – Event ribbon group.



Result

You created two OS tasks and mapped the runnables via RTE events.

It is not necessary to map the EcuM_MainFunction_Event, IoHwAb_MainFunction_0_01s_In_Event, Com_MainFunctionRx_Event, and Com_MainFunctionTx_Event BSW schedulable entities to OS tasks. This is automatically done when you auto configure the COM stack and the ECU state manager.

What's next

You can now complete the OS configuration in the Step 5: How to Complete the OS Configuration.

Step 5: How to Complete the OS Configuration

Objective

You have to adapt the configuration of the operating system of the CentralBodyEcu to execute the runnables of your SWCs. Completing the OS configuration mainly means working with OS tasks, OS events, OS alarms, OS application modes, and OS counters.

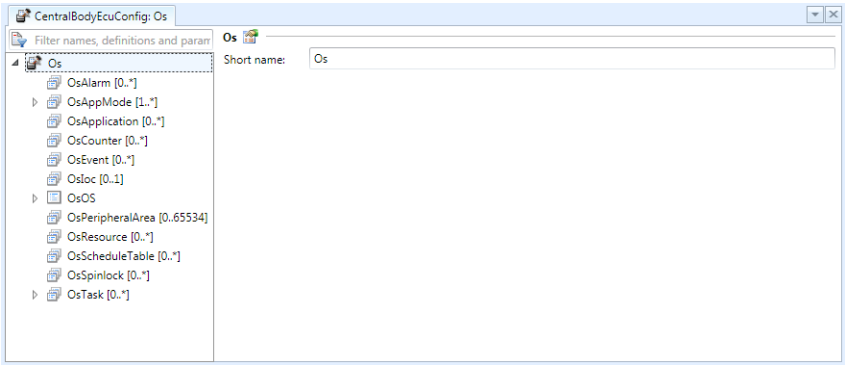
You will now adapt the OS configuration of the CentralBodyEcu.

Method

To complete the OS configuration

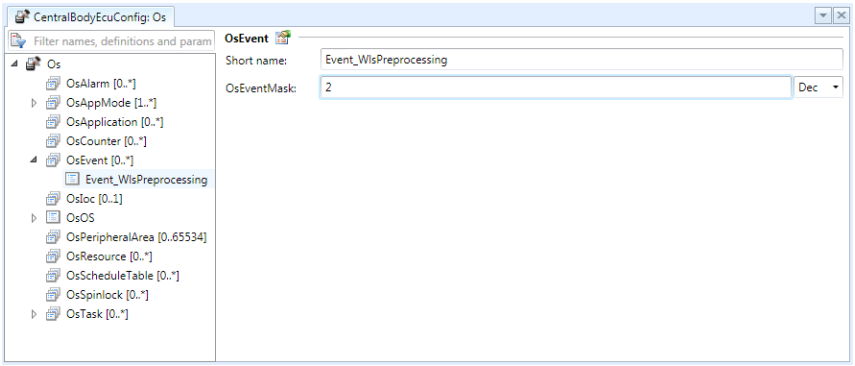
1 In the ECU Configuration Manager, right-click the Os module configuration element and select Configure BSW Module from its context menu.

The BSW Module Editor opens.



2 In the editor's tree view, right-click the OsEvent [0...*] element and select New.

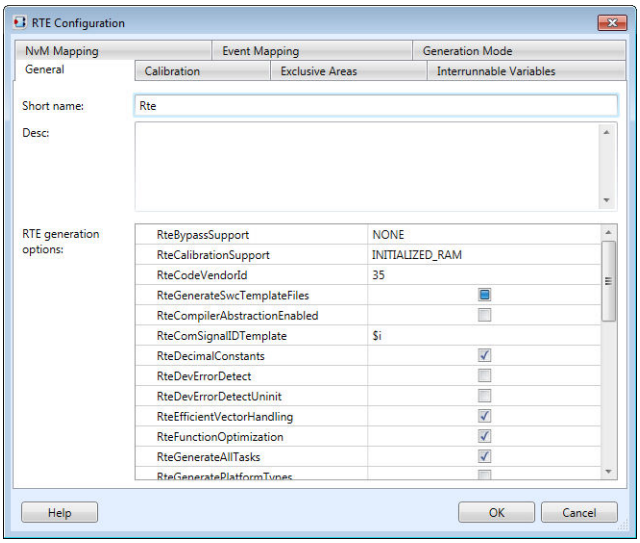
SystemDesk creates a new OS event. Rename the event to `Event_WlsPreprocessing` and specify the event's OsEventMask parameter as `2`.



3 Create another event named `Event_WlsRunnable` and specify its OsEventMask parameter as `1`.

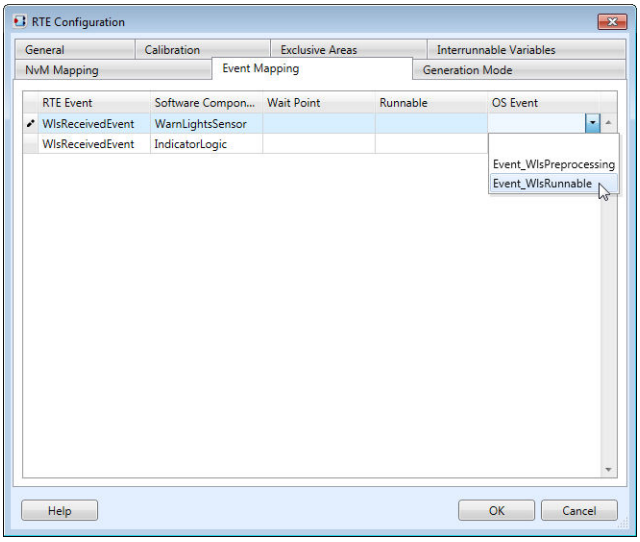
4 In the ECU Configuration Manager, right-click the Rte module configuration and select Configure Rte Module from its context menu.

The RTE Configuration dialog opens.



5 Change to the Event Mapping page.

6 From the list in the **OS Event** column, select **Event_WlsRunnable** to map this OS event to the **WlsReceivedEvent** RTE event of the **WarnLightsSensor** SWC.

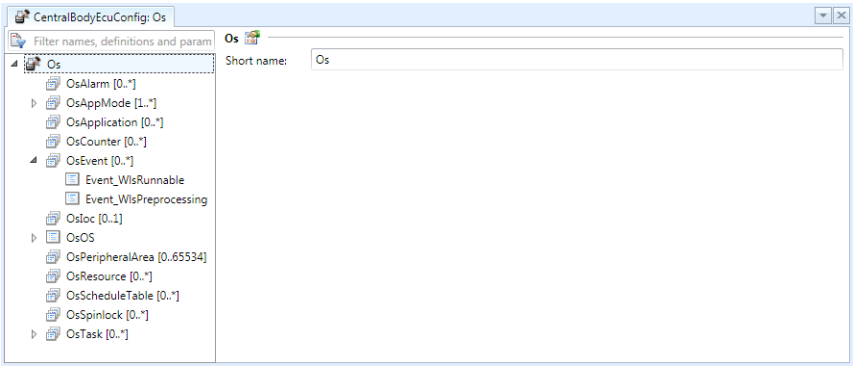


7 Repeat step 6 to map the **Event_WlsPreProcessing** event to the **WlsReceivedEvent** RTE event of the **IndicatorLogic** SWC.

8 Click **OK** to close the **RTE Configuration** dialog.

Result

You configured and completed the OS configuration of the **CentralBodyECU**.



What's next

In this step, you completed the OS Configuration. You can now generate the RTE in the next step (refer to Step 6: How to Generate the RTE on page 234).

Step 6: How to Generate the RTE

Objective

The run-time environment (RTE) is a software layer that connects the ECU application software to the basic software and the operating system. The RTE also interconnects the software components. There is one RTE per ECU, and the generated code is based on the configuration of that ECU.

In the previous steps, you fulfilled all the preconditions for generating the RTE. You configured the CentralBodyECU by performing steps such as mapping runnables to tasks and configuring the OS. In this step, you will now learn how to configure RTE code generation and generate C code for the RTE and all the other BSW modules of the CentralBodyECU.

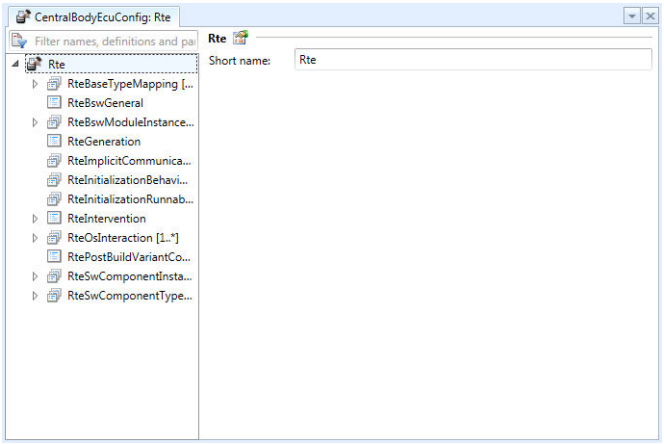
Method

To generate the RTE

1 In the ECU Configuration Manager, select the CentralBodyEcuConfig ECU configuration.

2 Right click the Rte module configuration and select Configure BSW Module from its context menu.

The BSW Module Editor opens.



