

ConfigurationDesk

Tutorial Starting with External Devices

For ConfigurationDesk 6.7

Release 2021-A – May 2021

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

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2012 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Tutorial	5
Introduction to the Tutorial	9
Working with the Tutorial.....	9
Overview of Lessons.....	10
Accessing the CfgStartingWithExternalDevices Demo Project and Its Applications	13
Overview of the CfgStartingWithExternalDevices Demo Project.....	13
How to Open the CfgStartingWithExternalDevices Demo Project.....	15
How to Activate an Application in the CfgStartingWithExternalDevices Demo Project.....	16
Lesson 1: Specifying an External Device Interface	17
Overview of Lesson 1.....	17
Step 1: How to Create an External Device Topology.....	18
Step 2: How to Configure the External Device Interface.....	19
Step 3: How to Copy Device Topology Elements.....	21
Step 4: How to Add Device Blocks to the Signal Chain.....	22
Step 5: How to Group Device Pins.....	23
Result of Lesson 1.....	24
Lesson 2: Creating a Hardware Topology	27
Overview of Lesson 2.....	27
Step 1: How to Create a Hardware Topology from Scratch.....	28
Result of Lesson 2.....	29
Lesson 3: Implementing I/O Functionality	31
Overview of Lesson 3.....	31
Step 1: How to Add Function Blocks to the Signal Chain.....	32
Step 2: How to Configure Function Blocks.....	33
Step 3: How to Assign Hardware Resources to Function Blocks.....	35



Step 4: How to Complete the Mapping Between Device Blocks and Function Blocks.....	36
Step 5: How to Provide Wiring Information for the External Cable Harness.....	38
Result of Lesson 3.....	40
Lesson 4: Specifying the Model Interface	41
Overview of Lesson 4.....	41
Step 1: How to Add Model Port Blocks to the Signal Chain.....	42
Step 2: How to Add a Behavior Model.....	43
Result of Lesson 4.....	46
Lesson 5: Using Working Views	47
Overview of Lesson 5.....	47
Step 1: How to Generate Additional Working Views from Signal Chain Elements.....	48
Result of Lesson 5.....	49
Lesson 6: Building the Real-Time Application	51
Overview of Lesson 6.....	51
How to Prepare the Build Process and Build the Real-Time Application.....	52
Result of Lesson 6.....	54
Optional Lesson: Resolving Conflicts	55
Overview of the Resolving Conflicts Lesson.....	55
Step 1: How to Show and Resolve Conflicts.....	56
Result of Resolving Conflicts.....	60
Summary	61
Your Working Results.....	61
ConfigurationDesk Glossary	63
Index	89

About This Tutorial

Content

The dSPACE software provides demo projects for ConfigurationDesk. The **CfgStartingWithExternalDevices** demo project contains applications that build on one another. This tutorial helps you to use the demo project to learn the basic steps in ConfigurationDesk when you start out with an external device such as an ECU.

Tip





- If you want to learn how to work with ConfigurationDesk starting with an existing Simulink model, refer to [ConfigurationDesk Tutorial Starting with Simulink](#) .
- For a ConfigurationDesk tutorial using MicroAutobox III hardware, refer to [ConfigurationDesk Tutorial MicroAutoBox III](#) .





Required knowledge

This tutorial is primarily intended for engineers who implement and build real-time applications. Knowledge of Windows applications and the basics of SCALEXIO hardware is assumed.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.

Symbol	Description
	Indicates important information that you should take into account to avoid malfunctions.
	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 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/go/help.

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 the Tutorial

Where to go from here

Information in this section



Working with the Tutorial.....	9
Overview of Lessons.....	10

Working with the Tutorial


Purpose of the tutorial

This tutorial helps you learn the basic configuration steps in ConfigurationDesk when you start out with an external device such as an ECU.

Tip

- If you want to learn how to work with ConfigurationDesk starting with an existing Simulink model, refer to [ConfigurationDesk Tutorial Starting with Simulink](#) .
- For a ConfigurationDesk tutorial using MicroAutobox III hardware, refer to [ConfigurationDesk Tutorial MicroAutoBox III](#) .

Recommended knowledge

Knowledge of the basic concepts of ConfigurationDesk will help you understand the context of the different lessons. For more information, refer to [Basic Concepts of ConfigurationDesk](#) ([ConfigurationDesk Getting Started](#) .


Ways to use the demo project

You can work with the demo project in two ways:

- You can start with the first lesson (application Lesson_1) and then follow the instructions given in the following chapters without activating the other applications.

- You can activate a specific application to complete only a specific lesson, for example, Lesson_3 to learn how to implement I/O functionality.


Tip

- Every application can be used as a starting point, so you can start at different lessons or just do one lesson at a time.
- If you want to learn how to create a project and a ConfigurationDesk application, refer to [Managing ConfigurationDesk Projects and Applications](#) ([ConfigurationDesk Real-Time Implementation Guide](#) ).

Experimenting with ControlDesk

The dSPACE software also contains a version of the demo project for ControlDesk. The ControlDesk demo project is based on this ConfigurationDesk demo project and its results, which are included in the Results application.

The files of the ControlDesk demo project are packed in the **CD_StartingWithExternalDevices.zip** archive, which resides in the **<RCP and HIL installation folder>\Demos\ConfigurationDesk\Tutorial** folder after installation of the dSPACE software. The files archived with the **Variable Descriptions** relative path are the build results of the ConfigurationDesk tutorial demo project.

To start working with ControlDesk, refer to [Experimenting with a SCALEXIO System](#) ([SCALEXIO – Hardware and Software Overview](#) ).

Conflict handling

While you complete the lessons of the tutorial, several conflicts will be displayed in and removed from the **Conflicts Viewer** depending on the current configuration steps. You can ignore these conflicts while you complete the lessons, because ConfigurationDesk allows for a flexible configuration without strict constraints to let you work more freely.

Optional lesson on resolving conflicts The tutorial contains an optional lesson on resolving conflicts, for which you can use the **ResolvingConflicts** application of the demo project. This application is a modified version of the Results application and must be activated separately to start the lesson on resolving conflicts.

Overview of Lessons

Lessons and applications

The following table shows the contents of the demo applications and the associated chapters of the tutorial.

Content	Demo Application	Lesson
<ul style="list-style-type: none"> ▪ How to create an external device interface ▪ How to configure the external device interface 	Lesson_1	Lesson 1: Specifying an External Device Interface on page 17

Content	Demo Application	Lesson
<ul style="list-style-type: none"> How to copy device topology elements How to add device blocks to the signal chain How to group device pins in device connectors 		
<ul style="list-style-type: none"> How to create a hardware topology 	Lesson_2	Lesson 2: Creating a Hardware Topology on page 27
<ul style="list-style-type: none"> How to add I/O functionality to the signal chain How to configure function blocks How to assign hardware resources How to map device blocks to function blocks How to provide wiring information for the external cable harness 	Lesson_3	Lesson 3: Implementing I/O Functionality on page 31
<ul style="list-style-type: none"> How to add model port blocks to the signal chain How to add a behavior model 	Lesson_4	Lesson 4: Specifying the Model Interface on page 41
<ul style="list-style-type: none"> How to generate additional working views from signal chain elements 	Lesson_5	Lesson 5: Using Working Views on page 47
<ul style="list-style-type: none"> How to prepare the build process and build a real-time application 	Lesson_6	Lesson 6: Building the Real-Time Application on page 51
<ul style="list-style-type: none"> How to show and resolve conflicts 	ResolvingConflicts	Optional Lesson: Resolving Conflicts on page 55

Tip

The workflow steps in the demo project show you one method of obtaining the result. In many cases, other methods are also possible.

Accessing the CfgStartingWithExternalDevices Demo Project and Its Applications

Where to go from here

Information in this section

Overview of the CfgStartingWithExternalDevices Demo Project.....	13
How to Open the CfgStartingWithExternalDevices Demo Project.....	15
How to Activate an Application in the CfgStartingWithExternalDevices Demo Project.....	16

Overview of the CfgStartingWithExternalDevices Demo Project

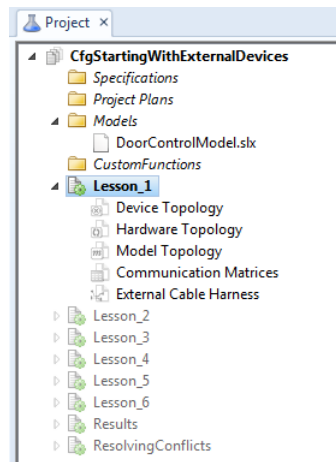
Introduction

The contents of the `CfgStartingWithExternalDevices` demo project demonstrate the typical workflow for implementing real-time applications when you start out with an ECU. The project starts with creating an external device interface and ends with building a real-time application.

Components of the project

The demo project contains one `ConfigurationDesk` application for each of the tutorial's lessons. Each lesson contains only very few work steps, so you can easily switch between the applications to complete different lessons. The `Models` folder contains a prepared behavior model called `DoorControlModel.slx`, which can be used in MATLAB/Simulink.

The applications build on one another. For example, all the data of the first application (`Lesson_1`) is included in the second one (`Lesson_2`). The second application also contains the results of the worksteps described for the first application.



Use scenario

The demo project shows a simplified but realistic example of testing ECUs in a hardware-in-the-loop (HIL) simulation. It consists of two ECUs for controlling door functionality, such as the door light, the motor for mirror adjustment and the door's lock/unlock mechanism. The ECUs provide the following signals at their pins:

ECU	PIN	Signal	Type
Door left	A1	Door Light output	Switch
	A2	Door Light GND	Reference
	A3	Mirror Heating output	Switch
	A4	Mirror Motor output	PWM
	A5	Mirror GND	Reference
	A6	Lock input	Switch
	A7	Unlock input	Switch
	A8	GND	Reference
Door right	B1	Door Light output	Switch
	B2	Door Light GND	Reference
	B3	Mirror Heating output	Switch
	B4	Mirror Motor output	PWM
	B5	Mirror GND	Reference
	B6	Lock acknowledge input	Switch
	B7	Unlock acknowledge input	Switch
	B8	GND	Reference

Location of files

The files of the demo project are available in the *Documents* folder.

They are also backed up in the `CfgStartingWithExternalDevices.zip` archive, which resides in the <RCP and HIL installation folder>\Demos\ConfigurationDesk\Tutorial folder after installation of the dSPACE software.

How to Open the CfgStartingWithExternalDevices Demo Project

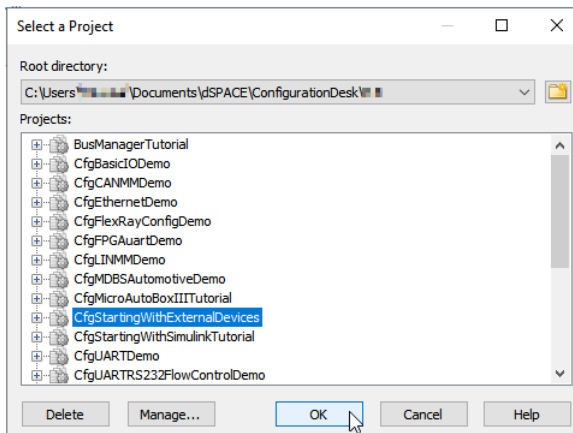
Objective

The `CfgStartingWithExternalDevices` demo project is available in the *Documents* folder, so you can open it like a normal project.

Method

To open the tutorial demo project

- 1 From the Start menu, select dSPACE RCP and HIL <dSPACE Release> – dSPACE ConfigurationDesk <x.y>.
- 2 On the Start Page, click Open Project + Application.
ConfigurationDesk opens the Select a Project dialog.



- 3 Select the `CfgStartingWithExternalDevices` project and click OK.

Result

You opened the tutorial demo project. The Lesson_1 application is active.

Related topics

Basics

[Starting ConfigurationDesk \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(274fd520e03b61c1b9ffc861754cacdc_img.jpg\)](#))

How to Activate an Application in the CfgStartingWithExternalDevices Demo Project

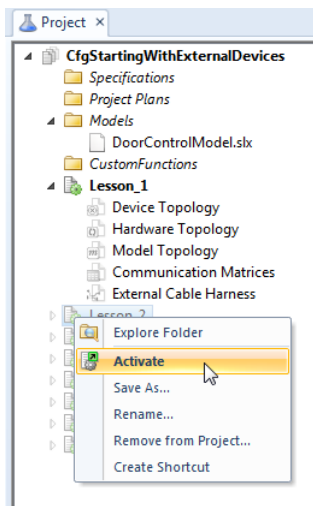
Objective To work with a specific tutorial demo project application, you must first activate it. The active application is displayed in bold letters.

Precondition

- The demo project is open.
- The Project Manager is displayed.

Method **To activate an application in the tutorial demo project**

- 1 In the Project Manager, right-click the inactive application you want to activate.
- 2 From the context menu, select **Activate** as shown below.



Result ConfigurationDesk activates the selected application and deactivates the application that was active before. You can now use the active application to complete further steps of the tutorial.

Related topics

HowTos

[How to Open the CfgStartingWithExternalDevices Demo Project.....](#) 15

Lesson 1: Specifying an External Device Interface

Where to go from here

Information in this section

Overview of Lesson 1	17
Step 1: How to Create an External Device Topology.....	18
Step 2: How to Configure the External Device Interface.....	19
Step 3: How to Copy Device Topology Elements.....	21
Step 4: How to Add Device Blocks to the Signal Chain.....	22
Step 5: How to Group Device Pins.....	23
Result of Lesson 1	24

Overview of Lesson 1

External devices

External devices are all the devices which are connected to the dSPACE hardware, for example, ECUs or external loads.

What will you do?

In this lesson you will:

- Create a representation of an external device in ConfigurationDesk (= external device topology).
- Configure the external device interface.
- Copy device topology elements.
- Add device blocks to the signal chain.
- Group device pins in device connectors.

Before you begin

Before you begin this lesson, the following precondition must be met:

- The Lesson_1 application is active. This is already the case if you just opened the tutorial demo project for the first time (refer to [How to Open the CfgStartingWithExternalDevices Demo Project](#) on page 15).
If you need to activate the application, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following steps:

- [Step 1: How to Create an External Device Topology](#) on page 18
- [Step 2: How to Configure the External Device Interface](#) on page 19
- [Step 3: How to Copy Device Topology Elements](#) on page 21
- [Step 4: How to Add Device Blocks to the Signal Chain](#) on page 22
- [Step 5: How to Group Device Pins](#) on page 23

Step 1: How to Create an External Device Topology

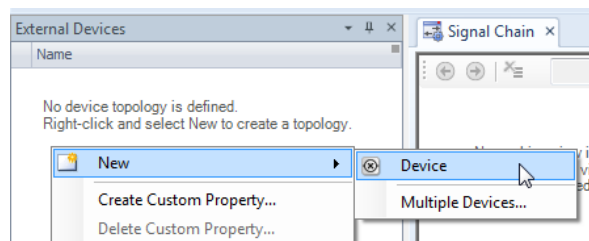
Objective

An external device topology represents one or more external device interface(s) in a ConfigurationDesk application. You can create a device topology step by step via the External Device Browser.

Method

To create an external device topology

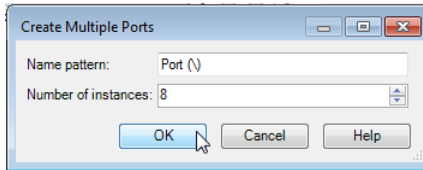
- 1 Switch to the Signal Chain view set.
- 2 In the External Device Browser, right-click in a free area.
- 3 From the context menu, select New - Device.