3 Click RteGeneration and select RteCalibrationSupport – INITIALIZED_RAM.

4 Enable the following RteGeneration properties:
- RteCompilerAbstractionEnabled
- RteFunctionOptimization
- RteMeasurementSupport
- RteTreatUnconnectedPortAsWarning

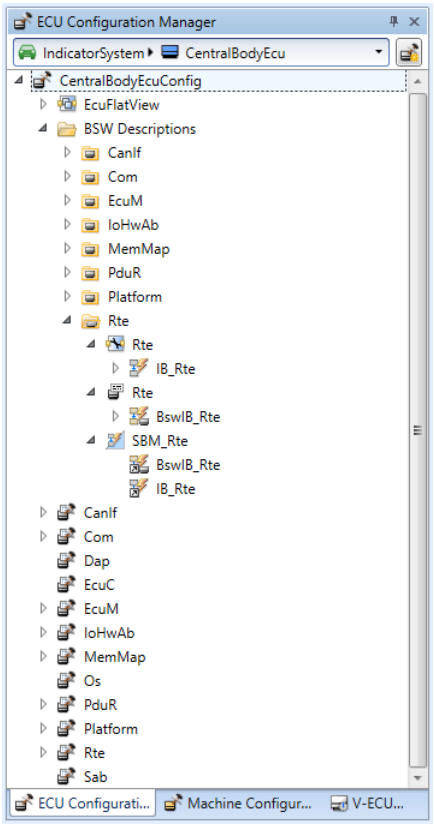
5 Disable the following RteGeneration properties:
- RteGeneratePlatformTypes
- RteGenerateStdAutosarTypes

- RteGenerateStdTypes
- RteMemoryMappingEnabled

6 In the ECU Configuration Manager, right-click the CentralBodyEcuConfig ECU configuration to open its context menu.

7 From the context menu, select Auto Configure and Generate – 3 Generate Code.

SystemDesk checks if all the preconditions for code generation are fulfilled and updates the RTE configuration if necessary. Then it generates code for all the BSW modules, including the RTE. This is necessary so you can build the simulation system in the next lesson.



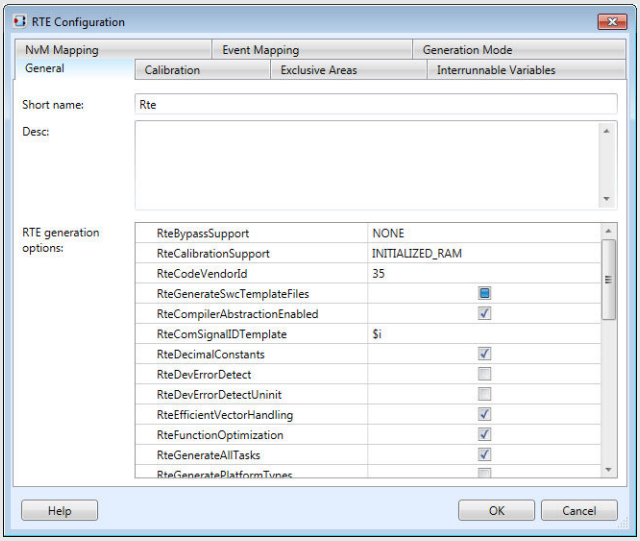
The generated RTE code files are linked in the component's implementation.

Result

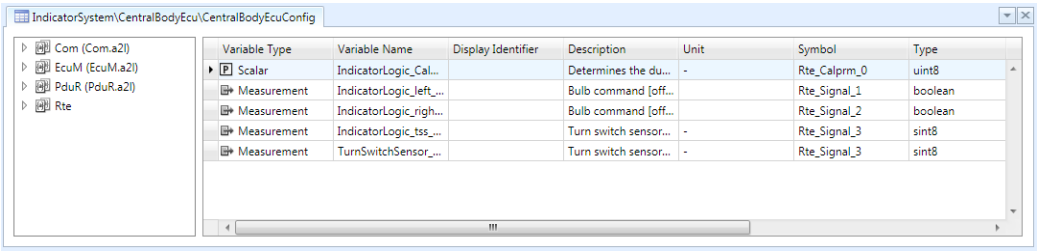
In this step, you learned how to configure the run-time environment (RTE) and generate code for all the BSW modules including the RTE.

Tip

- If you use the Auto Configure and Generate – V-ECU Implementation command for an ECU configuration, SystemDesk executes all the required plug-in methods for the basic software modules of the selected ECU configuration. This convenient command includes the 1 Update Configurations, 2 Update Components, and 3 Generate Code commands.
- Instead of the BSW Module Editor used in this step, you can also use the Configure Rte Module command that opens the RTE Configuration Properties dialog to configure the RTE.
  To open the dialog, right-click the Rte module configuration element and select Configure Rte Module from its context menu.



Generated variables You can display a list showing all the measurement variables and calibration parameters that are generated for the RTE. To open the variable list in the working area, right-click the CentralBodyEcuConfig ECU configuration in the ECU Configuration Manager and select Internal A2L Variables – Show. The variable list shows all the variables that can be exported to an A2L file.



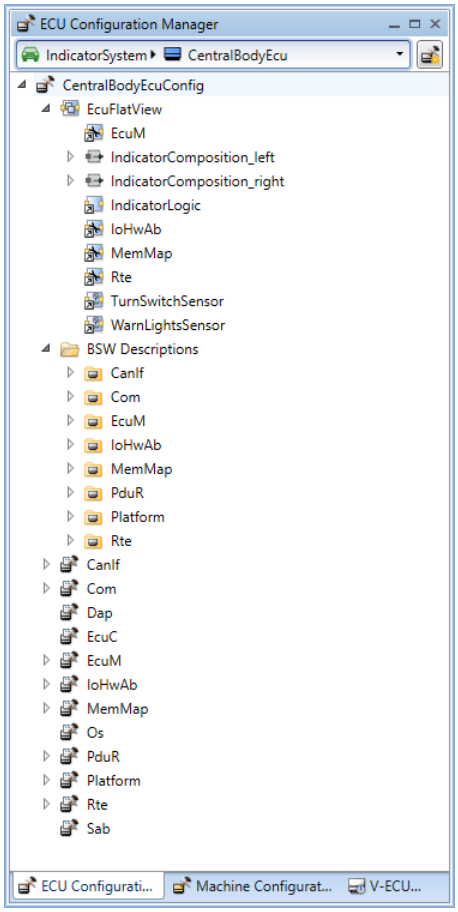
What's next

You have generated the RTE code. Now check your work against the result you should have achieved. Refer to Result of Lesson 13 on page 237.

Result of Lesson 13

Summary

If you have done all the steps in this lesson, your ECU configuration should look like this:



In this lesson, you learned how to configure an ECU. You modeled the ECU configuration and configured its basic software modules. You created tasks and mapped the runnables to them.

You can look at the generated code files for the basic software modules and the RTE in the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\IndicatorSystem\CentralBodyEcuConfig` folder.

Demo

You can reproduce the result of this lesson with the demo script `Lesson13_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder.

Note

The script reproduces the results of the previous lessons and extends your project by ECU configurations and basic software components not only for the **CentralBodyEcu**, but for all the ECUs of the tutorial project.

Further information

For detailed information on configuring ECUs, refer to Configuring ECUs (🕮 SystemDesk Manual).

This section includes the following information:

- Rules for mapping runnables. Refer to Basics on Runnable Entities Mapping (🕮 SystemDesk Manual).
- A section with information on configuring dSPACE BSW modules. Refer to Using dSPACE Basic Software Modules for Virtual Validation (🕮 SystemDesk Manual).

What's next

The next lesson shows you how to build the simulation system.

Lesson 14: Generating V-ECU Implementations

Where to go from here

Information in this section

Overview of Lesson 14 240
In this lesson, you will generate V-ECU implementations and export them in V-ECU implementation containers.
You will also learn how to perform VEOS test builds for V-ECUs and how to build a simulation system in VEOS.

Step 1: How to Create V-ECUs 241
In this step, you will create V-ECUs for all the ECU configurations of the IndicatorSystem system.

Step 2: How to Perform a V-ECU Test Build 244
In this step, you will learn how to perform test builds of V-ECUs.

Step 3: How to Export V-ECU Implementation Containers 246
In this step, you will learn how to export V-ECU implementation containers.

Step 4: How to Build a Simulation System with VEOS 248
In this step, you will learn how to build a simulation system with VEOS.

Result of Lesson 14 253
In this lesson, you learned how to generate and export V-ECU implementations and how to perform V-ECU test builds. You also learned how to build a simulation system with VEOS.

Overview of Lesson 14

Starting point

In the previous lesson, you learned how to model ECU configurations and basic software components.

What will you learn?

To simulate the behavior of a system you need a simulation system, which can consist of virtual ECUs (V-ECUs), optional environment models, and optional interconnections.

A virtual ECU (V-ECU) represents a real ECU or parts of it in a simulation scenario. The V-ECU comprises components from the application software and from the basic software. It provides functionalities comparable to those of a real ECU. The V-ECU implementation comprises code files of application software and basic software and can be used for building a simulation application for offline or real-time simulations without modification. An environment model represents a part or all of an ECU's environment in a simulation scenario.

SystemDesk lets you export V-ECU implementations in container files to share them with other tools. For example, you can import a V-ECU implementation container to VEOS to build an offline simulation application (OSA).

In this lesson, you will learn how to perform the following tasks:

- Generate V-ECU implementations from ECU configurations.
- Perform test builds for V-ECUs (requires a VEOS installation).
- Export V-ECU implementation containers for later use. This is independent of an existing VEOS installation on your PC.
- Build a simulation system in VEOS, which results in an offline simulation application (OSA file). You will only be able to perform this step if you have VEOS installed on your PC.

In the graphical user interface, the term V-ECU is used synonymously with V-ECU implementation. The instructions below follow this practice.

Before you begin

Because this tutorial is modular, you can work through a lesson without having completed the previous lessons.

Note

The starting point of this lesson is the demo script `Lesson13_complete.py`. The script reproduces the results of the previous lessons and extends your project by configuring and generating a V-ECU Implementation not only for the CentralBodyEcu, but for all the ECUs of the tutorial project.

Run the demo script `Lesson13_complete.py`. You can find it in the `Tutorial\SD\Scripts` folder.

Steps

This lesson contains the following steps:

- Step 1: How to Create V-ECUs on page 241
- Step 2: How to Perform a V-ECU Test Build on page 244
- Step 3: How to Export V-ECU Implementation Containers on page 246
- Step 4: How to Build a Simulation System with VEOS on page 248

Summary

To check if the steps you executed are correct, refer to Result of Lesson 14 on page 253.

Step 1: How to Create V-ECUs

Objective

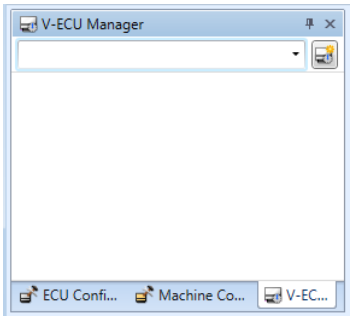
SystemDesk provides the V-ECU Manager to create and handle single V-ECUs.