A new device is added to the device topology.

- 4 ConfigurationDesk lets you change the name of the new device right after creating it.
Change the device name to LeftDoorControl.
- 5 Right-click the LeftDoorControl device and select New – Multiple Ports.

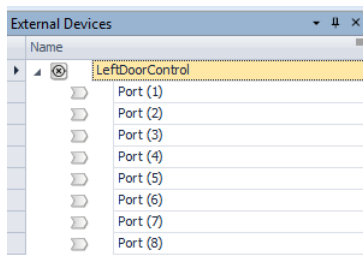
A Create Multiple Ports dialog opens.



- 6 In the Create Multiple Ports dialog, set the Number of instances to 8 for eight new device ports.
- 7 Click OK.

Result

You have created an external device topology. It contains one device with 8 device ports.



Step 2: How to Configure the External Device Interface

Objective

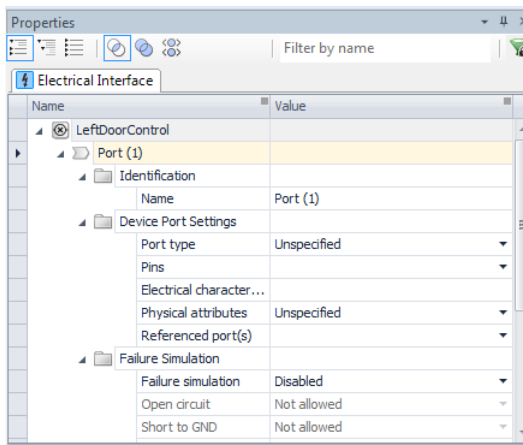
To specify characteristics of the external device interface, you have to configure the device topology elements.

Method

To configure the external device interface

- 1 In the External Device Browser, select the Port (1) device port.

The properties of the selected device port are displayed in the Properties Browser.



2 Change the property settings according to the following table:

Property	Value
Name	DoorLight
Port type	Out
Pin(s)	A1
Physical attributes	Voltage
Failure simulation	Enabled
Open circuit ¹⁾	Allowed
Short to GND ¹⁾	Allowed
Short to VBAT ¹⁾	Allowed
Short to signal generation channel ¹⁾	Not allowed
Short to signal measurement channel ¹⁾	Not allowed
Short to bus channel ¹⁾	Not allowed

¹⁾ Configurable only if failure simulation is enabled.

3 Repeat steps 1 to 2 to configure the properties of the other device ports according to the settings in the following table:

	Port (2)	Port (3)	Port (4)	Port (5)	Port (6)	Port (7)	Port (8)
Name	LightGND	MirrorHeating	MirrorMotor	MirrorGND	Lock	Unlock	GND
Port type	Reference	Out	Out	Reference	In	In	Reference
Pin(s)	A2	A3	A4	A5	A6	A7	A8
Physical attributes	Unspecified	Unspecified	Unspecified	Unspecified	Unspecified	Unspecified	Unspecified
Failure simulation	Disabled	Enabled	Enabled	Disabled	Enabled	Enabled	Disabled
Open circuit ¹⁾	–	Allowed	Allowed	–	Allowed	Allowed	–
Short to GND ¹⁾	–	Allowed	Allowed	–	Allowed	Allowed	–
Short to VBAT ¹⁾	–	Allowed	Allowed	–	Allowed	Allowed	–

	Port (2)	Port (3)	Port (4)	Port (5)	Port (6)	Port (7)	Port (8)
Short to signal generation channel ¹⁾	–	Not allowed	Not allowed	–	Not allowed	Not allowed	–
Short to signal measurement channel ¹⁾	–	Not allowed	Not allowed	–	Not allowed	Not allowed	–
Short to bus channel ¹⁾	–	Not allowed	Not allowed	–	Not allowed	Not allowed	–

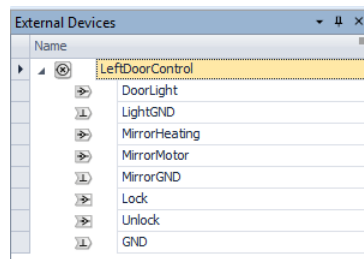
¹⁾ Configurable only if failure simulation is enabled.

- 4** Now that the device ports can be identified by their names, configure the Referenced port(s) property with the following settings:

	DoorLight	LightGND	MirrorHeating	MirrorMotor	MirrorGND	Lock	Unlock	GND
Referenced port(s)	LightGND	DoorLight	MirrorGND	MirrorGND	MirrorMotor MirrorHeating	GND	GND	Lock Unlock

Result

You configured the external device interface.



Step 3: How to Copy Device Topology Elements

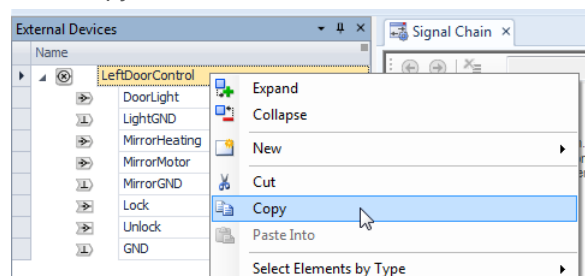
Objective

An easy way of creating new external device topology elements is to copy existing ones.

Method

To copy device topology elements

- 1** In the External Device Browser, right-click the LeftDoorControl device and select Copy.



- 2** Right-click a free area in the External Device Browser and select Paste Into.

You created a new device in the device topology by copying. The configuration of the device and its ports was also copied, except for the Referenced Port(s) property.

- 3 Change the name of the LeftDoorControl Copy device to RightDoorControl.
- 4 Set the Pin(s) property to B1, B2, etc.

Note

You can use the same pin name in different devices, but each pin name must be unique within a device.

- 5 Configure the Referenced port(s) property in the same way as for the LeftDoorControl device:

	DoorLight	LightGND	MirrorHeating	MirrorMotor	MirrorGND	Lock	Unlock	GND
Referenced port(s)	LightGND	DoorLight	MirrorGND	MirrorGND	MirrorMotor MirrorHeating	GND	GND	Lock Unlock

Result

You copied the LeftDoorControl device and changed the device name, pin names, and referenced ports according to the demo. Now there are two devices in the device topology.

Step 4: How to Add Device Blocks to the Signal Chain

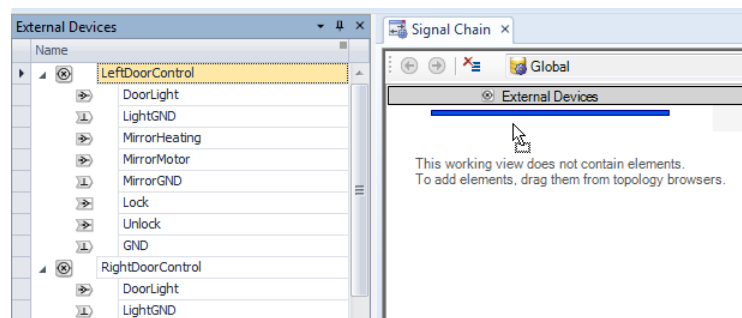
Objective

To use a device in the signal chain of your ConfigurationDesk application, you have to add a device block to the signal chain.

Method

To add device blocks to the signal chain


- 1 In the External Device Browser, click the LeftDoorControl device and drag it into the Global working view.



- 2 Repeat step 2 with the RightDoorControl device.

Result

You added the LeftDoorControl and RightDoorControl device blocks to the signal chain.

In the External Device Browser, the  symbol appears next to the added device topology elements to show that they are used in the signal chain.

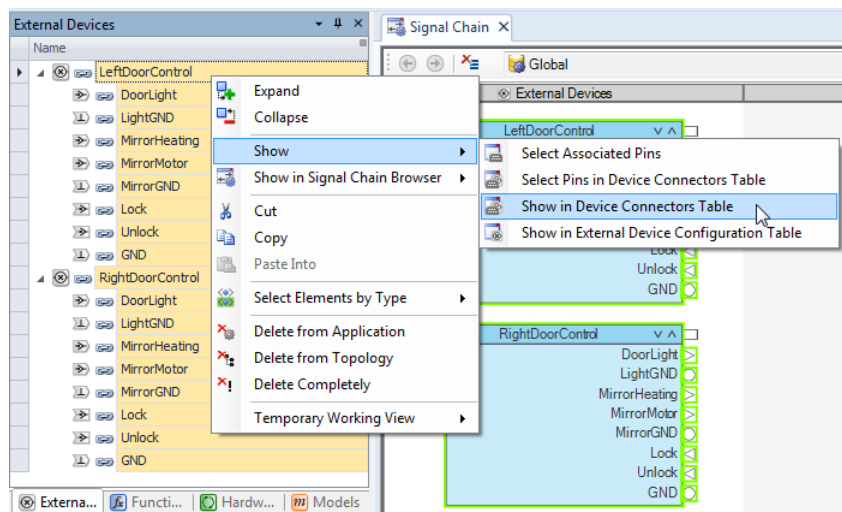
Step 5: How to Group Device Pins

Objective

You can use device connector and device pin elements in the External Device Connectors table to group pins within connectors representing the structure of the real ECU connector. This gives you a better overview of device pins, for example, in the wiring information.

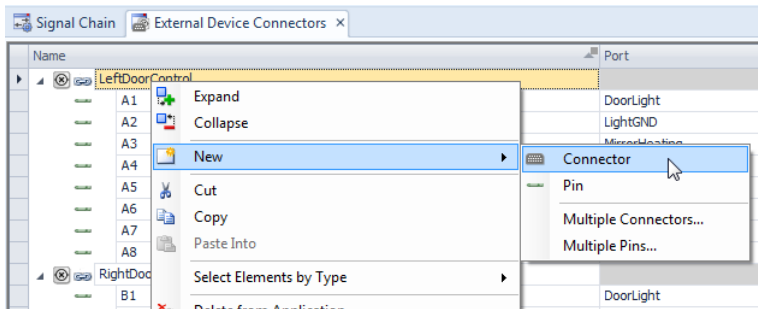
Method**To group device pins**

- 1 In the External Device Browser, press **Ctrl + A** to select all elements of the device topology.
- 2 Right-click the selection and select Show – Show in Device Connectors Table.



- 3 In the External Device Connectors table, select the A1 pin in the RightDoorControl device, then press **Ctrl** and select the pins A2 to A8. These pins are unassigned remnants of the copy process in [Step 3: How to Copy Device Topology Elements](#) on page 21.
- 4 Right-click the selection and select Delete from Topology from the context menu.

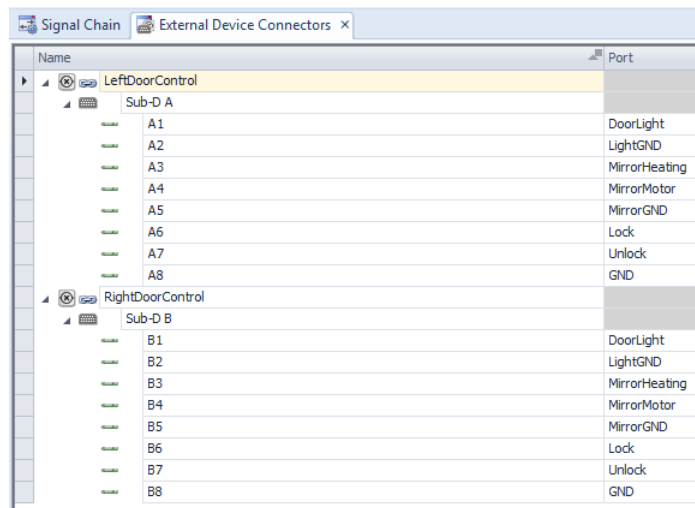
- 5 Right-click the LeftDoorControl device and select New – Connector.



- 6 Change the name of the Connector (1) to Sub-D A.
- 7 Select pins A1 to A8 while pressing the **Ctrl** key and drag the selection onto the Sub-D A connector.
- 8 Repeat steps 5 to 7 for the RightDoorControl device, this time grouping pins B1 to B8 in a connector named Sub-D B.

Result

You created device connectors for the demo devices and grouped device pins in them.



Result of Lesson 1

Result

You created a new device topology from scratch, configured the created devices and added them to the signal chain. You also grouped the device pins in device connectors.

Now your application is equivalent to the Lesson_2 demo application.

Further information

For detailed descriptions and instructions regarding the external device interface, refer to [Specifying the External Device Interface \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(919a2cb85b99741a73c0c31a427236a8_img.jpg\)](#)).

What's next

Now you can continue with [Lesson 2: Creating a Hardware Topology](#) on page 27 or activate a different application to continue with the corresponding lesson. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Lesson 2: Creating a Hardware Topology

Where to go from here

Information in this section

Overview of Lesson 2.....	27
Step 1: How to Create a Hardware Topology from Scratch.....	28
Result of Lesson 2.....	29

Overview of Lesson 2

Hardware topology

The hardware topology allows you to add hardware resources to a ConfigurationDesk application and assign them to your I/O functionality even before actual real-time hardware is available.

What will you do?

- In this lesson you will:
- Create a representation of a SCALEXIO platform in ConfigurationDesk (= hardware topology).

Before you begin

- Before you begin this lesson, one of the following preconditions must be met:
- You completed all the lessons of the tutorial up to this point.
- or
- You activated the Lesson_2 application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following steps:

- [Step 1: How to Create a Hardware Topology from Scratch](#) on page 28

Step 1: How to Create a Hardware Topology from Scratch

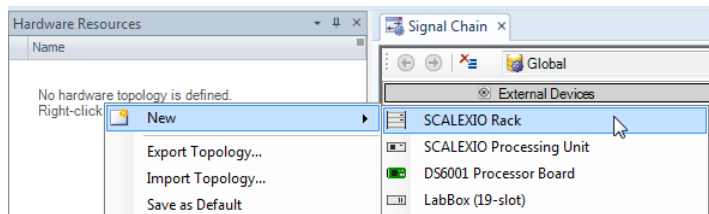
Objective

You can create hardware topologies from scratch according to your needs, even without having real hardware connected to ConfigurationDesk.

Method

To create a hardware topology

- 1 Switch to the Signal Chain view set if necessary.
- 2 In the Hardware Resource Browser, right-click in a free area.
- 3 From the context menu, select New – SCALEXIO Rack.



A new SCALEXIO rack is added to the hardware topology.

- 4 Right-click the SCALEXIO Rack (1) and select New – SCALEXIO Processing Unit.

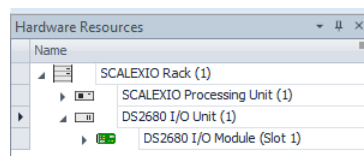
A new SCALEXIO Processing Unit is added to the hardware topology.

- 5 Right-click the SCALEXIO Rack (1) and select New – DS2680 I/O Unit.


A new DS2680 I/O Unit is added to the hardware topology.

Result

You created a hardware topology from scratch.