In this step, you will create V-ECUs for all the ECU configurations of the IndicatorSystem system.

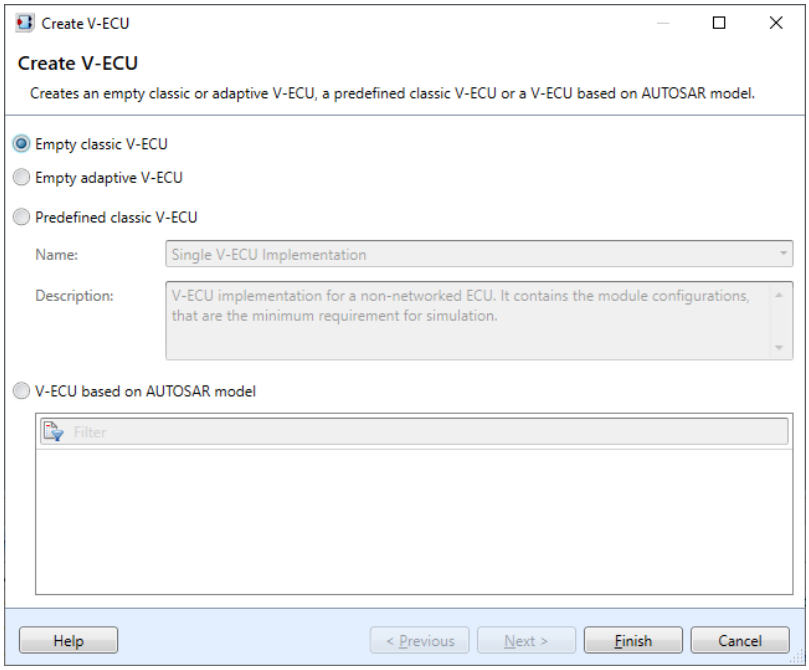
Method

To create a V-ECU

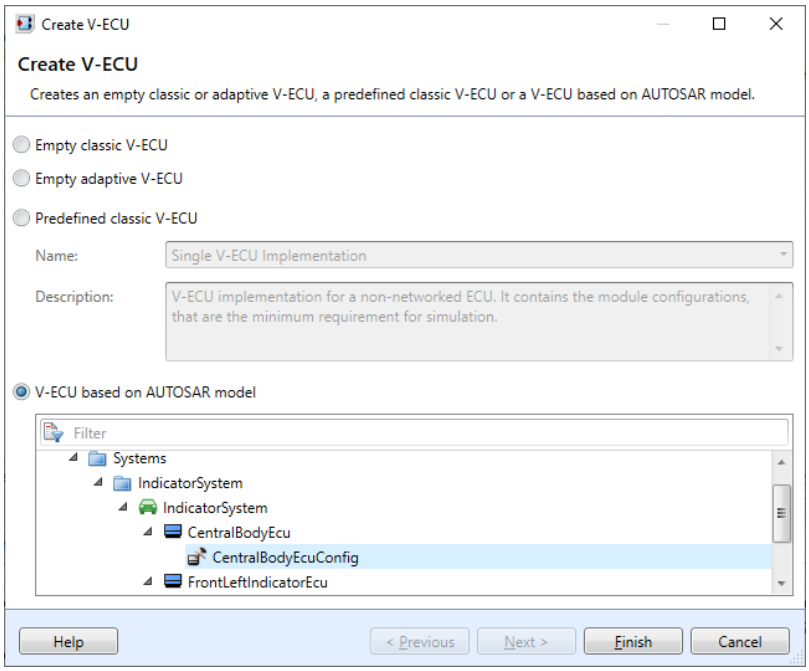
1 Select the V-ECU Manager.



2 In the V-ECU Manager, click .
SystemDesk opens the Create V-ECU dialog.

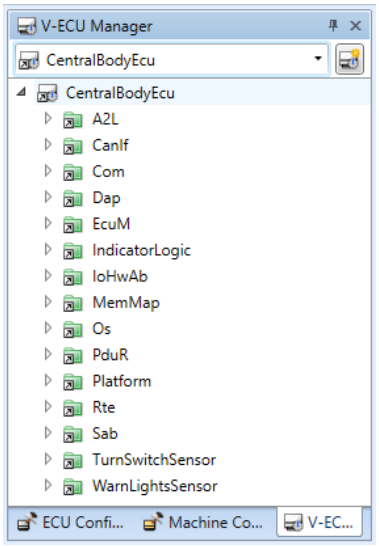


3 In the dialog, select V-ECU based on AUTOSAR model and select the CentralBodyEcuConfig ECU configuration.



4 Click Finish.

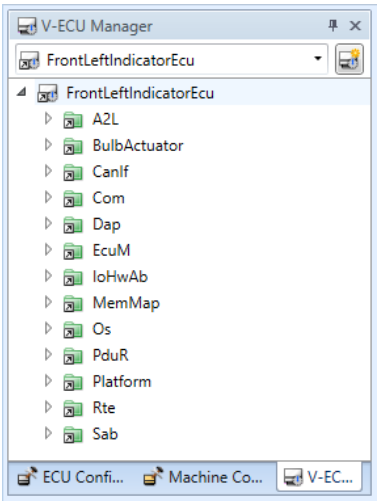
SystemDesk creates a V-ECU based on the CentralBodyEcuConfig ECU configuration.



5 Repeat steps 2 to 4 for the FrontLeftIndicatorEcuConfig and the FrontRightIndicatorEcuConfig ECU configurations.

Result

You created V-ECUs for all the ECU configurations of the IndicatorSystem system. The V-ECU Manager displays the selected V-ECU.



What's next

In the next step, you will learn how to perform test builds of V-ECUs. This requires a VEOS installation.

If you do not have a VEOS installation, proceed with Step 3: How to Export V-ECU Implementation Containers on page 246, where you will learn how to export V-ECU implementation containers.

Step 2: How to Perform a V-ECU Test Build

Objective

V-ECU implementations contain all input files required to perform the build for a specific simulation target. You can use SystemDesk to initiate a test build. This lets you detect errors before you load a V-ECU to a real simulation target.

SystemDesk can automatically generate build scripts for VEOS. You can adjust the build scripts, for example, to use them in automated continuous integration.

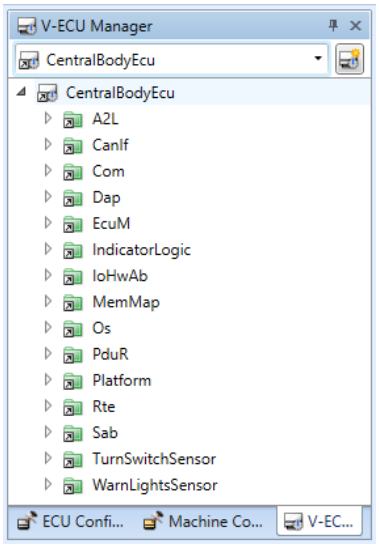
In this step, you will learn how to perform a test build for a V-ECU. This requires a VEOS installation.

If you do not have a VEOS installation, proceed with step 3 of this lesson.

Method

To perform a V-ECU test build

1 In the V-ECU Manager, select the CentralBodyEcu V-ECU.

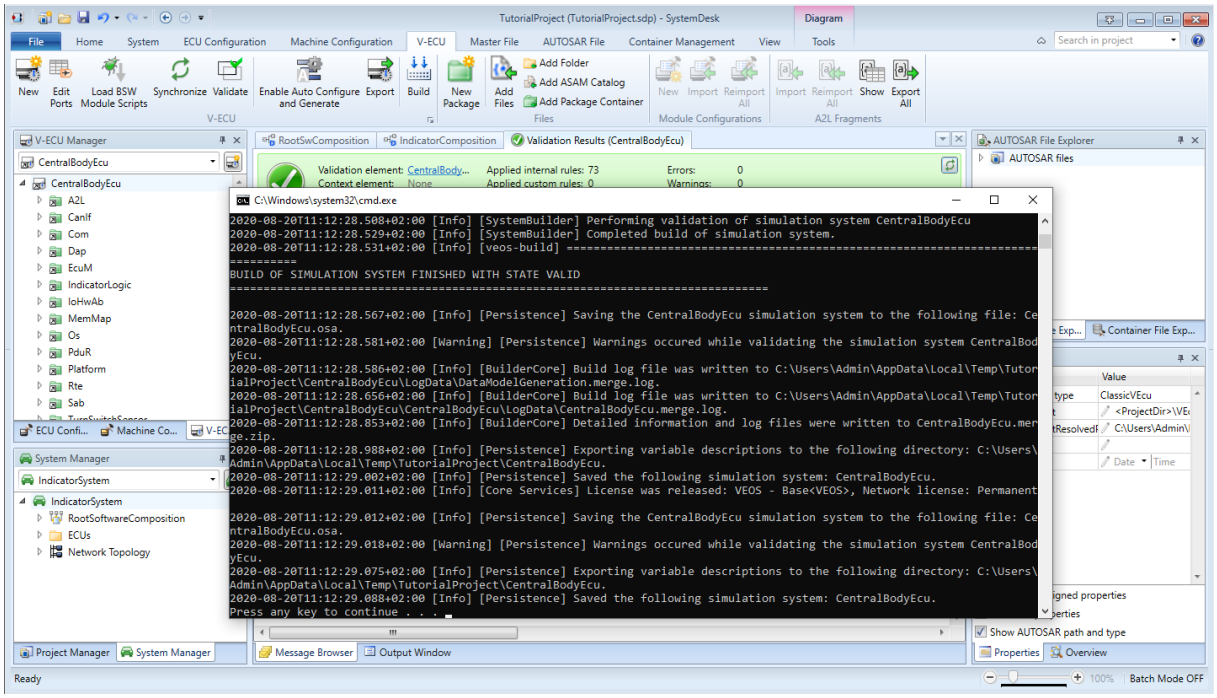


2 On the V-ECU ribbon, click V-ECU - Build.

SystemDesk generates the `CentralBodyEcu.py` build script and executes it.