Result of Lesson 2

Result	<p>You created a new hardware topology from scratch.</p> <p>Now your application is equivalent to the Lesson_3 demo application.</p>
Further information	<p>For detailed descriptions and instructions regarding hardware topologies, refer to:</p> <ul style="list-style-type: none">▪ Working with Hardware Topologies (ConfigurationDesk Real-Time Implementation Guide )
What's next	<p>Now you can continue with Lesson 3: Implementing I/O Functionality on page 31 or activate a different application to continue with the corresponding lesson. For instructions, refer to How to Activate an Application in the CfgStartingWithExternalDevices Demo Project on page 16.</p>

Lesson 3: Implementing I/O Functionality

Where to go from here

Information in this section

Overview of Lesson 3.....	31
Step 1: How to Add Function Blocks to the Signal Chain.....	32
Step 2: How to Configure Function Blocks.....	33
Step 3: How to Assign Hardware Resources to Function Blocks.....	35
Step 4: How to Complete the Mapping Between Device Blocks and Function Blocks.....	36
Step 5: How to Provide Wiring Information for the External Cable Harness.....	38
Result of Lesson 3.....	40

Overview of Lesson 3

I/O functionality

Implementing and configuring I/O functionality is one of the main work steps in implementing a real-time application. Function blocks represent the I/O functionality of the real-time hardware in the signal chain.

What will you do?

- In this lesson you will:
- Add function blocks to the signal chain.
 - Configure the function blocks.
 - Assign hardware resources to the function blocks.
 - Map the device blocks to the function blocks.
 - Provide the wiring information for the external cable harness.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You activated the Lesson_3 application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following steps:

- [Step 1: How to Add Function Blocks to the Signal Chain](#) on page 32
- [Step 2: How to Configure Function Blocks](#) on page 33
- [Step 3: How to Assign Hardware Resources to Function Blocks](#) on page 35
- [Step 4: How to Complete the Mapping Between Device Blocks and Function Blocks](#) on page 36
- [Step 5: How to Provide Wiring Information for the External Cable Harness](#) on page 38

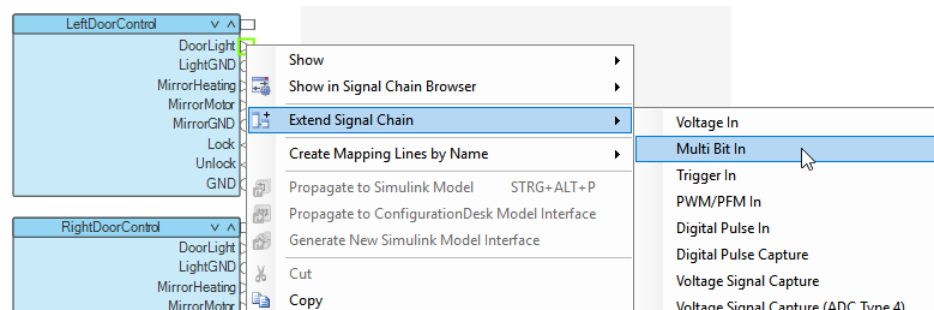
Step 1: How to Add Function Blocks to the Signal Chain

Objective

To use I/O functionality in your application, you have to add function blocks to the signal chain.

Method**To add function blocks to the signal chain**

- 1 Switch to the Signal Chain view set if necessary.
- 2 In the Global working view, right-click the DoorLight device port of the LeftDoorControl device and select **Extend Signal Chain – Multi Bit In**.

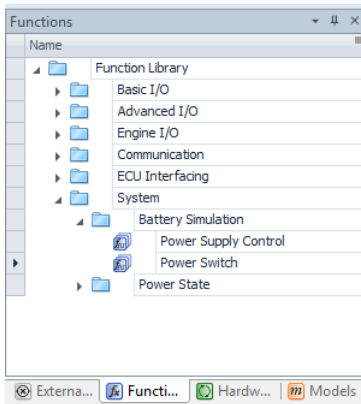


You added a function block to the signal chain.

- 3 Repeat step 2 with the following device ports and function blocks:

Device	Device Port	Function Block
LeftDoorControl	MirrorHeating	Multi Bit In
	MirrorMotor	PWM/PFM In
	Lock	Multi Bit Out
	Unlock	Multi Bit Out
RightDoorControl	DoorLight	Multi Bit In
	MirrorHeating	Multi Bit In
	MirrorMotor	PWM/PFM In
	Lock	Multi Bit Out
	Unlock	Multi Bit Out

- 4 In the Function Browser, open the System\Battery Simulation folder.



- 5 Click the Power Switch function block type and drag it to the Global working view.
- 6 On the Home ribbon, click Blocks – Order to arrange device and function blocks so that intersecting mapping lines are avoided.

Result

You added function blocks to the signal chain. ConfigurationDesk mapped the device ports and the signal ports of the function blocks. The settings of failure simulation classes (specified at the device ports) were transferred to the function blocks.

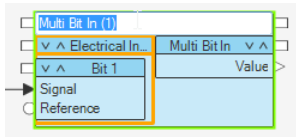
Step 2: How to Configure Function Blocks

Objective

To change function block names and properties to comply with your project definitions.

Method**To configure function blocks**

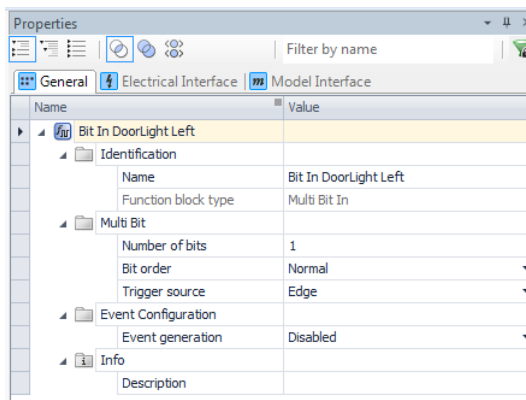
- 1 In the Global working view, double-click the Multi Bit In (1) function block.



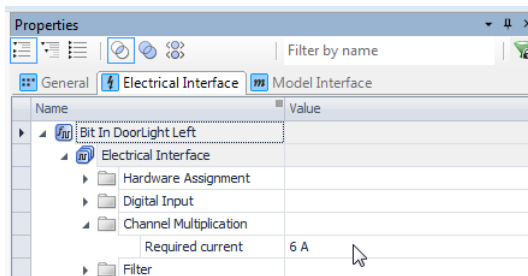
- 2 Change the function block name from Multi Bit In (1) to Bit In DoorLight Left.
- 3 Repeat steps 1 to 2 and rename the following function blocks:

Current Function Block Name	New Function Block Name
Multi Bit In (2)	Bit In MirrorHeating Left
PWM/PFM In (1)	PWM In MirrorMotor Left
Multi Bit Out (1)	Bit Out Lock Left
Multi Bit Out (2)	Bit Out Unlock Left
Multi Bit In (3)	Bit In DoorLight Right
Multi Bit In (4)	Bit In MirrorHeating Right
PWM/PFM In (2)	PWM In MirrorMotor Right
Multi Bit Out (3)	Bit Out Lock Right
Multi Bit Out (4)	Bit Out Unlock Right

- 4 The demo scenario requires you to change the settings of the Required current property.
In the Global working view, select the Bit In DoorLight Left function block.
The properties of the function block are displayed in the Properties Browser.



- 5 In the Properties Browser, switch to the Electrical Interface page and expand the Channel Multiplication category.



- 6 Change the Required current property to 10 A. You can do so by just entering the respective number, ConfigurationDesk automatically adds the correct unit.
- 7 Repeat steps 4 to 6 to configure the Required current property according to the following table:

Function Block	Required Current
Bit In MirrorHeating Left	10 A
PWM In MirrorMotor Left	10 A
Bit Out Lock Left	0.003 A
Bit Out Unlock Left	0.003 A
Bit In DoorLight Right	10 A
Bit In MirrorHeating Right	10 A
PWM In MirrorMotor Right	10 A
Bit Out Lock Right	0.003 A
Bit Out Unlock Right	0.003 A

Result

The function blocks have been configured.

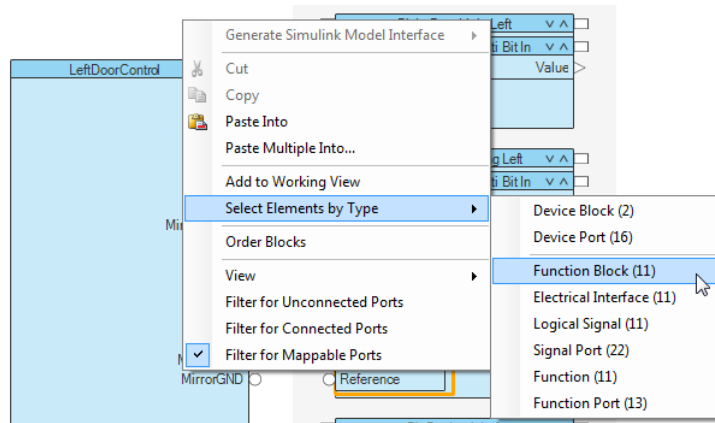
Step 3: How to Assign Hardware Resources to Function Blocks

Objective

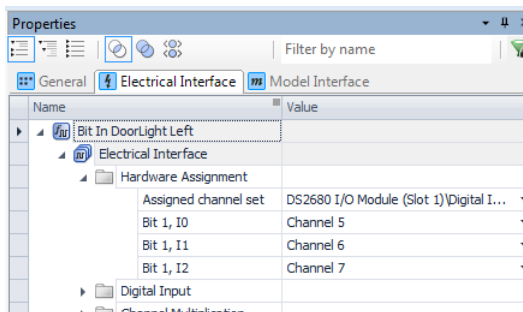
To assign hardware resources to function blocks, you can use the automatic hardware resource assignment provided by ConfigurationDesk.

Method**To assign hardware resources to function blocks**

- 1 In the Global working view, right-click in a free area.
- 2 From the context menu, select **Select Elements by Type – Function Block**.



- 3 On the Home ribbon, click **Hardware – Assign**.
The channel sets and channels of the function blocks are assigned to the first suitable and available channel sets and channels of the hardware topology.
- 4 To show the hardware resource assignment, select a function block in the Global working view, switch to the **Electrical Interface** page in the Properties Browser and expand the **Hardware Assignment** category:

**Result**

You have assigned channel sets and channels to the function blocks.

Step 4: How to Complete the Mapping Between Device Blocks and Function Blocks

Objective

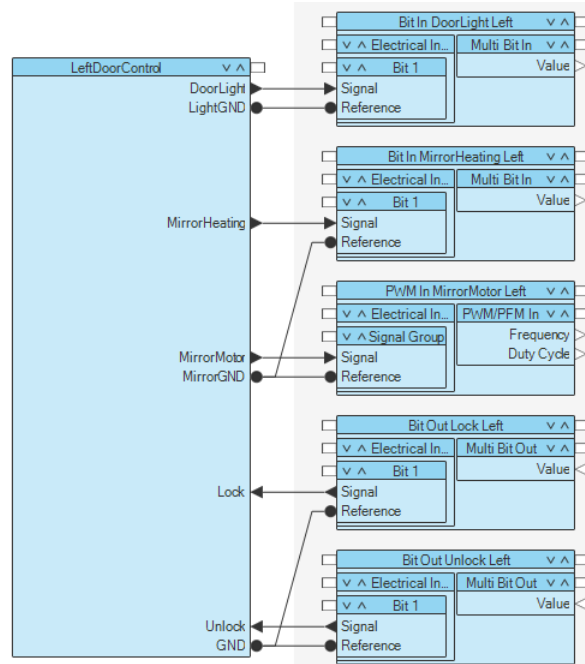
To complete the mapping between device and function blocks done by ConfigurationDesk, you have to map the reference ports manually.

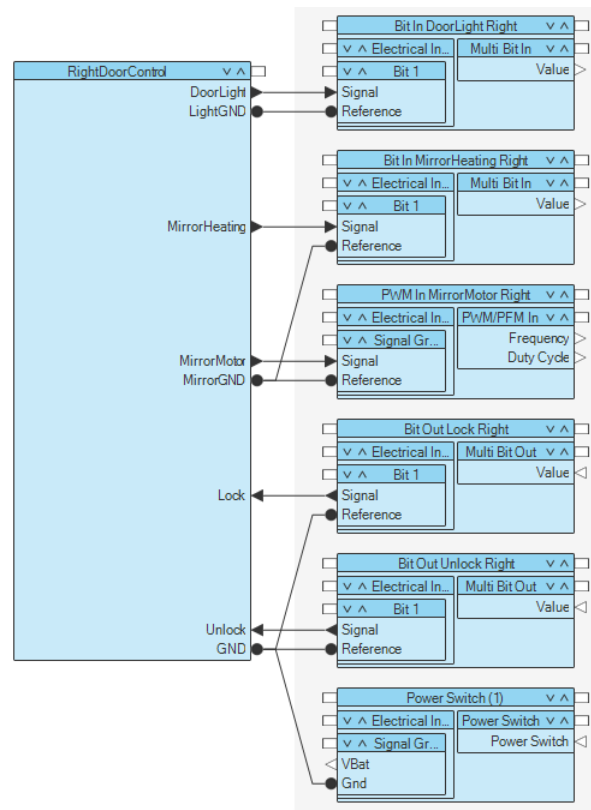
Mapping ports means drawing a mapping line between two ports, for example, between a device port and a signal port, to connect them in the signal chain.

Method

To complete the mapping between device blocks and function blocks

- 1 In the Global working view, click the LightGND device port.
- 2 Draw a mapping line to the signal Reference port of the Bit In DoorLight Left function block while you press the left mouse button.
- 3 Repeat step 2 with all the other reference ports according to the illustrations below, which show you the devices LeftDoorControl and RightDoorControl with completed device port mappings.





Result You mapped all the device ports to signal ports of function blocks.

Step 5: How to Provide Wiring Information for the External Cable Harness

Objective As an aid to building an external cable harness, ConfigurationDesk can calculate the wiring information and export it to a file.

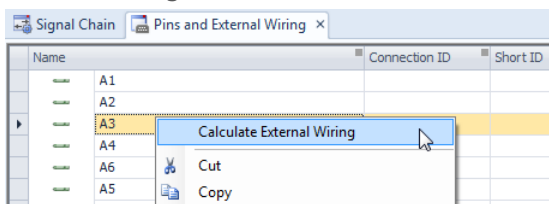
Method **To provide wiring information for the external cable harness**

- 1 On the Home ribbon, click Navigation – Windows – Pins and External Wiring.

The Pins and External Wiring table opens in the working area.

Name	Connection ID	Short ID	Connector	Signal	Port
A1			Sub-D A		DoorLight
A2			Sub-D A		LightGND
A3			Sub-D A		MirrorHeating
A4			Sub-D A		MirrorMotor
A6			Sub-D A		Lock
A5			Sub-D A		MirrorGND
A7			Sub-D A		Unlock
A8			Sub-D A		GND
B1			Sub-D B		DoorLight
B2			Sub-D B		LightGND
B3			Sub-D B		MirrorHeating
B4			Sub-D B		MirrorMotor
B5			Sub-D B		MirrorGND
B6			Sub-D B		Lock
B7			Sub-D B		Unlock
B8			Sub-D B		GND
A1			ECU1 [SCALEXIO ...]	Analog In 1 Chan...	
A2			ECU1 [SCALEXIO ...]	Analog In 1 Chan...	
A3			ECU1 [SCALEXIO ...]	Analog In 1 Chan...	

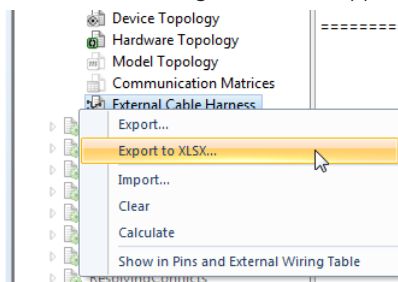
- In the Pins and External Wiring table, right-click and select Calculate External Wiring.