The build script exports the CentralBodyEcu V-ECU in a V-ECU implementation container and calls the `veos-build.exe` command line tool, which performs the build.

The veos-build tool displays its progress in the command line window, as shown in the following illustration.

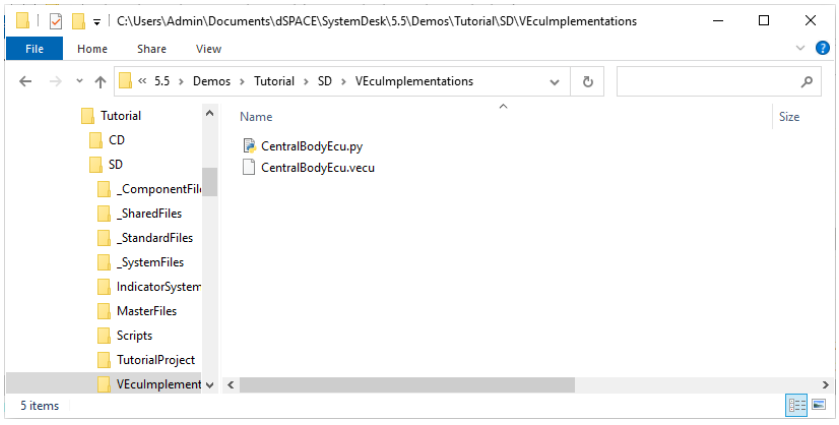


Result

You performed a test build of the CentralBodyEcu V-ECU.

The `CentralBodyEcu.vecu` V-ECU implementation container is saved to the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\VEcuImplementations` directory.

The `CentralBodyEcu.py` build script is saved to the same directory.



The offline simulation application into which the V-ECU is built is saved under

`<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\VEcuImplementations\CentralBodyEcu\Build\CentralBodyEcu.osa`.

You can repeat this step for the other V-ECUs of the **IndicatorSystem** system.

What's next

In the next step, you will learn how to export V-ECU implementation containers.

Step 3: How to Export V-ECU Implementation Containers

Objective

To use V-ECUs in simulation scenarios, you must export them as V-ECU implementation containers.

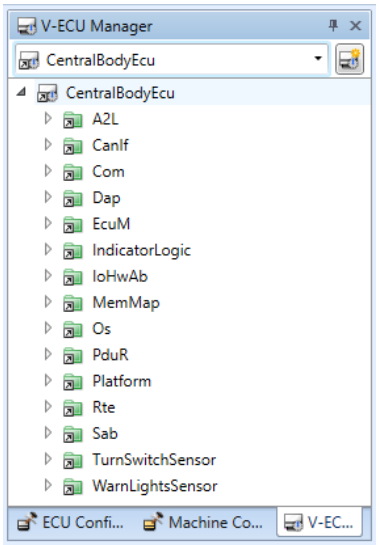
V-ECU implementation containers contain all input files required to perform the build for a specific simulation target.

In this step, you will export the V-ECUs you created in step 1 of this lesson in V-ECU implementation containers.

Method

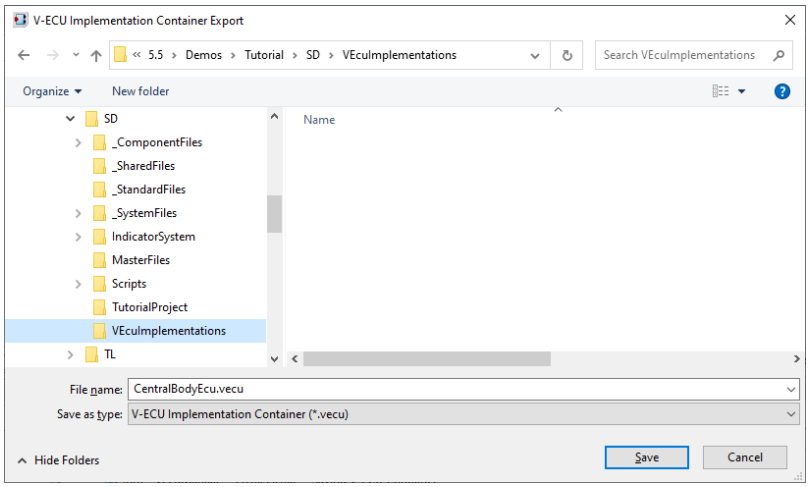
To export V-ECU implementation containers

1 In the V-ECU Manager, select the CentralBodyEcu V-ECU.



2 On the V-ECU ribbon, click V-ECU - Export.

SystemDesk opens the V-ECU Implementation Container Export dialog.



3 In the dialog, navigate to the directory to which you want to save the V-ECU implementation container and click Save.

SystemDesk exports the CentralBodyEcu V-ECU in the `CentralBodyEcu.vecu` V-ECU implementation container.

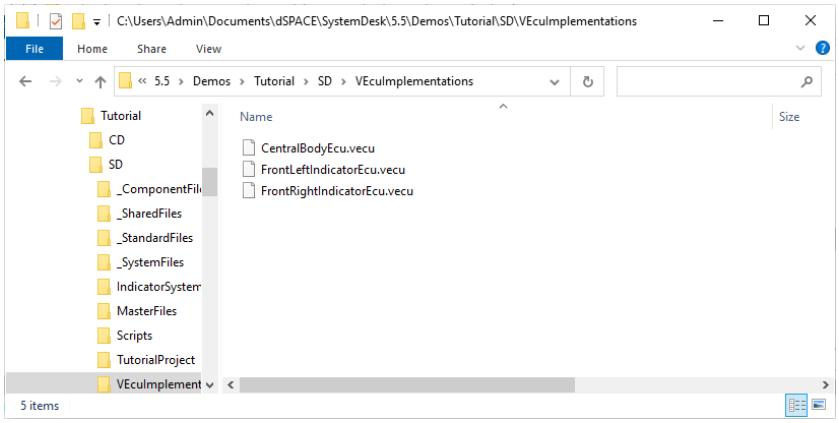
Note

If you performed the test build described in Step 2: How to Perform a V-ECU Test Build on page 244, the `CentralBodyEcu.vecu` file already exists in the default export location `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\SD\VEcuImplementations`. You can replace it with the new V-ECU implementation container.

4 Repeat steps 1 to 3 for the FrontLeftIndicatorEcu and FrontRightIndicatorEcu V-ECUs.

Result

You exported V-ECU implementation containers for all the V-ECUs you created in step 1 of this lesson.



What's next

In the next step, you will learn how to create a simulation system with VEOS using the V-ECU implementation containers you exported in this lesson. This requires a VEOS installation.

If you do not have a VEOS installation, you completed the tutorial with this lesson. Refer to Result of Lesson 14 on page 253.

Step 4: How to Build a Simulation System with VEOS

Objective

Building a simulation system with VEOS means building an offline simulation application. This offline simulation application simulates the corresponding simulation system on a PC. It comprises binary files for the single V-ECUs of the simulation system and is stored as an offline simulation application (OSA) file.

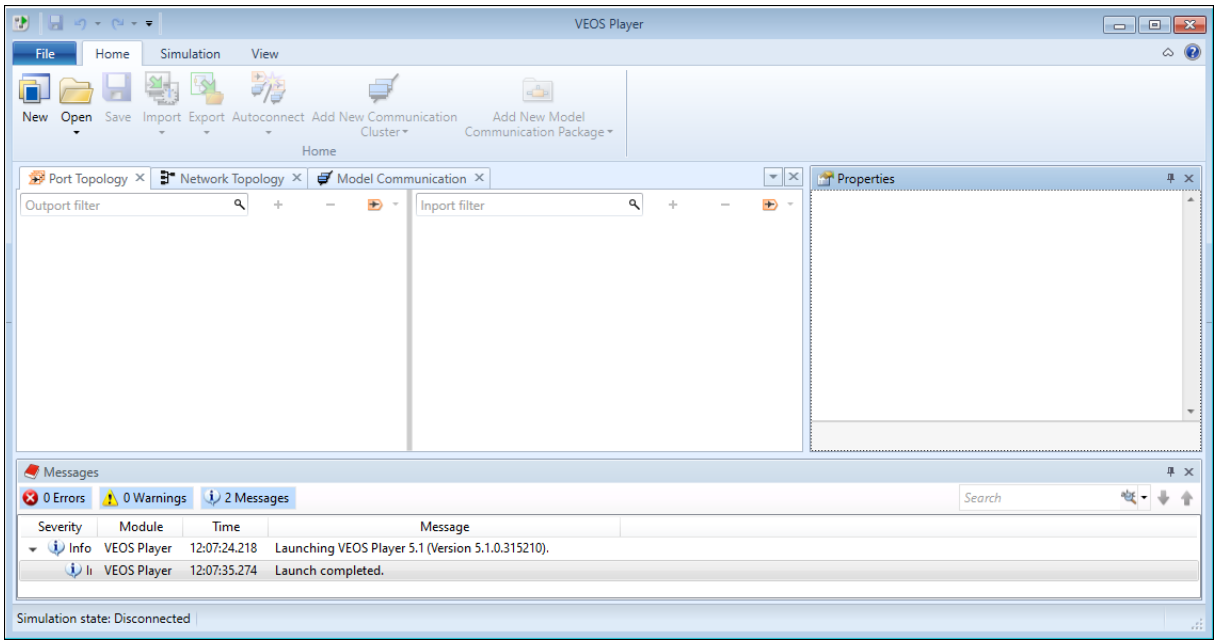
Generally, a simulation system can also contain one or more environment models for closed-loop simulations. For simplicity, this tutorial does not use environment models.

In this step, you will learn how to build a simulation system with the VEOS Player, which is the graphical user interface of VEOS.