The external wiring information is calculated and added to your ConfigurationDesk application. The pins contained in the calculated cable harness are marked with a chain symbol.

B1		8	1
B2		2	5
B3		9	1
B4		10	1

- Switch to the Project view set.
- In the Project Manager, right-click the External Cable Harness component of the active ConfigurationDesk application and select Export to XLSX.



An Export External Cable Harness dialog opens.

- In the Export External Cable Harness dialog, choose the folder you want to export the external cable harness file to.
- Enter the file name, with or without the file name extension.
- Click Save.

Result

You calculated the wiring information for an external cable harness and exported it to an Microsoft Excel™ file (XLSX file).

For more information on the external cable harness and a description of the exported Excel™ file, see [Calculating an External Cable Harness \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)\)](#).

Result of Lesson 3

Result

You added I/O functionality to the signal chain, configured function blocks and assigned hardware resources to them, completed the mapping between device blocks and function blocks, and provided the wiring information for the external cable harness.

Now your application is equivalent to the Lesson_4 demo application.

Further information

For detailed descriptions and instructions on the topics dealt with in this lesson, refer to:

- [Implementing I/O Functionality \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(4decd7f4d36b8b21e9f05326cc7983ef_img.jpg\)\)](#)
- [Assigning Hardware Resources to Function Blocks \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(c3e0af516d5b5e8e8267fd350d6c692b_img.jpg\)\)](#)
- [Calculating an External Cable Harness \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(ad3940efc458c16a00757f90b2b0f20a_img.jpg\)\)](#)

What's next

Now you can continue with [Lesson 4: Specifying the Model Interface](#) on page 41 or activate a different application to continue with the corresponding lesson. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Lesson 4: Specifying the Model Interface

Where to go from here

Information in this section

Overview of Lesson 4.....	41
Step 1: How to Add Model Port Blocks to the Signal Chain.....	42
Step 2: How to Add a Behavior Model.....	43
Result of Lesson 4.....	46

Overview of Lesson 4

Model interface

In ConfigurationDesk, the interface to the behavior model is implemented via model port blocks and model ports.

What will you do?

- In this lesson you will:
- Add model port blocks to the signal chain.
 - Add a behavior model.

Before you begin

- Before you begin this lesson, one of the following preconditions must be met:
- You completed all the lessons of the tutorial up to this point.
- or
- You activated the Lesson_4 application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following steps:

- [Step 1: How to Add Model Port Blocks to the Signal Chain](#) on page 42
- [Step 2: How to Add a Behavior Model](#) on page 43

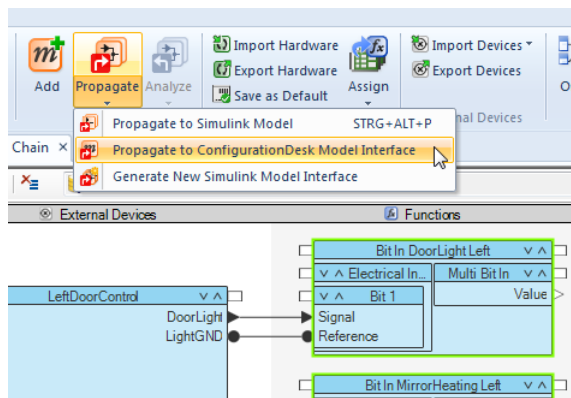
Step 1: How to Add Model Port Blocks to the Signal Chain

Objective

To use model port block data in your application, you must add model port blocks to the signal chain.

Method**To add model port blocks to the signal chain**

- 1 Switch to the Signal Chain view set if necessary.
- 2 In the Global working view, right-click in an empty area.
- 3 From the context menu, select **Select Elements by Type – Function Block**.
- 4 On the Home ribbon, select **Models – Propagate – Propagate to ConfigurationDesk Model Interface**.



For each function block, ConfigurationDesk adds a model port block with a suitable configuration and data direction to the signal chain. It also automatically maps the model ports to function ports.

Note

In the real workflow, you would instead use the Propagate – Generate New Simulink Model Interface command to also create model port blocks with matching IDs in a Simulink interface model. You could then use these model port blocks in Simulink to start implementing the behavior model or to add them to an existing model using the Paste and Keep IDs functionality.

This is not necessary here, because the demo provides a ready-to-use behavior model for you that you will add in the next step. For basic information on interface models and behavior models, refer to [Handling the Model Interface \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)](#)).

- 5 On the Home ribbon, click Blocks – Order to arrange function blocks and model port blocks so that intersecting mapping lines are avoided.

Result

You added model port blocks to the signal chain.

Step 2: How to Add a Behavior Model

Objective

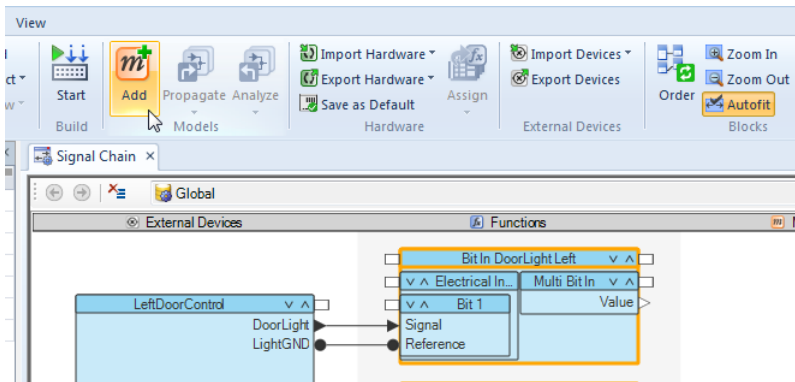
To work with model data in your application, you must add a behavior model.

Note

In the real workflow, you would first use the model port blocks from a Simulink interface model to implement the behavior model. This is not necessary here, because the demo provides a ready-to-use behavior model for you. For basic information on interface models and behavior models, refer to [Handling the Model Interface \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(4fe57c3593bf1b21d272ae7ac8dfaf77_img.jpg\)](#)).

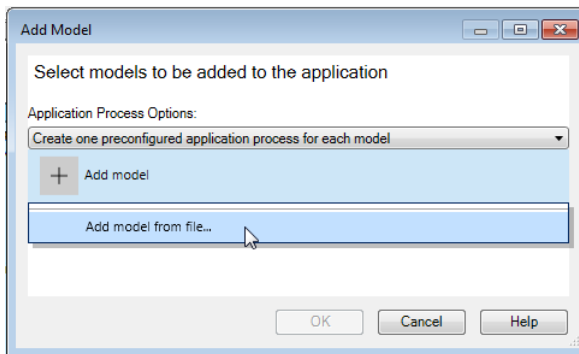
Method**To add a behavior model**

- 1 On the Home ribbon, click Models – Add.



An Add Model dialog opens.

- 2 Click Add model – Add model from file.

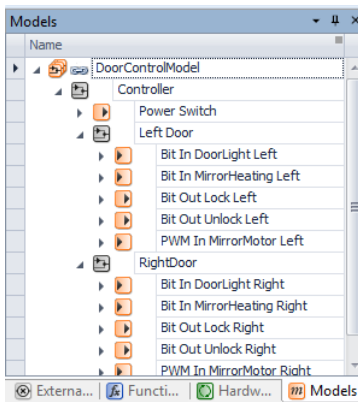


- 3 Select the following Simulink model file and click Open:

<Documents folder>\Tutorials\CfgStartingWithExternalDevices\Models\DoorControlModel1.slx

- 4 In the Add Model dialog, click OK.

ConfigurationDesk opens the model in MATLAB/Simulink, analyzes the model information, and adds it to the active ConfigurationDesk application as a model topology. The Model Browser displays the model topology in a hierarchical structure.



ConfigurationDesk also creates a new application process with the same name as the model. The model is then automatically assigned to the application process, which already includes a periodic task. You can see the application process and task assignment in the Task Configuration table in the Tasks view set.

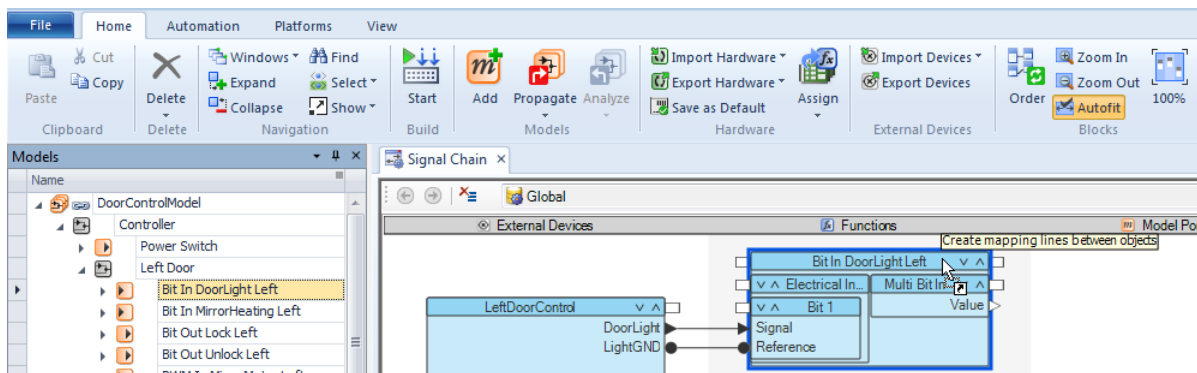
Task Configuration						
Name	Priority	DAQ Raster Name	Real-Time Testing	Period	Offset	
DoorControlModel						
Periodic Task 1	40	Periodic Task 1	<input checked="" type="checkbox"/>	0.001	0	

5 Note

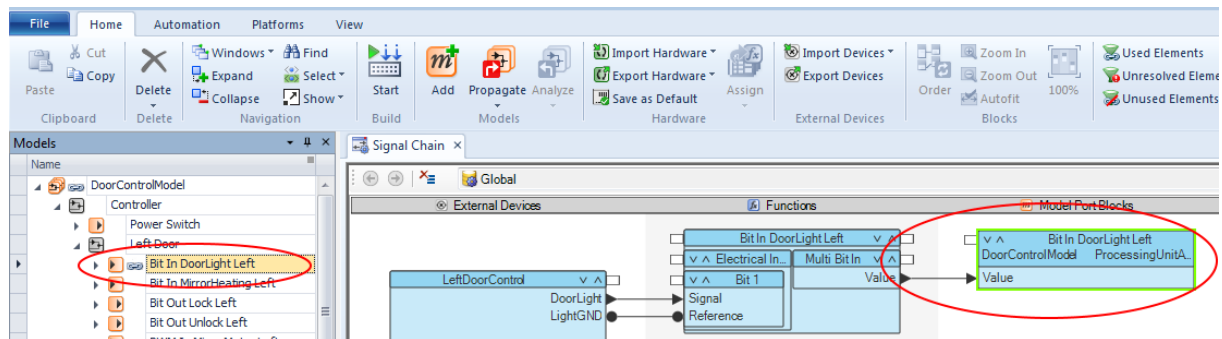
The IDs of the model port blocks you added in the previous step do not match the IDs of the model port blocks from the added model. Therefore, you have to remove the model port blocks from the signal chain and add the model port blocks with matching IDs to the signal chain from the model topology. This is not necessary in the real workflow.

In the Global working view, right-click in an empty area.

- 6 From the context menu, select **Select Elements by Type – Model Port Block**.
- 7 Right-click the selection and select **Delete from Application** from the context menu.
- 8 Drag the **Bit In DoorLight Left** model port block from the model topology to the **Bit In DoorLight Left** function block in the Global working view.



ConfigurationDesk adds the model port block to the signal chain and maps it to the function block. In the model topology, the chain symbol next to the model port block indicates that it is used in the active ConfigurationDesk application.



- 9 Repeat step 8 to add all model port blocks from the model topology to the signal chain and map them to the function blocks of the same name.

Result

You added a behavior model.

Result of Lesson 4

Result

You added model port blocks to the signal chain and added a behavior model.
Now your application is equivalent to the Lesson_5 demo application.

Further information

For detailed descriptions and instructions on the model interface, refer to [Specifying the Model Interface \(ConfigurationDesk Real-Time Implementation Guide\)](#).

What's next

Now you can continue with [Lesson 5: Using Working Views](#) on page 47 or activate a different application to continue with the corresponding lesson. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Lesson 5: Using Working Views

Where to go from here

Information in this section

Overview of Lesson 5.....	47
Step 1: How to Generate Additional Working Views from Signal Chain Elements.....	48
Result of Lesson 5.....	49

Overview of Lesson 5

Working views

A working view is a specific view of the signal chain. The Global working view contains all signal chain elements. Additional working views let you focus on specific signal chain elements. This simplifies work on complex signal chains.

What will you do?

In this lesson you will:

- Generate additional working views from selected signal chain elements.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You activated the Lesson_5 application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following steps:

- [Step 1: How to Generate Additional Working Views from Signal Chain Elements](#) on page 48

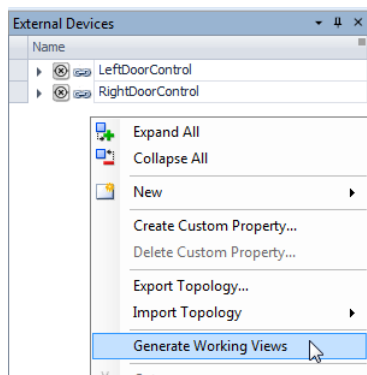
Step 1: How to Generate Additional Working Views from Signal Chain Elements

Objective

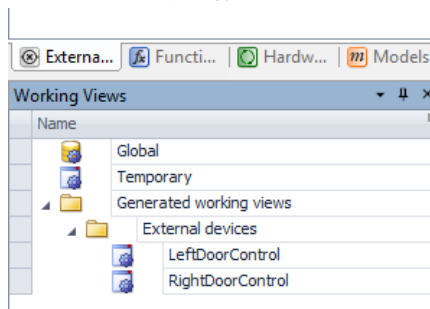
You can automatically create working views and working view groups according to the structure of used signal chain elements in topology browsers or the function library. Here, you will generate working views from the device topology you created in [Lesson 1: Specifying an External Device Interface](#) on page 17.

Method**To generate additional working views from signal chain elements**

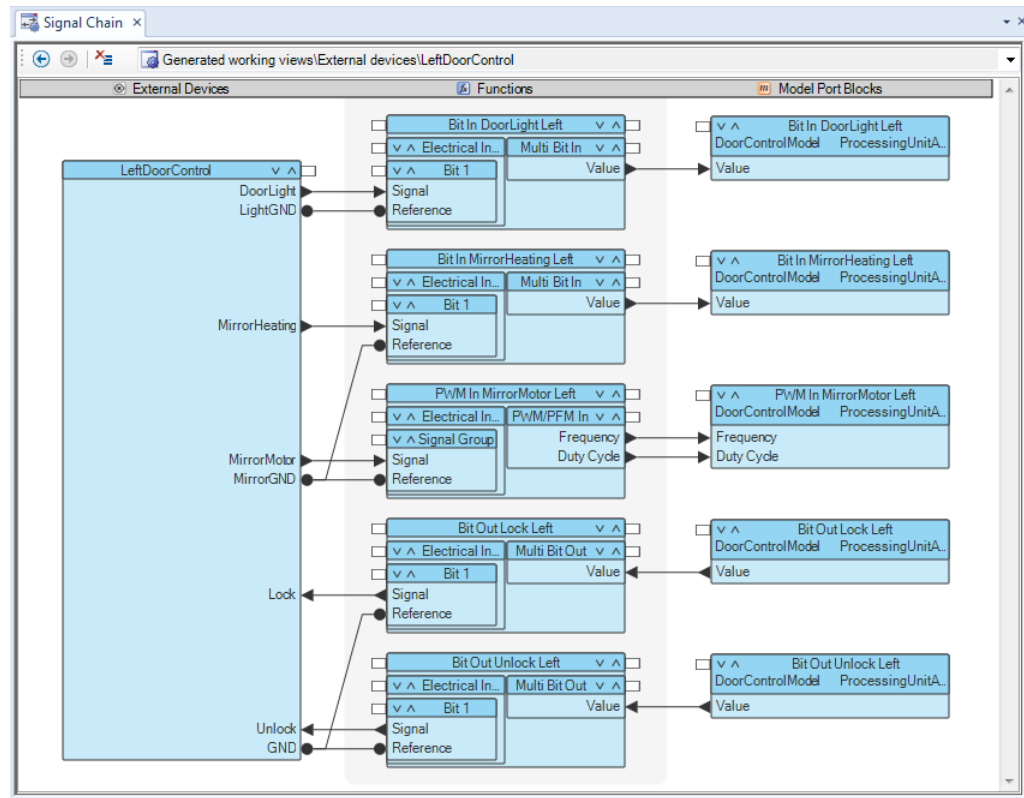
- 1 Switch to the Signal Chain view set if necessary.
- 2 In the External Device Browser, right-click a free area.
- 3 From the context menu, select **Generate Working Views**.



In the Working View Manager, working views are created for each device in the device topology:



- 4 Double-click a working view to open it in the Signal Chain Browser. The LeftDoorControl working view, for example, contains the LeftDoorControl device block and the signal chain elements connected to it:



Tip

You can also generate working views for all topologies at once or manually create user-defined working views and add selected signal chain elements to them. Refer to [Using Working Views](#) (*ConfigurationDesk Real-Time Implementation Guide*).

Result

You generated additional working views from signal chain elements.

Result of Lesson 5

Result

You learned how to generate additional working views from signal chain elements.

Now your application is equivalent to the Lesson_6 demo application.

Further information

For detailed descriptions and instructions on working views, refer to [Using Working Views \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(3dfb8d66e81160ad61421a3452093d1b_img.jpg\)](#)).

What's next

Now you can continue with [Lesson 6: Building the Real-Time Application](#) on page 51 or activate a different application to continue with the corresponding lesson. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Lesson 6: Building the Real-Time Application

Where to go from here

Information in this section

Overview of Lesson 6.....	51
How to Prepare the Build Process and Build the Real-Time Application.....	52
Result of Lesson 6.....	54

Overview of Lesson 6

Build process

Building real-time applications is a process which generates code as an executable file that can be run on the real-time hardware. You can manage the build process completely via ConfigurationDesk without switching to MATLAB/Simulink®.

What will you do?

In this lesson you will:

- Prepare the build process by applying build configuration settings.
- Build the executable real-time application.

Before you begin

Before you begin this lesson, one of the following preconditions must be met:

- You completed all the lessons of the tutorial up to this point.

or

- You activated the Lesson_6 application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

Steps

This lesson contains the following step:

- [How to Prepare the Build Process and Build the Real-Time Application](#) on page 52

How to Prepare the Build Process and Build the Real-Time Application

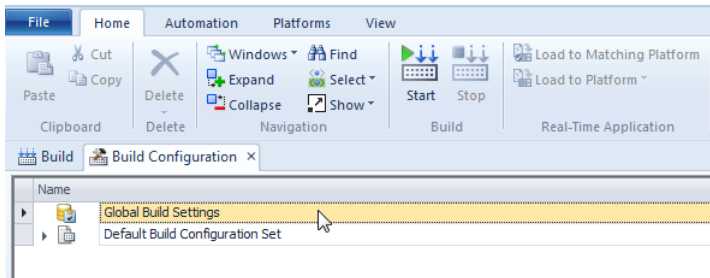
Objective

To execute a real-time application, you need to build it. You can then download it to a SCALEXIO system.

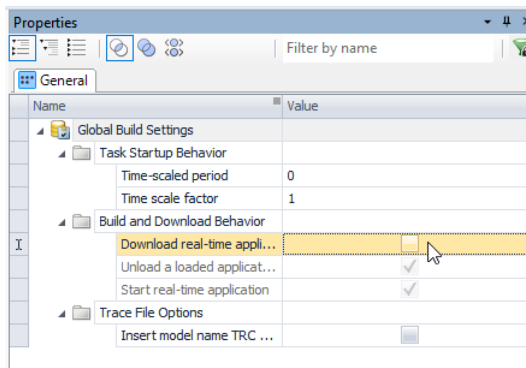
Before you build the real-time application, you can specify several build settings that influence the build process.

Method**To prepare the build process and build the real-time application**

- 1 Switch to the Build view set.
- 2 In the Build Configuration table, select Global Build Settings.



- 3 In the Properties Browser, clear the checkbox next to the Download real-time application after build property.

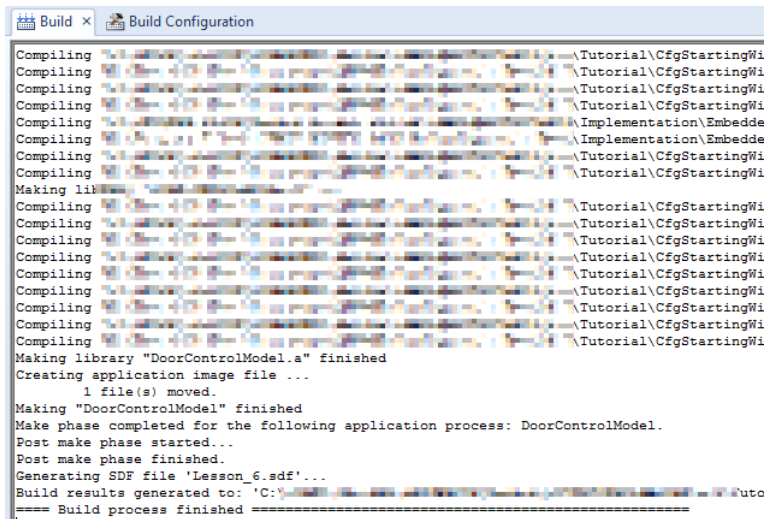


- 4 On the Home ribbon, click Build – Start.

The build process starts. If MATLAB is not running when you start the build process, it opens automatically for model code generation. The build process is aborted only for serious errors. Most problems that occur during the build

process are categorized as warnings. They are displayed in the Build Log Viewer. Messages regarding model code generation are displayed in the MATLAB Command Window.

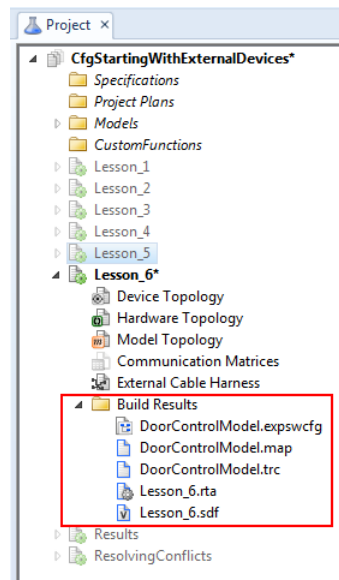
- 5 Wait until the message Build process finished is displayed in the Build Log Viewer.



Result

You prepared the build process and built the real-time application.

The build result files are displayed in the Project Manager.




In a real scenario, you could now download the real-time application to a hardware platform. For more information, refer to [Downloading and Executing Real-Time Applications \(ConfigurationDesk Real-Time Implementation Guide\)](#).

The Build Configuration table lets you specify various build settings. You can create different build configuration sets and assign application processes to

them. For more information, refer to [Building Real-Time Applications \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(d0a1791f26d167e866e44ebbf83efebe_img.jpg\)](#)).

Result of Lesson 6

Result	<p>You prepared the build process and built the executable real-time application.</p> <p>Now your application is equivalent to the Results demo application.</p>
Further information	<p>For detailed descriptions and instructions on build settings and building real-time applications, refer to Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide ).</p>
What's next	<p>You finished the last lesson of the tutorial. To repeat a previous lesson or complete a lesson that you skipped, activate the corresponding application. For instructions, refer to How to Activate an Application in the CfgStartingWithExternalDevices Demo Project on page 16.</p> <p>You can also activate the Resolving_Conflicts application to complete the optional lesson Optional Lesson: Resolving Conflicts on page 55.</p> <p>For a summary of your working results, refer to Summary on page 61.</p>

Optional Lesson: Resolving Conflicts

Where to go from here

Information in this section

Overview of the Resolving Conflicts Lesson.....	55
Step 1: How to Show and Resolve Conflicts.....	56
Result of Resolving Conflicts.....	60

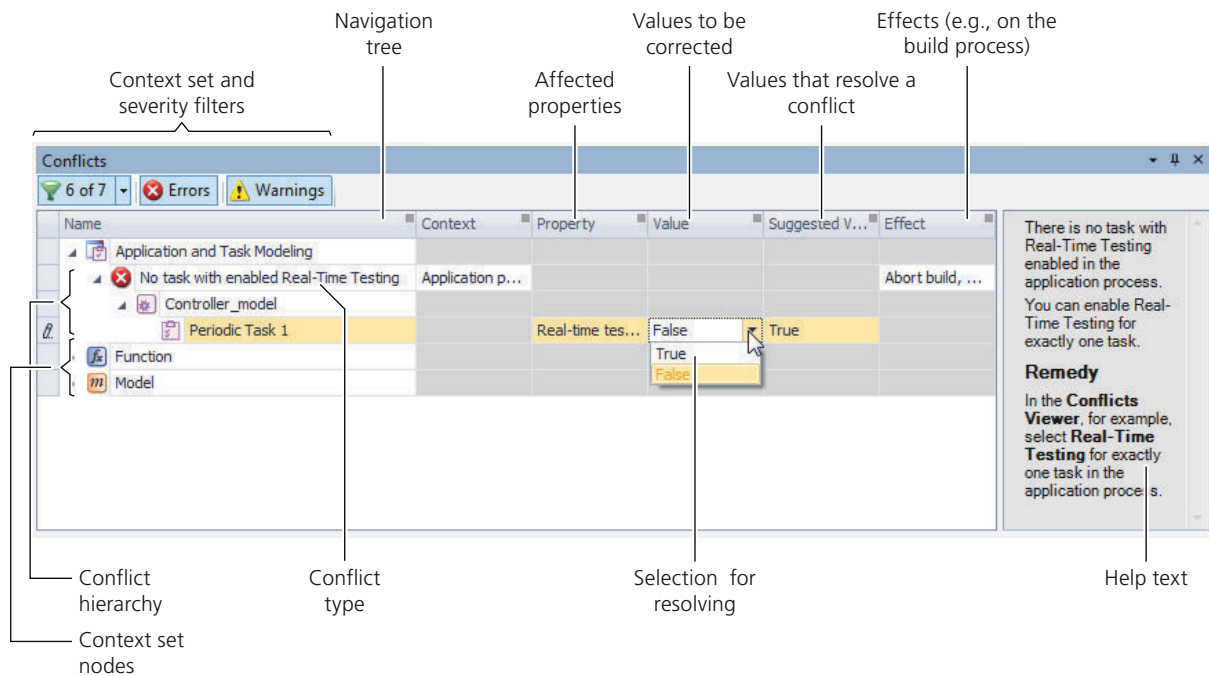
Overview of the Resolving Conflicts Lesson

Conflicts

Tip

Unlike in the other lessons of the tutorial, you cannot continue with your working results from previous lessons. Refer to [Before you begin](#) on page 56.

ConfigurationDesk allows for flexible configuration without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings. ConfigurationDesk automatically detects conflicts and provides the Conflicts Viewer to display and help resolve them. Before you build a real-time application, you have to resolve at least the most severe conflicts to get proper build results.



What will you do?

In this lesson you will:

- Analyze conflicts in the Conflicts Viewer and resolve them.

Before you begin

Before you begin this lesson, the following precondition must be met:

- You activated the ResolvingConflicts application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16. The application is similar to the results of lesson 6, except that a number of conflicts were added to different areas of the application.

Steps

This lesson contains the following step:

- [Step 1: How to Show and Resolve Conflicts](#) on page 56

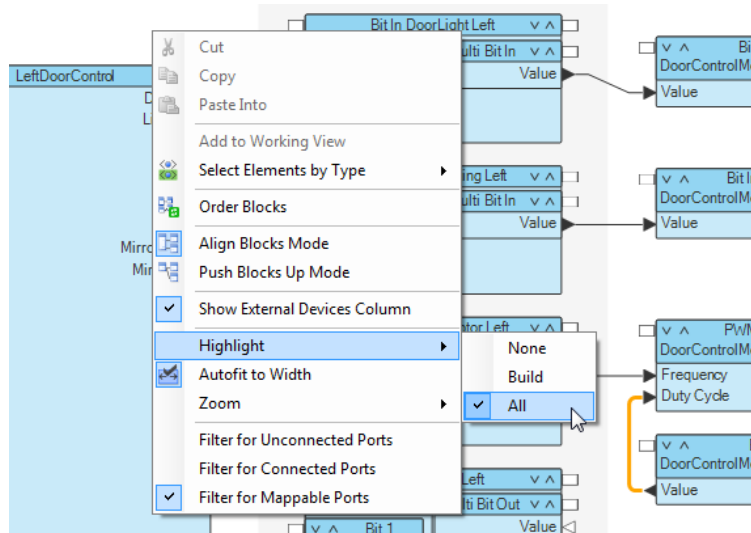
Step 1: How to Show and Resolve Conflicts

Objective

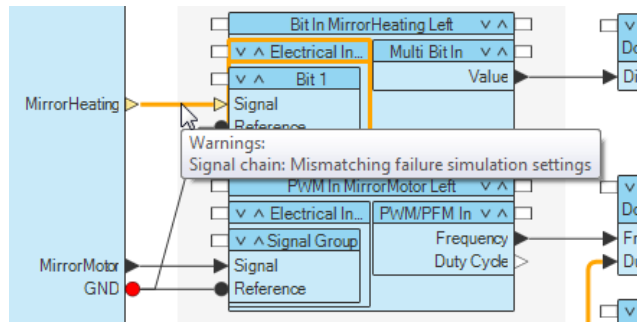
To indicate configuration problems, ConfigurationDesk generates conflicts. You can use the Conflicts Viewer to show and resolve the conflicts.

Preconditions

Make sure that the highlighting of all conflicting elements in the signal chain is active:



Signal chain elements involved in a conflict are highlighted and colored according to the severity of the conflict. If you point the mouse to a signal chain element that is highlighted, a tooltip is displayed.