Method

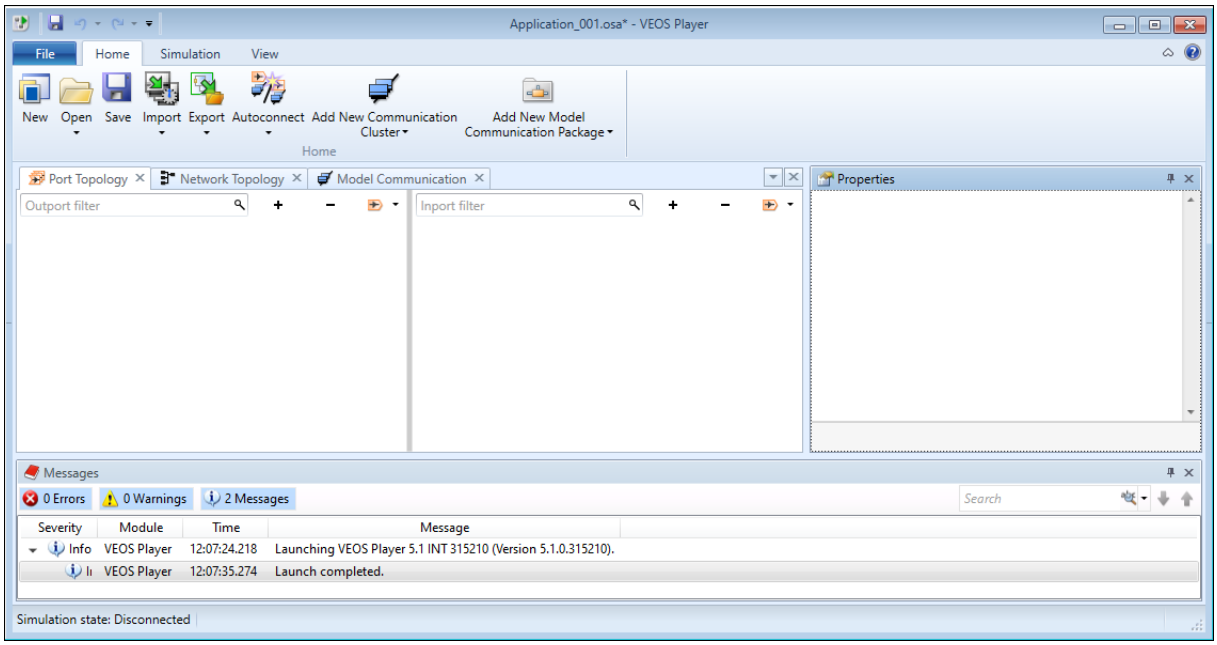
To build a simulation system with VEOS

1 Start the VEOS Player via Start - dSPACE VEOS 5.1 - dSPACE VEOS Player 5.1.



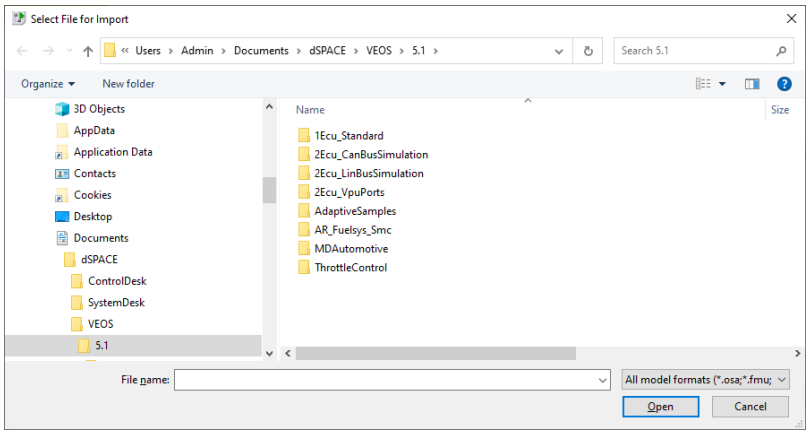
2 On the Home ribbon, click New.

VEOS creates an empty offline simulation application.



3 On the **Home** ribbon, click **Import**.

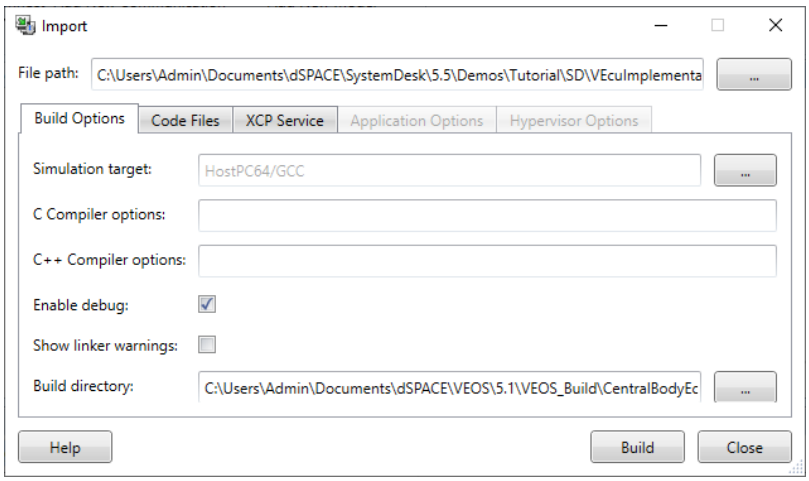
VEOS opens the **Select File for Import** dialog.



4 Navigate to the directory to which you saved the V-ECU implementation containers in step 3 of this lesson and select the `CentralBodyEcu.vecu` file.

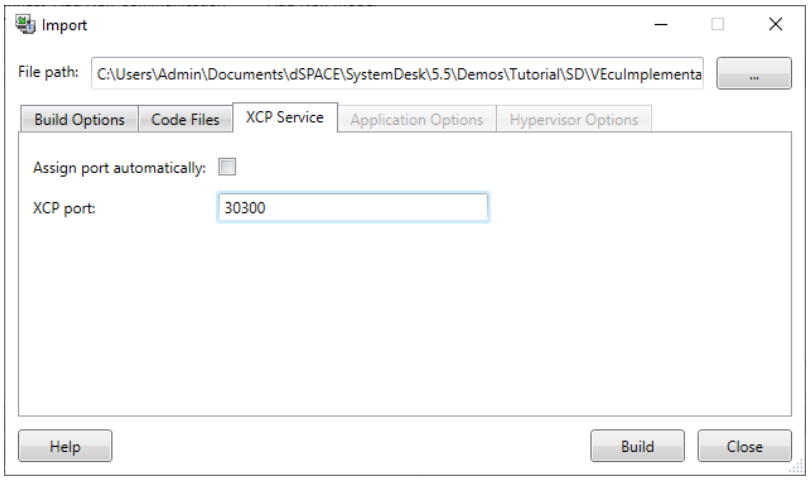
Click **Open**.

VEOS opens the **Import** dialog.



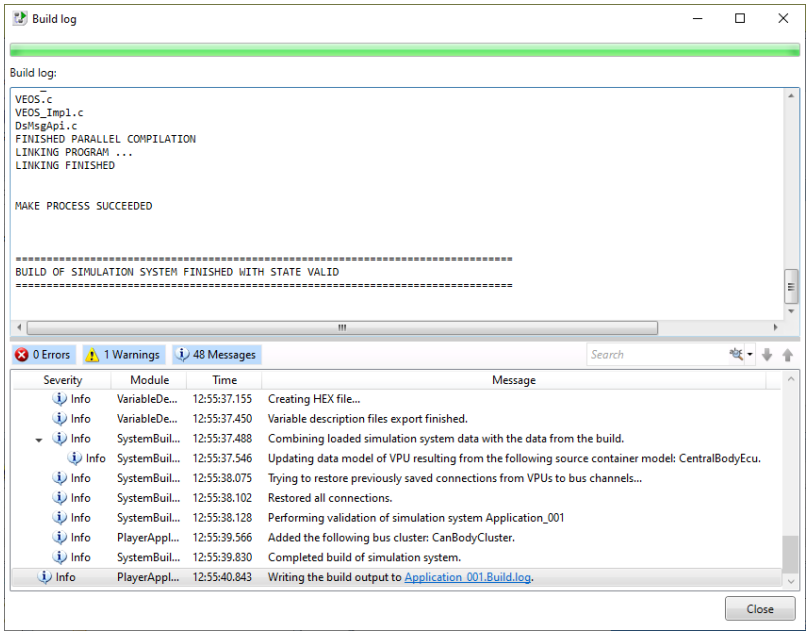
5 In the Import dialog, select the XCP Service page.

On the page, clear the Assign port automatically checkbox and enter `30300` in the XCP port edit field.



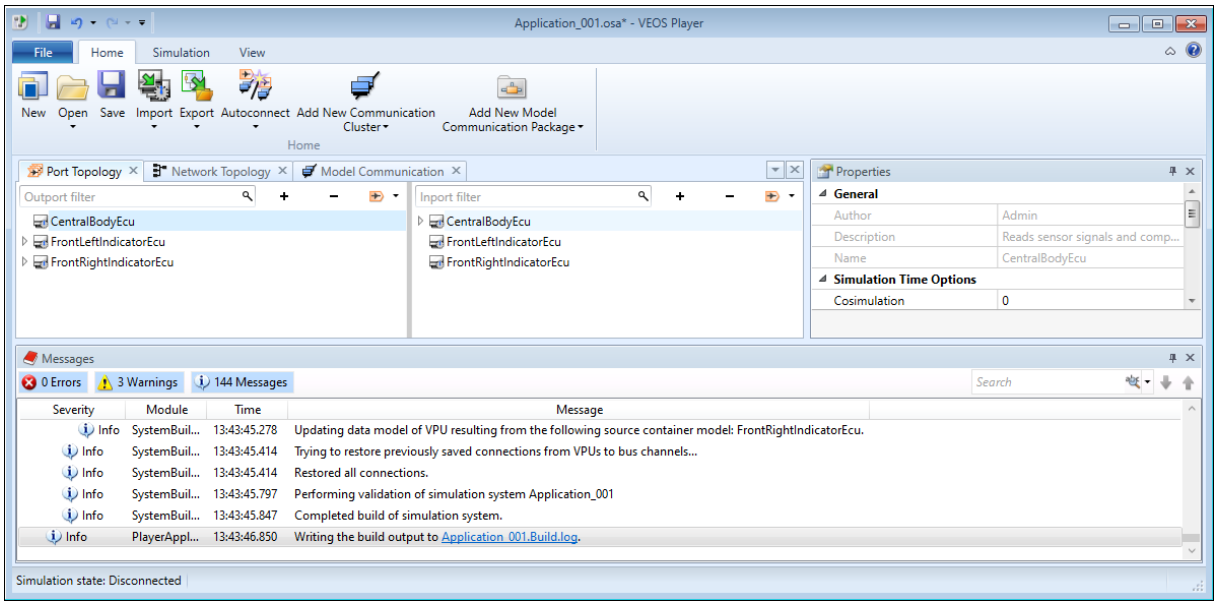
6 Click Build.

VEOS builds the V-ECU and imports the resulting VPU to the simulation system.



7 Repeat steps 3 to 6 for the `FrontLeftIndicatorEcu.vecu` (XCP port: 30302) and the `FrontRightIndicatorEcu.vecu` (XCP port: 30304) V-ECU implementation containers.

When you are finished, the OSA should look like this:



8 Click Home - Save and save the OSA to `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\VEOS\IndicatorSystem\IndicatorSystem.osa`.

Result

You built a simulation system with VEOS consisting of the central body V-ECU, the front left indicator V-ECU, and the front right indicator V-ECU as well as their network connection via CAN.

The build results are stored in the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\VEOS\IndicatorSystem` folder you specified in this step.

The folder contains the following items:

- The offline simulation application (OSA) file you need for offline simulation with VEOS
- Variable description (A2L) files for each V-ECU you need for experimenting and testing with ControlDesk
- A subfolder with temporary files for each V-ECU

What's next

With this step, you completed the lesson. Now check your work against the result you should have achieved. Refer to Result of Lesson 14 on page 253.

Result of Lesson 14

Summary

In this lesson, you learned how to generate export V-ECU implementations and how to perform V-ECU test builds. You also learned how to build a simulation system with VEOS.

With this lesson, you successfully completed this tutorial, which does not cover simulating the TutorialProject simulation system.

Demo

You can reproduce the result of this lesson with the demo script `Lesson14_complete.py` located in the `Tutorial\SD\Scripts` folder of your working folder. Depending on whether you have a VEOS installation, the script will either build a simulation system or simply export V-ECU implementation containers.

Further information

For detailed information on building and simulating a simulation system, refer to Integrating the Simulation System (🕮 VEOS Manual).

What's next

Working with V-ECU implementation containers For details on integrating V-ECU implementation containers into an offline simulation application in VEOS, refer to Importing V-ECU Implementations (🕮 VEOS Manual).

You can also use SYNECT to do this. Refer to Integrating System Models (🕮 SYNECT Guide).

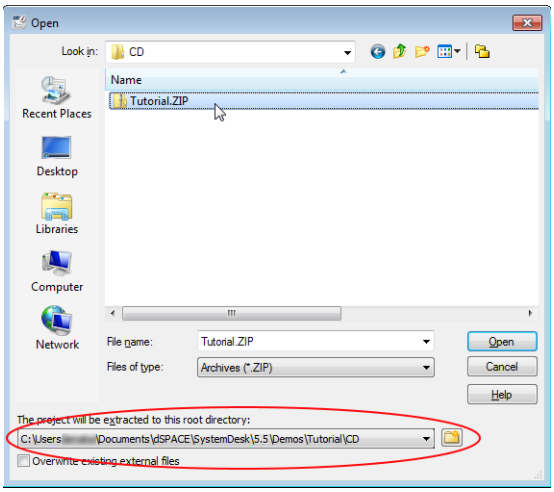
You can also add the V-ECU implementation containers to a ConfigurationDesk application. Refer to Adding V-ECU Implementations to a ConfigurationDesk Application (🕮 ConfigurationDesk Real-Time Implementation Guide).

Working with the ControlDesk demo project The SystemDesk tutorial includes the `Tutorial.CDP` ControlDesk demo project with a preconfigured experiment for measuring signals generated in a VEOS simulation of the indicator system you modeled in the tutorial. If you do not have a VEOS installation, you can load the project and experiment in ControlDesk, but you will not be able to perform the simulation and related measurements.

To work with the ControlDesk demo project included in the SystemDesk installation, proceed as follows:

1. Start ControlDesk.
2. On the File ribbon, click Open and select Open Project and Experiment from Backup.

3. In the Open dialog, make sure that the root directory is set to `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\CD`.



   Navigate to the `<My Documents>\dSPACE\SystemDesk\5.5\Demos\Tutorial\CD` folder and open the `Tutorial.ZIP` file.

   If a **Project Manager** dialog is displayed informing you that the project needs to be converted, click **Yes**.

   ControlDesk loads the **Tutorial** project and activates the **EXP_VEOS_Tutorial** experiment.

4. The ControlDesk project is delivered ready-to-use with a pre-built **IndicatorSystem.osa** file that is loaded to the VEOS platform, i.e., you can start the simulation immediately.

   If you want to use the **IndicatorSystem.osa** file you built in step 2 of lesson 14, open its context menu in the **Project Manager** and select **Reload system**.

   ControlDesk opens a **Platform Management** dialog prompting you to confirm your choice. Click **Yes**.

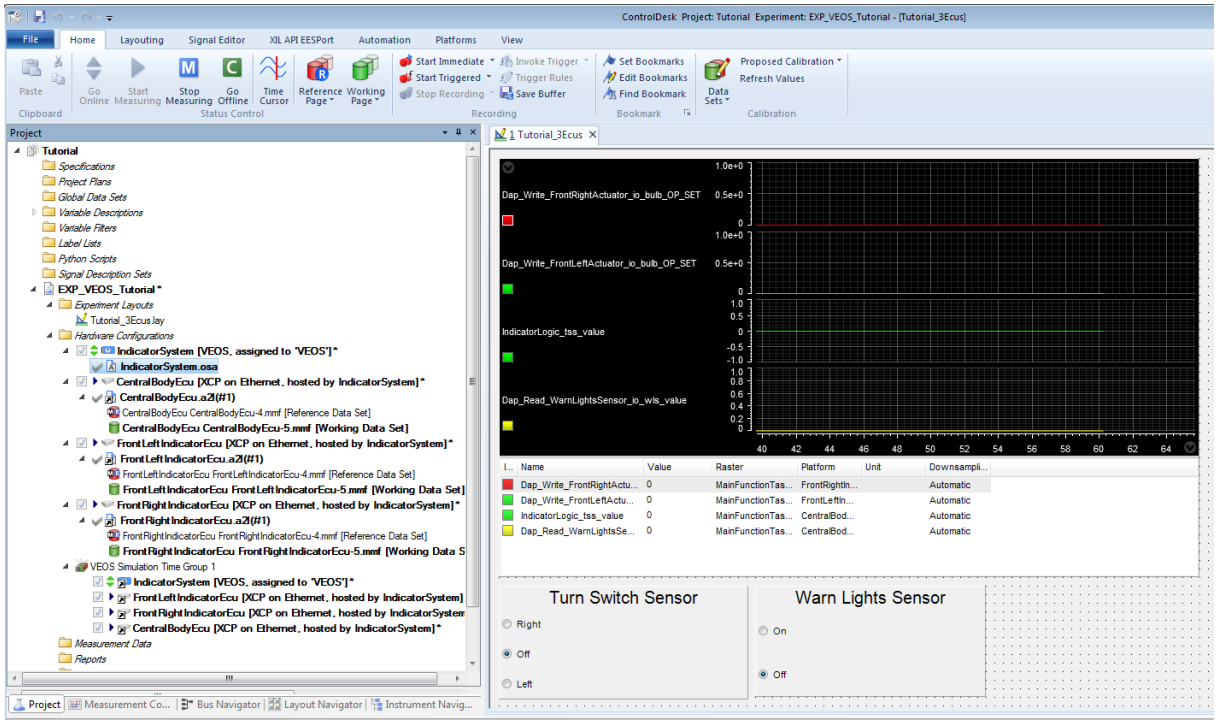
   A **DataSetManager** dialog opens asking you to confirm that the data sets of the active variable description will be restored after the new variable description is loaded. Click **Yes**.

   ControlDesk displays the **Select ECU Interface Settings** dialog, which shows the differences between ECU interface settings currently used by the device and those which have been imported from the variable description. Click **OK**.

Tip

The VEOS platform might be displayed as disconnected ( ) in the ControlDesk Project controlbar. If VEOS is displayed in the Platforms/Devices controlbar, you can proceed with the next step. However, if VEOS is *not* displayed in the Platforms/Devices controlbar, you have to register VEOS and assign it to the VEOS platform in the EXP_VEOS_Tutorial experiment before you can start measuring. Refer to How to Register a Platform (ControlDesk Platform Management) and How to Assign dSPACE Real-Time Hardware or VEOS to a Platform (ControlDesk Platform Management).

5. On the Home ribbon, click Start Measuring to start the measurement.
6. ControlDesk starts the measurement, as shown in the following illustration.

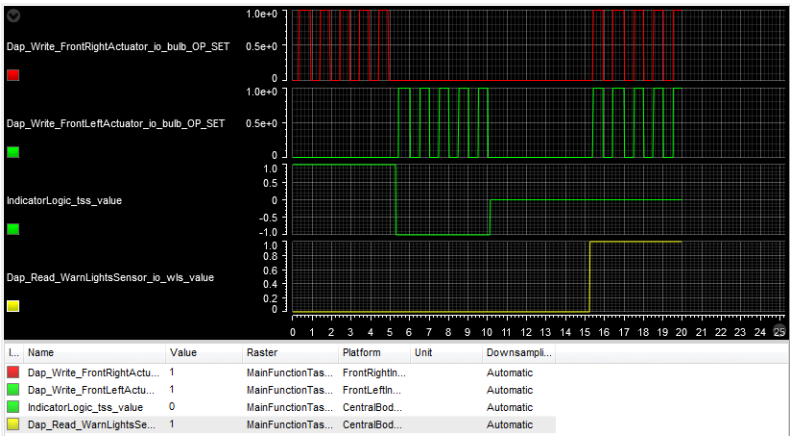


7. Use the radio buttons below the plotter to stimulate the turn switch sensor and the warn lights sensor. You can observe the effect on the IndicatorLogic_tss_value and the Dap_Read_WarnLightsSensor_io_wls_value plots, respectively.