**Method****To show and resolve conflicts**

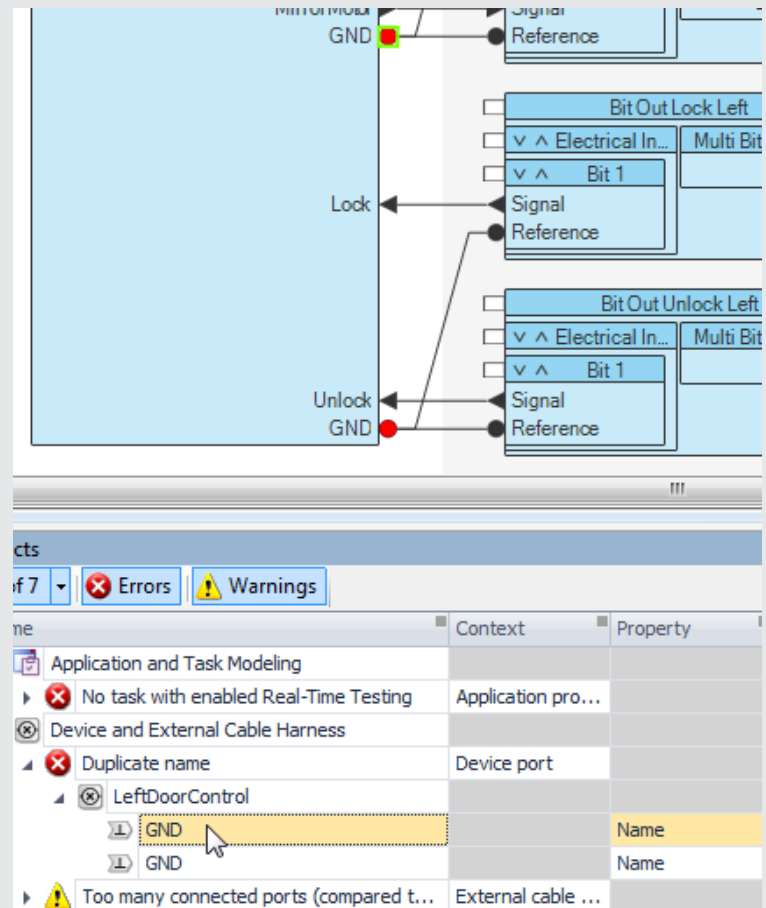
- 1 Open the Conflicts Viewer if necessary. It shows a number of conflicts.

Conflicts						
6 of 7						
Errors Warnings						
Name	Context	Property	Value	Suggested Values	Effect	
Application and Task Modeling						
No task with enabled Real-Time Testing	Application pro...				Abort build, Ab...	
Device and External Cable Harness						
Duplicate name	Device port				Abort XLSX exp...	
Too many connected ports (compared to...	External cable ...					
Missing wiring connections	External cable ...					
Mismatching failure simulation settings	Signal chain				Warning during...	
Function						
Unsupported maximum current for centr...	HW resource a...				Warning during...	
Missing channel assignment	Function block				Generate defa...	
Model						
Invalid connection between data port bl...	Model port map...				Generate no code	

- 2 Expand the Device and External Cable Harness – Duplicate name conflict in the navigation tree.

Tip

The elements causing the conflict are shown, two device ports in this case. Selecting one of the ports in the Conflicts Viewer causes it to be selected (green frame) in open working views. Both ports are colored red.



- 3 In the Value column, enter the name MirrorGND for the first of the two ports.
The Duplicate name conflict is removed.
- 4 Resolve the other conflicts with the help of the remedies described in the table below:

Conflict Type	Description/Remedy
Mismatching failure simulation settings	<p>There are two conflicts where the failure simulation settings of a signal port and a mapped device port are incompatible.</p> <p>The affected mappings are colored orange.</p> <p>In the Value column of the Conflicts Viewer, expand the conflicts and select:</p> <ul style="list-style-type: none"> Allowed for the Short to GND property of the MirrorHeating device port.

Conflict Type	Description/Remedy
	<ul style="list-style-type: none"> Not allowed for the Short to signal measurement channel property of the Signal signal port connected to the Unlock device port. <p>Note that you could also resolve the conflicts by changing the failure simulation settings in the Properties Browser.</p>
Missing wiring connections / Too many connected ports (compared to device mapping)	<p>The Missing wiring connections and Too many connected ports (compared to device mapping) conflict is due to a mapping between a device and a signal port that is missing either in the signal chain or in the calculated external cable harness.</p> <p>Expand the conflicts' hierarchies to display the device and signal ports that are involved. Selecting them will cause them to be selected in the Signal Chain Browser, too. There, you will also see highlighted mapping line for the Missing wiring connections conflicts.</p> <p>To resolve the conflict:</p> <ol style="list-style-type: none"> In the Global working view, select the highlighted mapping line. Press Del to delete the mapping line from the signal chain. Map the Lock port of the RightDoorControl device to the Signal port of the Bit Out Lock Right function block. Note that during mapping, the only port connection which resolves the Missing wiring connection conflict is marked green.
Missing channel assignment	<p>A channel assignment is missing for an electrical interface with an assigned channel set.</p> <p>The Conflicts Viewer shows you that the affected function block is the PWM In MirrorMotor Right block. Also, the Signal Group with the missing channel assignment is highlighted in the Signal Chain Browser (orange frame).</p> <p>In the Conflicts Viewer, expand the conflict and select Channel 15 for the channel request PWM/PFM In, I2 in the Value column.</p> <p>Note that in the value list, some other channels are highlighted in orange or red. This means that selecting them will cause other channel assignment conflicts.</p> <p>Also, note that resolving the conflict causes a new Missing wiring connections conflict to appear.</p>
Missing wiring connections	<p>This Missing wiring connections conflict is caused by the channel assignment you have made to resolve the Missing channel assignment conflict. The channel assignment is not part of the calculated wiring information.</p> <p>To resolve this conflict, you must recalculate the wiring information. Note that in a real use case this means that you must modify or replace an existing physical external cable harness.</p> <p>In the Project Manager, right-click the External Cable Harness and select Calculate.</p>
Invalid connection between data port blocks of same model	<p>Two model port blocks of DoorControlModel are mapped. Mappings between model port blocks of the same model are not allowed.</p> <p>In the Conflicts Viewer, expand the conflict and select the conflict source. The invalid mapping line is highlighted in the Signal Chain Browser.</p> <p>To resolve the conflict:</p> <ol style="list-style-type: none"> In the Global working view, delete the highlighted mapping. Map the Duty Cycle port of the PWM In MirrorMotor Left model port block to the Duty Cycle function port of the PWM In MirrorMotor Left function block. Map the Value port of the Bit Out Lock Left model port block to the Value function port of the Bit Out Lock Left function block.
No task with enabled Real-Time Testing	<p>Real-Time testing must be enabled for exactly one task.</p> <p>In the Conflicts Viewer, expand the conflict and set the Real-time testing property to True in the Value column.</p> <p>Usually, the Real-time testing property is configured in the Task Configuration table of the Tasks view set.</p>

Result

You resolved all conflicts.

Tip

The Conflicts Viewer provides various filters that help you focus on specific conflicts. This is particularly useful for large ConfigurationDesk applications with many conflicts. For more information, refer to [Details on Filtering Conflicts \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#)).

Result of Resolving Conflicts

Result

You resolved all conflicts.

Now your application is equivalent to the Lesson_6 demo application.

Further information

For detailed descriptions and instructions regarding conflicts, refer to [Resolving Conflicts \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\)](#)).

What's next

You finished the optional lesson on conflicts. To repeat a previous lesson or complete a lesson that you skipped, activate the corresponding application. For instructions, refer to [How to Activate an Application in the CfgStartingWithExternalDevices Demo Project](#) on page 16.

For a summary of your working results, refer to [Summary](#) on page 61.

Summary

Your Working Results

What you learned

You learned:

- How to use the basic features of ConfigurationDesk.
- How to create and configure the external device interface.
- How to create a hardware topology.
- How to implement I/O functionality via function blocks.
- How to specify the interface to the behavior model.
- How to use working views to focus on specific signal chain elements.
- How to prepare the build process and build an executable real-time application.
- How to show and resolve conflicts (only if you did the optional lesson Resolving Conflicts).

ConfigurationDesk Glossary

Introduction	The glossary briefly explains the most important expressions and naming conventions used in the ConfigurationDesk documentation.
--------------	--

Where to go from here

Information in this section

A.....	64
B.....	64
C.....	67
D.....	70
E.....	71
F.....	73
G.....	74
H.....	74
I.....	75
L.....	76
M.....	77
N.....	80
O.....	80
P.....	80
R.....	82
S.....	83
T.....	84
U.....	85

V.....	86
W.....	86
X.....	87

A

Application There are two types of applications in ConfigurationDesk:

- A part of a ConfigurationDesk project: [ConfigurationDesk application](#).
- An application that can be executed on dSPACE real-time hardware: [real-time application](#).

Application process A component of a [processing unit application](#). An application process contains one or more [tasks](#).



Application process component A component of an [application process](#). The following application process components are available in the **Components** subfolder of an application process:

- [Behavior models](#) that are assigned to the application process, including their predefined [tasks](#), [runnable functions](#), and [events](#).
- [Function blocks](#) that are assigned to the application process.

AutomationDesk A dSPACE software product for creating and managing any kind of automation tasks. Within the dSPACE tool chain, it is mainly used for automating tests on dSPACE hardware.

AUTOSAR system description file An AUTOSAR XML (ARXML) file that describes a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. The described network communication usually consists of more than one bus system (e.g., CAN, LIN, FlexRay).

B

Basic PDU A general term used in the documentation to address all the PDUs the Bus Manager supports, except for [container IPDUs](#), [multiplexed IPDUs](#), and [secured IPDUs](#). Basic PDUs are represented by the  or  symbol in

tables and browsers. The Bus Manager provides the same functionalities for all basic PDUs, such as [ISignal IPDUs](#) or NMPDUs.

Behavior model A model that contains the control algorithm for a controller (function prototyping system) or the algorithm of the controlled system (hardware-in-the-loop system). It does not contain I/O functionality nor access to the hardware. Behavior models can be modeled, for example, in MATLAB/Simulink by using Simulink Blocksets and Toolboxes from the MathWorks®.

You can add Simulink behavior models to a ConfigurationDesk application. You can also add code container files containing a behavior model such as [Functional Mock-up Units](#), or [Simulink implementation containers](#) to a ConfigurationDesk application.

Bidirectional signal port A [signal port](#) that is independent of a data direction or current flow. This port is used, for example, to implement bus communication.

BSC file A [bus simulation container](#) file that is generated with the [Bus Manager](#) and contains the configured bus communication of one [application process](#).

Build Configuration table A pane that lets you create build configuration sets and configure build settings, for example, build options, or the build and download behavior.

Build Log Viewer A pane that displays messages and warnings during the [build process](#).

Build process A process that generates an executable real-time application based on your [ConfigurationDesk application](#) that can be run on a [SCALEXIO system](#) or MicroAutoBox III system. The build process can be controlled and configured via the [Build Log Viewer](#). If the build process is successfully finished, the build result files ([build results](#)) are added to the ConfigurationDesk application.

Build results The files that are created during the [build process](#). Build results are named after the [ConfigurationDesk application](#) and the [application process](#) from which they originate. You can access the build results in the [Project Manager](#).

Bus access The representation of a run-time [communication cluster](#). By assigning one or more [bus access requests](#) to a bus access, you specify which communication clusters form one run-time communication cluster.

In ConfigurationDesk, you can use a bus [function block](#) (CAN, LIN) to implement a bus access. The [hardware resource assignment](#) of the bus function block specifies the bus channel that is used for the bus communication.

Bus access request The representation of a request regarding the [bus access](#). There are two sources for bus access requests:

- At least one element of a [communication cluster](#) is assigned to the Simulated ECUs, Inspection, or Manipulation part of a [bus configuration](#). The related bus access requests contain the requirements for the bus channels that are to be used for the cluster's bus communication.

- A frame gateway is added to the Gateways part of a bus configuration. Each frame gateway provides two bus access requests that are required to specify the bus channels for exchanging bus communication.

Bus access requests are automatically included in [BSC files](#). To build a [real-time application](#), each bus access request must be assigned to a bus access.

Bus Access Requests table A pane that lets you access [bus access requests](#) of a [ConfigurationDesk application](#) and assign them to [bus accesses](#).

Bus configuration A Bus Manager element that implements bus communication in a [ConfigurationDesk application](#) and lets you configure it for simulation, inspection, and/or manipulation purposes. The required bus communication elements must be specified in a [communication matrix](#) and assigned to the bus configuration. Additionally, a bus configuration lets you specify gateways for exchanging bus communication between [communication clusters](#). A bus configuration can be accessed via specific tables and its related Bus Configuration [function block](#).

Bus Configuration Function Ports table A pane that lets you access and configure function ports of [bus configurations](#).

Bus Configurations table A pane that lets you access and configure [bus configurations](#) of a [ConfigurationDesk application](#).

Bus Inspection Features table A pane that lets you access and configure bus configuration features of a [ConfigurationDesk application](#) for inspection purposes.

Bus Manager

- Bus Manager in ConfigurationDesk
A ConfigurationDesk component that lets you configure bus communication and implement it in [real-time applications](#) or generate [bus simulation containers](#).
- Bus Manager (stand-alone)
A dSPACE software product based on ConfigurationDesk that lets you configure bus communication and generate bus simulation containers.

Bus Manipulation Features table A pane that lets you access and configure bus configuration features of a [ConfigurationDesk application](#) for manipulation purposes.

Bus simulation container A container that contains bus communication configured with the [Bus Manager](#). Bus simulation container ([BSC](#)) files can be used in the [VEOS Player](#) and in ConfigurationDesk. In the VEOS Player, they let you implement the bus communication in an [offline simulation application](#).

In ConfigurationDesk, they let you implement the bus communication in a [real-time application](#) independently from the Bus Manager.

Bus Simulation Features table A pane that lets you access and configure bus configuration features of a [ConfigurationDesk application](#) for simulation purposes.

Buses Browser A pane that lets you display and manage the [communication matrices](#) of a [ConfigurationDesk application](#). For example, you can access communication matrix elements and assign them to bus configurations. This pane is available only if you work with the [Bus Manager](#).

C

Cable harness A bundle of cables that provides the connection between the I/O connectors of the real-time hardware and the [external devices](#), such as the ECUs to be tested. In ConfigurationDesk, it is represented by an [external cable harness](#) component.

CAFX file A ConfigurationDesk application fragment file that contains [signal chain](#) elements that were exported from a user-defined [working view](#) or the Temporary working view of a [ConfigurationDesk application](#). This includes the elements' configuration and the [mapping lines](#) between them.

CDL file A [ConfigurationDesk application](#) file that contains links to all the documents related to an application.

Channel multiplication A feature that allows you to enhance the max. current or max. voltage of a single hardware channel by combining several channels. ConfigurationDesk uses a user-defined value to calculate the number of hardware channels needed. Depending on the [function block type](#), channel multiplication is provided either for current enhancement (two or more channels are connected in parallel) or for voltage enhancement (two or more channels are connected in series).

Channel request A channel assignment required by a [function block](#). ConfigurationDesk determines the type(s) and number of channels required for a function block according to the assigned [channel set](#), the function block features, the block configuration and the required physical ranges. ConfigurationDesk provides a set of suitable and available [hardware resources](#) for each channel request. This set is produced according to the [hardware topology](#) added to the active [ConfigurationDesk application](#). You have to assign each channel request to a specific channel of the hardware topology.

Channel set A number of channels of the same [channel type](#) located on the same I/O board or I/O unit. Channels in a channel set can be combined, for example, to provide a signal with [channel multiplication](#).

Channel type A term to indicate all the [hardware resources](#) (channels) in the hardware system that provide exactly the same characteristics. Examples for

channel type names: Flexible In 1, Digital Out 3, Analog In 1. An I/O board in a hardware system can have [channel sets](#) of several channel types. Channel sets of one channel type can be available on different I/O boards.

Cluster [Communication cluster](#).

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>

Communication cluster A communication network of [network nodes](#) that are connected to the same physical channels and share the same bus protocol and address range.