   The illustration below shows the plotter for the following stimulation sequence:

   1. Right turn switch sensor stimulated for five seconds.
   2. Left turn switch sensor stimulated for five seconds.

3. No sensor stimulated for five seconds.
4. Warn lights sensor stimulated for five seconds.



Further examples

There are further examples of using SystemDesk. For detailed information, refer to Basics on Demos for SystemDesk (🕮 SystemDesk Manual).

Summary

Your Working Results

Finished work

You have successfully finished your work. The SystemDesk project you created covers typical AUTOSAR elements and some advanced features. You are now able to derive the concepts of SystemDesk from the various lessons and apply them in your daily work.

Glossary

Introduction Briefly explains the most important expressions and naming conventions used in the SystemDesk documentation.

A

Adaptive V-ECU A V-ECU that allows for the *dynamic* integration of new applications and changes to the network communication at run time, in contrast to a *statically* configured classic V-ECU. Communication is service-oriented via Ethernet.

An example are V-ECUs developed according to the AUTOSAR Adaptive Platform.

Application data type A data type that supports the application view of developing ECU software.

Application software The functional part of the ECU software according to the AUTOSAR Classic Platform. It is the layer above the run-time environment and the basic software.

The application software consists of software components that implement the controller code, for example.

Application software component An atomic software component that is hardware independent.

AR element An AUTOSAR element that can be placed in a package element. AUTOSAR packages themselves are also AR elements, which means they can be placed in other packages to build a package hierarchy.

AR elements are also referred to as packageable elements.

Assembly connector An element that connects the ports of two SWCs in a composition SWC.

Atomic software component Any software component that can contain an internal behavior.

AUTOSAR has defined the following types of atomic SWCs:

- Application SWC
- Complex device driver SWC
- ECU abstraction SWC
- NV block SWC
- Sensor actuator SWC
- Service proxy SWC
- Service component SWC

AUTOSAR Abbreviation of *AUT*omotive *O*pen *S*ystem *AR*chitecture.

The AUTOSAR partnership is an alliance in which the majority of OEMs, suppliers, tool providers and semiconductor companies work together to develop and establish a de-facto open industry-standard for automotive electric/electronics (E/E) architecture and to manage the growing E/E complexity.

AUTOSAR Adaptive Platform A standardized ECU software architecture for high-performance ECUs that handle complex tasks, such as tasks for autonomous driving.

AUTOSAR Classic Platform A standardized ECU software architecture that can be separated in the following parts:

- Application software
- Run-time environment
- Basic software

AUTOSAR element An element according to the AUTOSAR XML schema.

AUTOSAR File Explorer A SystemDesk control bar that lets you configure import and export of AUTOSAR files for data exchange.

AUTOSAR model The definition of a model according to AUTOSAR. An AUTOSAR model can be described with AUTOSAR XML files that you can use to exchange the AUTOSAR elements of an AUTOSAR model. Each file represents a partial AUTOSAR model and can be validated against the AUTOSAR XML schema.

AUTOSAR path A string that can be used to reference elements in an AUTOSAR model. The AUTOSAR path of an element consists of the sequence of element short names in the hierarchy of an AUTOSAR model.

B

Basic software The generic term for the following software that is part of the ECU software according to the AUTOSAR Classic Platform:

- Operating system (OS)
- AUTOSAR Services
- Communication stack
- I/O stack (ECU abstraction and microcontroller abstraction)
- Memory stack
- Complex device driver

Together with the run-time environment, the basic software is the platform for the ECU application software.

Basic software component An atomic software component that has ports and interfaces for access to the C-API of a basic software module.

There are three kinds of basic software components:

- Service components
- Complex device driver components
- ECU abstraction components

Basic software module A basic software functionality present on an ECU and accessible via a C-API. A basic software module has a module configuration with a set of configuration parameters that specify the module for code generation.

C

Classic V-ECU A V-ECU whose resources, such as the operating system and network communication, are statically configured. Therefore, in contrast to an adaptive V-ECU, you cannot add applications or make changes to the network communication at run time. Communication is mainly signal-based, as in network communication via CAN or FlexRay, but later classic V-ECUs also support service-oriented communication via Ethernet and SOME/IP.

An example are V-ECUs developed according to the AUTOSAR Classic Platform.

Client server interface An interface that describes the operations that are provided or required by a software component via a port.

Code-based V-ECU A V-ECU that is based on external AUTOSAR or non-AUTOSAR code files.

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Complex device driver software component An atomic software component that is used to access peripheral hardware, the ECU abstraction layer, and the microcontroller abstraction layer.

Component diagram A graphical editor for specifying the ports and interfaces of one software component.

Composition diagram A graphical view for specifying a composition SWC, its inner SWCs, and the connections between them. If there are several composition diagrams specified for a composition SWC, there is always one default composition diagram.

Composition software component A structuring element that consists of software components and their interconnections via ports. A composition and its connections can be specified graphically in a composition diagram.

Container File Explorer A SystemDesk control bar that lets you configure SWC containers for data exchange with TargetLink.

Container Manager A tool for exchanging SWC SWC containers between SystemDesk and TargetLink.

D

Data type An element that classifies a particular type of information, a set of data having predefined characteristics. The characteristics of a data type define the values that a variable can contain.

AUTOSAR has defined two different levels of data types for developing ECU software. Application data types support the application view of developing ECU software while implementation data types let you specify implementation details such as SW base types or endianness.

Data types can be used to specify the structure and semantic of the following data:

- Variable data prototypes of sender receiver interfaces
- Argument data prototypes in client server interfaces
- Interrunnable variables

Delegation connector An element that connects the port of a composition SWC to a port of an inner SWC.

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

E

ECU configuration The entire configuration of the basic software and the run-time environment of a single ECU.

ECU configuration framework A set of SystemDesk features that you can use to configure ECUs with basic software of any vendor.

ECU Configuration Manager A SystemDesk control bar to configure the run-time environment and the basic software of an ECU.

ECU flat view A composition SWC that is created by SystemDesk when you create an ECU configuration.

The ECU flat view contains prototypes of all the atomic SWCs that are mapped to an ECU instance and additionally SWCs for the basic software and the RTE. The ECU flat view is flat in the sense that the hierarchy that is introduced by composition SWCs is resolved by removing the compositions and connecting the atomic SWCs directly.

Element A functional part in SystemDesk.

Exclusive area A modeling element for specifying critical sections within the code that cannot preempt/interrupt each other. An exclusive area can be used to specify the mutual exclusion of runnable entities.

I

Identifiable element An AUTOSAR element that can be identified by its AUTOSAR path and short name, i.e., all identifiable AUTOSAR elements have a short name. In a valid AUTOSAR model, all identifiable elements are unique.

Implementation data type A data type to specify implementation details such as SW base types or endianness.

Interface An element that specifies the interaction between ports of software components.

AUTOSAR specifies the following interface types:

- Client server interface
- Sender receiver interface
- Mode switch interface
- Parameter interface
- NV data interface
- Trigger interface

Interrunnable variable (IRV) Modeling element for specifying communication between the runnable entities in one atomic software component.

L

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

M

Master File An AUTOSAR file that holds the AUTOSAR elements of a SystemDesk project. Reading/writing a master file is like exchanging a project's elements that are assigned to the master file with the file contents. This is most useful for version control or standardization of elements.

Master File Explorer A SystemDesk control bar that lets you assign the AUTOSAR elements of a SystemDesk project to AUTOSAR master files and read/write the master files.

Message Browser A SystemDesk control bar that provides a history of all the error and warning messages that occur during work with SystemDesk. The Message Browser provides quick access to the files, folders and elements related to a message.

Mode switch interface An interface that contains a mode declaration group. Via a mode switch interface, a software component can get information on a mode switch.

Model-based V-ECU A V-ECU that is based on an ECU in an AUTOSAR system that is available in SystemDesk.

Modeling module A SystemDesk software module that is intended for users who model ECU software.

N

NV block software component An atomic SWC that controls read an write access of all the atomic SWCs of the ECU application software to the NVRAM.

NV data interface An interface that declares a number of nonvolatile data items to be exchanged between NV block SWCs and other atomic software components.

NVRAM manager A basic software module that provides services to ensure data storage and maintenance of an ECU's nonvolatile data. The module manages the nonvolatile data of an EEPROM and/or a FLASH EPROM emulation device.

O

OS task An element that provides the framework for executing the ECU application. An OS task can be executed concurrently with other tasks. The task scheduler of the operating system controls the execution of an OS task according to the task priority and the task scheduling policy.

There are basic and extended OS tasks:

- A basic task has a defined beginning and a defined end. A basic task releases the processor only if it is being terminated, or if the operating system is executing a task with higher priority, or if an interrupt occurs. A basic task can enter only the task states *suspended*, *ready* and *running*. A basic task cannot wait for OS events.
- Unlike a basic task, an extended task can also enter the *waiting* task state, in which it can wait for an OS event.

Output Window A SystemDesk control bar that can be used to display the standard outputs of custom tools that are integrated in SystemDesk.

Overview A SystemDesk control bar that provides a reduced-size view of the currently active diagram.

P

Package A structuring element for grouping elements in any hierarchy.

Parameter interface An interface that declares a number of parameters and characteristic values to be exchanged between parameter SWCs and other SWCs.

Parameter software component An SWC that provides calibration parameters for other SWCs.

Port A part of a software component that is an interaction point between that component and other software components. The interaction between ports is specified via interfaces.

Project The top-level element for working with SystemDesk, holding all the elements, configurations and information for solving a specific problem.

Project Manager A SystemDesk control bar that provides access to a SystemDesk project and all the elements belonging to it. It displays all the elements of the project hierarchically, according to the AUTOSAR package structure.

Properties control bar A SystemDesk control bar that lets you specify all the properties of a selected element.

Prototype An element that references a type and reuses its properties. Type definitions can be reused as parts of other type definitions.

R

Root software composition A special composition SWC that describes the inter-working software components of an ECU network in the context of a system.

A root software composition can be structured hierarchically: a composition SWC in the root software composition represents a hierarchy level.

RTE event A part of an SWC internal behavior. An RTE event defines the situations and conditions for starting or continuing the execution of a specific runnable entity.

RTE intervention Code that is inserted in the run-time environment for testing SWCs during a simulation run.

The code consists of RTE intervention points that are inserted as access points into the communication of SWCs and runnables. The communication can be read or modified with RTE intervention services.

Runnable entity A part of an SWC internal behavior. With regard to code execution, a runnable is the smallest unit that can be scheduled and executed. Each runnable is implemented by exactly one C function.

Run-time environment (RTE) A generated software layer of the ECU software according to the AUTOSAR Classic Platform that connects the ECU application software to the basic software. It also interconnects the different software components of the ECU application software. There is one RTE per ECU.

S

Sender receiver interface An interface that describes data elements that are provided or required by a software component via a port.

Sensor actuator software component An atomic SWC that links from the software representation of a sensor/actuator to its hardware interface provided by the basic software. It has to be mapped to the ECU it belongs to.

Service connection diagram A graphical view for connecting application software components with basic software components in an ECU flat view.

Service software component An atomic SWC that provides service functionality via standardized AUTOSAR interfaces.

A service component describes a specific software component which is part of the basic software, and which provides services of a specific ECU. Service component ports are connected to application software components via specific connectors in service connection diagrams.

Instances of this software component type are only to be created in the ECU configuration phase for the specific purpose of the service configuration.

Simulation System A composition of models, virtual ECUs, and their interconnections required for simulating the behavior of a system.

Software component The generic term for atomic SWCs, composition SWCs, and parameter SWCs. An SWC logically groups and encapsulates single functionalities. SWCs communicate with each other via ports.

Splittable mechanism AUTOSAR has introduced the splittable mechanism to support the distribution of splittable elements across AUTOSAR files.

The following terms are essential for the splittable mechanism:

- Split: Generic term for the parts that in combination define a splittable element.
- Main split: According to AUTOSAR, only one split, i.e., the main split carries the information that cannot be split, such as properties of the root node of the splittable element.
- Split key: The information that identifies a split. In most cases the short name and, if available, variant information is sufficient for this.

Standardized AUTOSAR interface An interface with predefined interface elements. It is defined by the AUTOSAR consortium.

Basic software modules such as the NVRAM manager provide standardized interfaces that allow application software components to use services of the NVRAM manager.

Standardized interface A C-API standardized according to AUTOSAR that is provided by a basic software module.

Other basic software modules or the run-time environment can use the functionality of the basic software module via the standardized API.

SWC container A bundle of software component-related files such as AUTOSAR files (ARXML), code files (H/C), and variable description files (A2L). SWC containers are intended for exchanging software components with TargetLink.

SWC implementation An element that describes how a specific SWC internal behavior is realized for a given platform (microprocessor type and compiler). An SWC implementation mainly consists of a list of source files, object

files, compiler attributes, and dependencies between the make and build processes.

SWC internal behavior An element that represents the internal structure of an atomic software component. An SWC internal behavior is realized by an SWC implementation.

An SWC internal behavior is characterized by the following subelements:

- Runnable entities
- RTE events
- Exclusive areas
- Interrunnable variables
- Per instance memories
- Static Memories
- Service needs
- Parameters

System A combination of a root software composition, a folder with ECUs, and a network topology. A system also contains mapping information, such as the mapping of software components to ECUs.

A system represents the software architecture of an ECU network according to AUTOSAR.

System Manager A SystemDesk control bar that displays a system and its child elements and offers context menu commands to create, configure, and manage them.

T

Toolbox A SystemDesk control bar that provides access to all the elements that are required for modeling software components graphically.

Type An element that defines properties. Prototypes reference types and reuse these properties.

V

Validation A check for the compliance of elements with specific criteria that are defined in validation rules. The validation rules that are applied in a validation are specified in a validation rule configuration.

Variant A model of ECU software that contains only AUTOSAR elements with bound variations.

A variant is the result of binding the AUTOSAR elements with unbound variations in a variant-rich model.

V-ECU Generation module A SystemDesk software module that is intended for users who validate and test V-ECUs.

V-ECU implementation Implementation of the platform-independent part of a virtual ECU that can be run in an offline simulation as well as in a real-time simulation.

V-ECU Manager A SystemDesk control bar for specifying V-ECU software and generating V-ECU implementation containers for virtual validation.

Virtual ECU (V-ECU) An executable element in a simulation system that represents a real ECU or parts of it in a simulation scenario. It provides functionalities comparable to those of a real ECU.

VPU Abbreviation of *virtual processing unit*.

VPU is a generic term for a part of a simulation system that can be run in an offline simulation by VEOS. An environment model that is externally added to a simulation system results in an environment VPU.

W

Working area A part of SystemDesk's user interface that provides access to SystemDesk diagrams, editors, etc.

A

adding
  client server operations 103
  SWC internal behavior 119
adding package to
  project 31
adding to a composition
  software components 44
adding to interfaces
  variable data prototypes 66
adding to software components
  ports 51
application data types
  defining 68
  mapping to implementation data types 154
assigning data types
  variable data prototypes 73
assigning to argument data prototypes
  data types 106
assigning to data types
  compu methods 106
assigning to ports
  client server interfaces 110
  interfaces 75
AUTOSAR master files
  importing 214
  using 206
AUTOSAR template
  importing 33

B

basics
  demo scripts 23
  SystemDesk 11
building
  simulation system 248

C

calibration parameters
  creating 143
client server interfaces
  assigning to ports 110
  creating 101
client server operations
  adding 103
  defining 103
closing
  project 32
COM stack
  configuring 225
Common Program Data folder 10, 261
CommonProgramDataFolder 10, 261
completing
  OS configuration 231
composition
  connecting to underlying software components 57
composition diagrams
  creating 43
  customizing 47
  opening 43
compu methods
  assigning to data types 106
  defining 68, 96
configuring
  COM stack 225
  ECU instance 179
  ECU state manager 225
connecting
  software components 54
connecting to underlying software components
  composition 57
constants
  mapping 158
creating
  calibration parameters 143
  client server interfaces 101
  composition diagrams 43
  ECU configurations 219
  ECU instance 178
  I/O hardware abstraction 223
  interrunnable variables 127
  measurements 142
  project 30
  runnable entities 120
  sender receiver interfaces 63
  software components 41
  system 191
  V-ECUs 241
customizing
  composition diagrams 47

D

data access
  specifying by runnable entities 134
data constr elements
  defining 93
data types
  assigning to argument data prototypes 106
  defining 93
defining
  application data types 68
  client server operations 103
  compu methods 68, 96
  data constr elements 93
  data types 93
  physical dimensions 96
  units 96
demo scripts
  basics 23
  extracting 24
  running 26
Documents folder 10, 262
DocumentsFolder 10, 262

E

ECU configurations
  creating 219
ECU instance
  configuring 179
  creating 178
ECU state manager
  configuring 225
export
  V-ECU implementation container 246
exporting
  SWC containers 168
  system extracts 212
extracting
  demo scripts 24

F

first task
  SystemDesk 20

G

generating
  RTE 234

I

I/O hardware abstraction
  creating 223
importing
  AUTOSAR master files 214
  AUTOSAR template 33
  network communication elements 186
initial values
  specifying 80
integrating
  SW implementation 171
interfaces
  assigning to ports 75
interrunnable variables
  creating 127
introduction 19

L

lesson 1
  overview 29
  result 36
lesson 10
  overview 185
  result 188
lesson 11
  overview 190
  result 202
lesson 12
  overview 205
  result 215
lesson 13
  overview 218
  result 237
lesson 14
  overview 240
  result 253
lesson 2
  overview 40
  result 59

lesson 3
overview 62
result 88
lesson 4
overview 92
result 112
lesson 5
overview 116
result 138
lesson 6
overview 141
result 151
lesson 7
overview 153
result 164
lesson 8
overview 167
result 175
lesson 9
overview 177
result 182
lessons
overview 22
Local Program Data folder 10, 264
LocalProgramDataFolder 10, 264

M

mapping
constants 158
mapping to ECU instances
software components 195
mapping to implementation data types
application data types 154
mapping to network communication
SWC communication 200
mapping to OS tasks
runnables 228
measurements
creating 142

N

network communication elements
importing 186
nonqueued communication
specifying 80

O

opening
composition diagrams 43
project 32
OS configuration
completing 231
overview
lesson 1 29
lesson 10 185
lesson 11 190
lesson 12 205
lesson 13 218
lesson 14 240
lesson 2 40
lesson 3 62
lesson 4 92
lesson 5 116
lesson 6 141
lesson 7 153
lesson 8 167
lesson 9 177
lessons 22

P

physical dimensions
defining 96
ports
adding to software components 51
project
adding package to 31
closing 32
creating 30
opening 32
saving 32

Q

queued communication
specifying 80

R

result
lesson 1 36
lesson 10 188
lesson 11 202
lesson 12 215
lesson 13 237
lesson 14 253
lesson 2 59
lesson 3 88
lesson 4 112
lesson 5 138
lesson 6 151
lesson 7 164
lesson 8 175
lesson 9 182
results
working 257
RTE
generating 234
runnable entities
creating 120
triggering with RTE events 123
runnables
mapping to OS tasks 228
running
demo scripts 26

S

saving
project 32
selecting
SWC implementations 198
sender receiver interfaces
creating 63
simulation system
building 248
software components
adding to a composition 44
connecting 54
creating 41
mapping to ECU instances 195
specifying
initial values 80
nonqueued communication 80
queued communication 80
specifying by runnable entities
data access 134
splittable mechanism 206
starting
SystemDesk 14
SWC communication
mapping to network communication 200
SWC containers
exporting 168
SWC implementation
integrating 171
SWC implementations
selecting 198
SWC internal behavior
adding 119
system
creating 191
system extracts
exporting 212
SystemDesk
basics 11
first task 20
starting 14
user interface 16

T

test build
V-ECU 244
triggering with RTE events
runnable entities 123

U

units
defining 96
user interface
SystemDesk 16
user profiles 19
using
AUTOSAR master files 206

V

variable data prototypes
adding to interfaces 66
assigning data types 73
V-ECU
test build 244
V-ECU implementation container
export 246

V-ECUs
creating 241

W

working
results 257