Communication matrix A file that defines the communication of a bus network. It can describe the bus communication of one [communication cluster](#) or a bus network consisting of different bus systems and clusters. Files of various file formats can be used as a communication matrix: For example, [AUTOSAR system description files](#), [DBC files](#), [LDF files](#), and [FIBEX files](#).

Communication package A package that bundles Data Inport blocks which are connected to Data Outport blocks. Hence, it also bundles the signals that are received by these blocks. If Data Inport blocks are executed within the same [task](#) and belong to the same [communication package](#), their data inports are read simultaneously. If Data Outport blocks that are connected to the Data Inport blocks are executed in the same task, their output signals are sent simultaneously in one data package. Thus, communication packages guarantee simultaneous signal updates within a running task (data snapshot).

Configuration port A port that lets you create the [signal chain](#) for the bus communication implemented in a Simulink behavior model. The following configuration ports are available:

- The configuration port of a [Configuration Port block](#).
- The Configuration port of a CAN, LIN, or FlexRay function block.

To create the signal chain for bus communication, the configuration port of a Configuration Port block must be mapped to the Configuration port of a CAN, LIN, or FlexRay function block.

Configuration Port block A [model port block](#) that is created in ConfigurationDesk during model analysis for each of the following blocks found in the Simulink behavior model:

- RTICANMM ControllerSetup block
- RTILINMM ControllerSetup block
- FLEXRAYCONFIG UPDATE block

Configuration Port blocks are also created for bus simulation containers. A Configuration Port block provides a [configuration port](#) that must be mapped

to the Configuration port of a CAN, LIN, or FlexRay function block to create the signal chain for bus communication.

ConfigurationDesk application A part of a ConfigurationDesk [project](#) that represents a specific implementation. You can work with only one application at a time, and that application must be activated.

An application can contain:

- [Device topology](#)
- [Hardware topology](#)
- [Model topology](#)
- [Communication matrices](#)
- [External cable harness](#)
- [Build results](#) (after a successful [build process](#) has finished)

You can also add folders with application-specific files to an application.

ConfigurationDesk model interface The part of the [model interface](#) that is available in ConfigurationDesk. This specific term is used to explicitly distinguish between the model interface in ConfigurationDesk and the model interface in Simulink.

Conflict A result of conflicting configuration settings that is displayed in the [Conflicts Viewer](#). ConfigurationDesk allows flexible configuration without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings. ConfigurationDesk automatically detects conflicts and provides the Conflicts Viewer to display and help resolve them. Before you build a [real-time application](#), you have to resolve at least the most severe conflicts (e.g., errors that abort the [build process](#)) to get proper [build results](#).

Conflicts Viewer A pane that displays the configuration [conflicts](#) that exist in the active [ConfigurationDesk application](#). You can resolve most of the conflicts directly in the Conflicts Viewer.

Container IPDU A term according to AUTOSAR. An [IPDU](#) that contains one or more other IPDUs (i.e., contained IPDUs). When a container IPDU is mapped to a [frame](#), all its contained IPDUs are included in that frame as well.

ControlDesk A dSPACE software product for managing, instrumenting and executing experiments for ECU development. ControlDesk also supports calibration, measurement and diagnostics access to ECUs via standardized protocols such as CCP, XCP, and ODX.

CTLGZ file A ZIP file that contains a V-ECU implementation. CTLGZ files are exported by [TargetLink](#) or [SystemDesk](#). You can add a V-ECU implementation based on a CTLGZ file to the [model topology](#) just like adding a Simulink model based on an [SLX file](#).

Cycle time restriction A value of a [runnable function](#) that indicates the sample time the runnable function requires to achieve correct results. The cycle time restriction is indicated by the Period property of the runnable function in the [Properties Browser](#).

D

Data import A port that supplies data from ConfigurationDesk's function outputs to the behavior model.

In a multimodel application, data imports also can be used to provide data from a data output associated to another behavior model ([model communication](#)).

Data output A port that supplies data from behavior model signals to ConfigurationDesk's function inputs.

In a multimodel application, data outputs also can be used to supply data to a data input associated to another behavior model ([model communication](#)).

DBC file A Data Base Container file that describes CAN or LIN bus systems. Because the DBC file format was primarily developed to describe CAN networks, it does not support definitions of LIN masters and schedules.

Device block A graphical representation of devices from the [device topology](#). It can be mapped to [function blocks](#) via [device ports](#).

Device connector A structural element that lets you group [device pins](#) in a hierarchy in the [External Device Connectors table](#) to represent the structure of the real connector of your [external device](#).

Device pin A representation of a connector pin of your [external device](#). [Device ports](#) are assigned to device pins. ConfigurationDesk can use the device pin assignment together with the [hardware resource assignment](#) and the device port mapping to calculate the [external cable harness](#).

Device port An element of a [device topology](#) that represents the signal of an [external device](#) in ConfigurationDesk.

Device port group A structural element of a [device topology](#) that can contain [device ports](#) and other device port groups.

Device topology A component of a [ConfigurationDesk application](#) that represents [external devices](#) in ConfigurationDesk. You can create a device topology from scratch or easily extend an existing device topology. You can also merge device topologies to extend one. To edit or create device topologies independently of ConfigurationDesk, you can export and import [DTFX](#) and [XLSX](#) files.

Documents folder A standard folder for user-specific documents.
 %USERPROFILE%\Documents\dSPACE\<ProductName>\
 <VersionNumber>

DSA file A dSPACE archive file that contains a [ConfigurationDesk application](#) and all the files belonging to it as one unit. It can later be imported to another ConfigurationDesk [project](#).

dSPACE Help The dSPACE online help that contains all the relevant user documentation for dSPACE products. Via the F1 key or the Help button in the dSPACE software you get context-sensitive help on the currently active context.

dSPACE Log A collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

DTFX file A [device topology](#) export file that contains information on the interface to the [external devices](#), such as the ECUs to be tested. The information includes details of the available [device ports](#), their characteristics, and the assigned pins.

E

ECHX file An [external cable harness](#) file that contains the wiring information for the external cable harness. The external cable harness is the connection between the I/O connectors of the real-time hardware and the devices to be tested, for example, ECUs.

ECU Abbreviation of *electronic control unit*.

An embedded computer system that consists of at least one CPU and associated peripherals. An ECU contains communication controllers and communication connectors, and usually communicates with other ECUs of a bus network. An ECU can be member of multiple bus systems and [communication clusters](#).

ECU application An application that is executed on an [ECU](#). In [ECU interfacing](#) scenarios, parts of the ECU application can be accessed (e.g., by a [real-time application](#)) for development and testing purposes.

ECU function A function of an [ECU application](#) that is executed on the [ECU](#). In [ECU interfacing](#) scenarios, an ECU function can be accessed by functions that are part of a [real-time application](#), for example.

ECU Interface Manager A dSPACE software product for preparing [ECU applications](#) for [ECU interfacing](#). The ECU Interface Manager can generate ECU interface container ([EIC](#)) files to be used in ConfigurationDesk.

ECU interfacing A generic term for methods and tools to read and/or write individual [ECU functions](#) and variables of an [ECU application](#). In ECU interfacing scenarios, you can access ECU functions and variables for development and testing purposes while the ECU application is executed on the [ECU](#). For example, you can perform ECU interfacing with [SCALEXIO systems](#) or MicroAutoBox III systems to access individual ECU functions by a [real-time application](#).

EIC file An ECU interface container file that is generated with the [ECU Interface Manager](#) and describes an [ECU application](#) that is configured for [ECU interfacing](#). You can import EIC files to ConfigurationDesk to perform ECU interfacing with [SCALEXIO systems](#) or MicroAutoBox III systems.

Electrical interface unit A segment of a [function block](#) that provides the interface to the [external devices](#) and to the real-time hardware (via [hardware](#)

[resource assignment](#)). Each electrical interface unit of a function block usually needs a [channel set](#) to be assigned to it.

Event A component of a [ConfigurationDesk application](#) that triggers the execution of a [task](#). The following event types are available:

- [Timer event](#)
- [I/O event](#)
- [Software event](#)

Event port An element of a [function block](#). The event port can be mapped to a [runnable function port](#) for modeling an asynchronous task.

Executable application The generic term for [real-time applications](#) and [offline simulation applications](#). In ConfigurationDesk, an executable application is always a real-time application since ConfigurationDesk does not support offline simulation applications.

Executable application component A component of an [executable application](#). The following components can be part of an executable application:

- Imported [behavior models](#) including predefined [tasks](#), [runnable functions](#), and [events](#). You can assign these behavior models to [application processes](#) via drag & drop or by selecting the Assign Model command from the context menu of the relevant application process.
- Function blocks added to your ConfigurationDesk application including associated [I/O events](#). Function blocks are assigned to application processes via their model port mapping.

Executable Application table A pane that lets you model [executable applications](#) (i.e., [real-time applications](#)) and the [tasks](#) used in them.

EXPSWCFG file An experiment software configuration file that contains configuration data for automotive fieldbus communication. It is created during the [build process](#) and contains the data in XML format.

External cable harness A component of a [ConfigurationDesk application](#) that contains the wiring information for the external cable harness (also known as [cable harness](#)). It contains only the logical connections and no additional information such as cable length, cable diameters, dimensions or the arrangement of connection points, etc. It can be calculated by ConfigurationDesk or imported from a file so that you can use an existing cable harness and do not have to build a new one. The wiring information can be exported to an [ECHX file](#) or [XLSX file](#).

External device A device that is connected to the dSPACE hardware, such as an ECU or external load. The external [device topology](#) is the basis for using external devices in the [signal chain](#) of a [ConfigurationDesk application](#).

External Device Browser A pane that lets you display and manage the [device topology](#) of your active [ConfigurationDesk application](#).

External Device Configuration table A pane that lets you access and configure the most important properties of device topology elements via table.

External Device Connectors table A pane that lets you specify the representation of the physical connectors of your [external device](#) including the device pin assignment.

F

FIBEX file An XML file according the ASAM MCD-2 NET standard (also known as Field Bus Exchange Format) defined by ASAM. The file can describe more than one bus system (e.g., CAN, LIN, FlexRay). It is used for data exchange between different tools that work with message-oriented bus communication.

Find Results Viewer A pane that displays the results of searches you performed via the Find command.

FMU file A [Functional Mock-up Unit](#) file that describes and implements the functionality of a model. It is an archive file with the file name extension FMU. The FMU file contains:

- The functionality defined as a set of C functions provided either in source or in binary form.
- The model description file (`modelDescription.xml`) with the description of the interface data.
- Additional resources needed for simulation.

You can add an FMU file to the [model topology](#) just like adding a Simulink model based on an [SLX file](#).

Frame A piece of information of a bus communication. It contains an arbitrary number of non-overlapping [PDUs](#) and the data length code (DLC). CAN frames and LIN frames can contain only one PDU. To exchange a frame via bus channels, a [frame triggering](#) is needed.

Frame triggering An instance of a [frame](#) that is exchanged via a bus channel. It includes transmission information of the frame (e.g., timings, ID, sender, receiver). The requirements regarding the frame triggerings depend on the bus system (CAN, LIN, FlexRay).

Function block A graphical representation in the [signal chain](#) that is instantiated from a [function block type](#). A function block provides the I/O functionality and the connection to the real-time hardware. It serves as a container for functions, for [electrical interface units](#) and their [logical signals](#). The function block's ports ([function ports](#) and/or [signal ports](#)), provide the interfaces to the neighboring blocks in the signal chain.

Function block type A software plug-in that provides a specific I/O functionality. Every function block type has unique features which are different from other function block types.

To use a function block type in your [ConfigurationDesk application](#), you have to create an instance of it. This instance is called a [function block](#). Instances of function block types can be used multiple times in a ConfigurationDesk

application. The types and their instantiated function blocks are displayed in the [function library](#) of the [Function Browser](#).

Function Browser A pane that displays the [function library](#) in a hierarchical tree structure. [Function block types](#) are grouped in function classes. Instantiated [function blocks](#) are added below the corresponding function block type.

Function inport A [function port](#) that inputs the values from the [behavior model](#) to the [function block](#) to be processed by the function.

Function library A collection of [function block types](#) that allows access to the I/O functionality in ConfigurationDesk. The I/O functionality is based on function block types. The function library provides a structured tree view on the available function block types. It is displayed in the [Function Browser](#).

Function outputport A [function port](#) that outputs the value of a function to be used in the [behavior model](#).

Function port An element of a [function block](#) that provides the interface to the [behavior model](#) via [model port blocks](#).

Functional Mock-up Unit An archive file that describes and implements the functionality of a model based on the Functional Mock-up Interface (FMI) standard.

G

Global working view The default [working view](#) that always contains all [signal chain](#) elements.

H

Hardware resource A hardware element (normally a channel on an I/O board or I/O unit) which is required to execute a [function block](#). A hardware resource can be localized unambiguously in a hardware system. Every hardware resource has specific characteristics. A function block therefore needs a hardware resource that matches the requirements of its functionality. This means that not every function block can be executed on every hardware resource. There could be limitations on a function block's features and/or the physical ranges.

Hardware resource assignment An action that assigns the [electrical interface unit](#) of a [function block](#) to one or more [hardware resources](#). Function blocks can be assigned to any hardware resource which is suitable for

the functionality and available in the [hardware topology](#) of your [ConfigurationDesk application](#).

Hardware Resource Browser A pane that lets you display and manage all the hardware components of the [hardware topology](#) that is contained in your active [ConfigurationDesk application](#) in a hierarchical structure.

Hardware topology A component of a [ConfigurationDesk application](#) that contains information on a specific hardware system which can be used with ConfigurationDesk. It provides information on the components of the system, such as [channel type](#) and slot numbers. It can be scanned automatically from a registered [platform](#), created in ConfigurationDesk's [Hardware Resource Browser](#) from scratch, or imported from an [HTFX file](#).

HTFX file A file containing the [hardware topology](#) after an explicit export. It provides information on the components of the system and also on the channel properties, such as board and [channel types](#) and slot numbers.

I/O event An asynchronous [event](#) triggered by I/O functions. You can use I/O events to trigger tasks in your application process asynchronously. You can assign the events to the tasks via drag & drop, via the Properties Browser if you have selected a task, or via the Assign Event command from the context menu of the relevant task.

Interface model A temporary Simulink model that contains blocks from the Model Interface Blockset. ConfigurationDesk initiates the creation of an interface model in Simulink. You can copy the blocks with their identities from the interface model and paste them into an existing Simulink behavior model.

Interpreter A pane that lets you run Python scripts and execute line-based commands.

Inverse model port block A model port block that has the same configuration (same name, same port groups, and port names) but the inverse data direction as the original model port block from which it was created.

IOCNET Abbreviation of I/O carrier network.

A dSPACE proprietary protocol for internal communication in a [SCALEXIO system](#) between the real-time processors and I/O units. The IOCNET lets you connect more than 100 I/O nodes and place the parts of your SCALEXIO system long distances apart.

IPDU Abbreviation of interaction layer protocol data unit.

A term according to AUTOSAR. An IPDU contains the communication data that is routed from the interaction layer to a lower communication layer and vice

versa. An IPDU can be implemented, for example, as an [ISignal IPDU](#), [multiplexed IPDU](#), or [container IPDU](#).

ISignal A term according to AUTOSAR. A signal of the interaction layer that contains communication data as a coded signal value. To transmit the communication data on a bus, ISignals are instantiated and included in [ISignal IPDUs](#).

ISignal IPDU A term according to AUTOSAR. An [IPDU](#) whose communication data is arranged in [ISignals](#). ISignal IPDUs allow the exchange of ISignals between different [network nodes](#).

L

LDF file A LIN description file that describes networks of the LIN bus system according to the LIN standard.

LIN master A member of a LIN [communication cluster](#) that is responsible for the timing of LIN bus communication. A LIN master provides one LIN master task and one LIN slave task. The LIN master task transmits frame headers on the bus, and provides [LIN schedule tables](#) and LIN collision resolver tables. The LIN slave task transmits frame responses on the bus. A LIN cluster must contain exactly one LIN master.

LIN schedule table A table defined for a [LIN master](#) that contains the transmission sequence of frame headers on a LIN bus. For each LIN master, several LIN schedule tables can be defined.

LIN slave A member of a LIN [communication cluster](#) that provides only a LIN slave task. The LIN slave task transmits frame responses on the bus when they are triggered by a frame header. The frame header is sent by a [LIN master](#). A LIN cluster can contain several LIN slaves.

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>
```

Logical signal An element of a [function block](#) that combines all the [signal ports](#) which belong together to provide the functionality of the signal. Each logical signal causes one or more [channel requests](#). Channel requests are available after you have assigned a [channel set](#) to the logical signal.

Logical signal chain A term that describes the logical path of a signal between an [external device](#) and the [behavior model](#). The main elements of the logical signal chain are represented by different graphical blocks ([device blocks](#), [function blocks](#) and [model port blocks](#)). Every block has ports to provide the mapping to neighboring blocks.

In the documentation, usually the short form 'signal chain' is used instead.

M

MAP file A file that maps symbolic names to physical addresses.

Mapping line A graphical representation of a connection between two ports in the [signal chain](#). You can draw mapping lines in a [working view](#).

MCD file A model communication description file that is used to implement a [multimodel application](#). It lets you add several [behavior models](#) that were separated from an overall model to the [model topology](#).

The MCD file contains information on the separated models and information on the connections between them. The file is generated with the [Model Separation Setup Block](#) in MATLAB/Simulink. The block resides in the Model Interface Blockset (dsmpplib) from dSPACE.

MDL file A Simulink model file that contains the [behavior model](#). You can add an MDL file to your [ConfigurationDesk application](#).

As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

Message Viewer A pane that displays a history of all error and warning messages that occur during work with ConfigurationDesk.

Model analysis A process that analyzes the model to determine the interface of a [behavior model](#). You can select one of the following commands:

- **Analyze Simulink Model (Model Interface Only)**
Analyzes the interface of a behavior model. The [model topology](#) of your active [ConfigurationDesk application](#) is updated with the properties of the analyzed behavior model.
- **Analyze Simulink Model (Including Task Information)**
Analyzes the [model interface](#) and the elements of the behavior model that are relevant for the task configuration. The task configuration in ConfigurationDesk is then updated accordingly.

Model Browser A pane that lets you display and access the [model topology](#) of an active [ConfigurationDesk application](#). The Model Browser provides access to all the [model port blocks](#) available in the [behavior models](#) which are linked to a ConfigurationDesk application. The model elements are displayed in a hierarchy, starting with the model roots. Below the model root, all the subsystems containing model port blocks are displayed as well as the associated model port blocks.

Model communication The exchange of signal data between the models within a [multimodel application](#). To set up model communication, you must use a [mapping line](#) to connect a data output (sending model) to a data input

(receiving model). The best way to set up model communication is using the [Model Communication Browser](#).

Model Communication Browser A pane that lets you open and browse [working views](#) like the [Signal Chain Browser](#), but shows only the Data Output and Data Input blocks and the [mapping lines](#) between them.

Model Communication Package table A pane that lets you create and configure model communication packages which are used for [model communication](#) in [multimodel applications](#).

Model implementation An implementation of a [behavior model](#). It can consist of source code files, precompiled objects or libraries, variable description files and a description of the model's interface. Specific model implementation types are, for example, [model implementation containers](#), such as [Functional Mock-up Units](#) or [Simulink implementation containers](#).

Model implementation container A file archive that contains a [model implementation](#). Examples are FMUs, SIC files, and VECU files.

Model interface An interface that connects ConfigurationDesk with a [behavior model](#). This interface is part of the signal chain and is implemented via model port blocks. The model port blocks in ConfigurationDesk can provide the interface to:

- Model port blocks (from the [Model Interface Package for Simulink](#)) in a Simulink behavior model. In this case, the model interface is also called ConfigurationDesk model interface to distinguish it from the Simulink model interface available in the Simulink behavior model.
- Different types of model implementations based on code container files, e.g., Simulink implementation containers, Functional Mock-up Units, and V-ECU implementations.

Model Interface Package for Simulink A dSPACE software product that lets you specify the interface of a [behavior model](#) that you can directly use in ConfigurationDesk. You can also create a code container file ([SIC file](#)) that contains the model code of a Simulink [behavior model](#). The SIC file can be used in ConfigurationDesk and [VEOS Player](#).

Model port An element of a [model port block](#). Model ports provide the interface to the [function ports](#) and to other model ports (in [multimodel applications](#)).

These are the types of model ports:

- Data input
- Data output
- Runnable function port
- Configuration port

Model port block A graphical representation of the [ConfigurationDesk model interface](#) in the [signal chain](#). Model port blocks contain model ports that can be mapped to function blocks to provide the data flow between the function blocks in ConfigurationDesk and the [behavior model](#). The model ports can also be mapped to the model ports of other model port blocks with

data inports or data outports to set up [model communication](#). Model port blocks are available in different types and can provide different port types:

- Data port blocks with [data inports](#) and [data outports](#)
- [Runnable Function blocks](#) with [runnable function ports](#)
- [Configuration Port blocks](#) with [configuration ports](#). Configuration Port blocks are created during model analysis for each of the following blocks found in the Simulink behavior model:
 - RTICANMM ControllerSetup block
 - RTILINMM ControllerSetup block
 - FLEXRAYCONFIG UPDATE block

Configuration Port blocks are also created for bus simulation containers.

Model Separation Setup Block A block that is contained in the [Model Interface Package for Simulink](#). It is used to separate individual models from an overall model in MATLAB/Simulink. Additionally, model separation generates a model communication description file ([MCD file](#)) that contains information on the separated models and their connections. You can use this MCD file in ConfigurationDesk.

Model topology A component of a [ConfigurationDesk application](#) that contains information on the subsystems and the associated model port blocks of all the behavior models that have been added to a ConfigurationDesk application.

Model-Function Mapping Browser A pane that lets you create and update [signal chains](#) for Simulink [behavior models](#). It directly connects them to I/O functionality in ConfigurationDesk.

MTFX file A file containing a [model topology](#) when explicitly exported. The file contains information on the interface to the [behavior model](#), such as the implemented [model port blocks](#) including their subsystems and where they are used in the model.

Multicore real-time application A [real-time application](#) that is executed on several cores of one [PU](#) of the real-time hardware.

Multimodel application A [real-time application](#) that executes several [behavior models](#) in parallel on dSPACE real-time hardware ([SCALEXIO](#) or [MicroAutoBox III](#)).

Multiplexed IPDU A term according to AUTOSAR. An [IPDU](#) that consists of one dynamic part, a selector field, and one optional static part. Multiplexing is used to transport varying [ISignal IPDUs](#) via the same bytes of a multiplexed IPDU.

- The dynamic part is one [ISignal IPDU](#) that is selected for transmission at run time. Several [ISignal IPDUs](#) can be specified as dynamic part alternatives. One of these alternatives is selected for transmission.

- The selector field value indicates which ISignal IPDU is transmitted in the dynamic part during run time. For each selector field value, there is one corresponding ISignal IPDU of the dynamic part alternatives. The selector field value is evaluated by the receiver of the multiplexed IPDU.
- The static part is one ISignal IPDU that is always transmitted.

Multi-PU application Abbreviation of multi-processing-unit application. A multi-PU application is a [real-time application](#) that is partitioned into several [processing unit applications](#). Each processing unit application is executed on a separate [PU](#) of the real-time hardware. The processing units are connected via [IOCNET](#) and can be accessed from the same host PC.

N

Navigation bar An element of ConfigurationDesk's user interface that lets you switch between [view sets](#).

Network node A term that describes the bus communication of an [ECU](#) for only one [communication cluster](#).

O

Offline simulation A purely PC-based simulation scenario without a connection to a physical system, i.e., neither simulator hardware nor ECU hardware prototypes are needed. Offline simulations are independent from real time and can run on [VEOS](#).

Offline simulation application An application that runs on [VEOS](#) to perform [offline simulation](#). An offline simulation application can be built with the [VEOS Player](#) and the resulting [OSA file](#) can be downloaded to VEOS.

OSA file An [offline simulation application](#) file that is built with the [VEOS Player](#) and can be downloaded to [VEOS](#) to perform [offline simulation](#).

P

Parent port A port that you can use to map multiple [function ports](#) and [model ports](#). All child ports with the same name are mapped. ConfigurationDesk enforces the mapping rules and allows only [mapping lines](#) which agree with them.

PDU Abbreviation of protocol data unit.

A term according to AUTOSAR. A PDU transports data (e.g., control information or communication data) via one or multiple network layers according to the AUTOSAR layered architecture. Depending on the involved layers and the function of a PDU, various PDU types can be distinguished, e.g., [Signal IPDUs](#), [multiplexed IPDUs](#), and NMPDUs.

Physical signal chain A term that describes the electrical wiring of [external devices](#) (ECU and loads) to the I/O boards of the real-time hardware. The physical signal chain includes the [external cable harness](#), the pinouts of the connectors and the internal cable harness.

Pins and External Wiring table A pane that lets you access the external wiring information

Platform A dSPACE real-time hardware system that can be registered and displayed in the [Platform Manager](#).

Platform Manager A pane that lets you handle registered hardware [platforms](#). You can download, start, and stop [real-time applications](#) via the Platform Manager. You can also update the firmware of your [SCALEXIO system](#) or MicroAutoBox III system.

Preconfigured application process An [application process](#) that was created via the Create preconfigured application process command. If you use the command, ConfigurationDesk creates new [tasks](#) for each [runnable function](#) provided by the model which is not assigned to a predefined task. ConfigurationDesk assigns the corresponding runnable function and (for periodic tasks) [timer events](#) to the new tasks. The tasks are preconfigured (e.g., DAQ raster name, period).

Processing Resource Assignment table A pane that lets you configure and inspect the processing resources in an [executable application](#). This table is useful especially for [multi-processing-unit applications](#).

Processing unit application A component of an [executable application](#). A processing unit application contains one or more [application processes](#).

Project A container for [ConfigurationDesk applications](#) and all project-specific documents. You must define a project or open an existing one to work with ConfigurationDesk. Projects are stored in a [project root folder](#).

Project Manager A pane that provides access to ConfigurationDesk [projects](#) and [applications](#) and all the files they contain.

Project root folder A folder on your file system to which ConfigurationDesk saves all project-relevant data, such as the [applications](#) and documents of a [project](#). Several projects can use the same project root folder. ConfigurationDesk uses the [Documents folder](#) as the default project root

folder. You can specify further project root folders. Each can be made the default project root folder.

Properties Browser A pane that lets you access the properties of selected elements.

PU Abbreviation of processing unit.

A hardware assembly that consists of a motherboard or a dSPACE processor board, possibly additional interface hardware for connecting I/O boards, and an enclosure, i.e., a SCALEXIO Real-Time PC.

R

Real-time application An application that can be executed in real time on dSPACE real-time hardware. The real-time application is the result of a [build process](#). It can be downloaded to real-time hardware via an [RTA file](#). There are different types of real-time applications:

- [Single-core real-time application](#).
- [Multicore real-time application](#).
- [Multi-PU application](#).

Restbus simulation A simulation method to test real [ECUs](#) by connecting them to a simulator that simulates the other ECUs in the [communication clusters](#).

RTA file A [real-time application](#) file. An RTA file is an executable object file for processor boards. It is created during the [build process](#). After the build process it can be downloaded to the real-time hardware.

Runnable function A function that is called by a [task](#) to compute results. A model implementation provides a runnable function for its base rate task. This runnable function can be executed in a task that is triggered by a timer event. In addition, a Simulink behavior model provides a runnable function for each Hardware-Triggered Runnable Function block contained in the Simulink behavior model. This runnable function is suitable for being executed in an asynchronous task.

Runnable Function block A type of [model port block](#). A Runnable Function block provides a [runnable function port](#) that can be mapped to an [event port](#) of a [function block](#) for modeling an asynchronous task.

Runnable function port An element of a [Runnable Function block](#). The runnable function port can be mapped to an [event port](#) of a [function block](#) for modeling an asynchronous task.

RX Communication data that is received by a bus member.

SCALEXIO system A dSPACE hardware-in-the-loop (HIL) system consisting of one or more real-time industry PCs ([PUs](#)), I/O boards, and I/O units. They communicate with each other via the [IOCNET](#). The system simulates the environment to test an [ECU](#). It provides the sensor signals for the ECU, measures the signals of the ECU, and provides the power (battery voltage) for the ECU and a bus interface for [restbus simulation](#).

SDF file A system description file that contains information on the CPU name(s), the corresponding object file(s) to be downloaded and the corresponding variable description file(s). It is created during the build process.

Secured IPDU A term according to AUTOSAR. An [IPDU](#) that secures the payload of another PDU (i.e., authentic IPDU) for secure onboard communication (SecOC). The secured IPDU contains the authentication information that is used to secure the authentic IPDU's payload. If the secured IPDU is configured as a cryptographic IPDU, the secured IPDU and the authentic IPDU are mapped to different [frames](#). If the secured IPDU is not configured as a cryptographic IPDU, the authentic IPDU is directly included in the secured IPDU.

SIC file A [Simulink implementation container](#) file that contains the model code of a Simulink [behavior model](#). The SIC file can be used in ConfigurationDesk and in VEOS Player.

Signal chain A term used in the documentation as a short form for [logical signal chain](#). Do not confuse it with the [physical signal chain](#).

Signal Chain Browser A pane that lets you open and browse [working views](#) such as the [Global working view](#) or user-defined working views.

Signal inport A [signal port](#) that represents an electrical connection point of a [function block](#) which provides signal measurement (= input) functionality.

Signal outport A [signal port](#) that represents an electrical connection point of a [function block](#) which provides signal generation (= output) functionality.

Signal port An element of a [function block](#) that provides the interface to [external devices](#) (e.g., [ECUs](#)) via [device blocks](#). It represents an electrical connection point of a function block.

Signal reference port A [signal port](#) that represents a connection point for the reference potential of [inports](#), [outports](#) and [bidirectional ports](#). For example: With differential signals, this is a reference signal, with single-ended signals, it is the ground signal (GND).

Simulink implementation container A container that contains the model code of a Simulink behavior model. A Simulink implementation container is generated from a Simulink behavior model by using the [Model Interface Package](#)

[for Simulink](#). The file name extension of a Simulink implementation container is SIC.

Simulink model interface The part of the [model interface](#) that is available in the connected Simulink behavior model.

Single-core real-time application An [executable application](#) that is executed on only one core of the real-time hardware.

Single-PU system Abbreviation of single-processing-unit system.
A system consisting of exactly one [PU](#) and the directly connected I/O units and I/O routers.

SLX file A Simulink model file that contains the [behavior model](#). You can add an SLX file to your [ConfigurationDesk application](#).
As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

Software event An event that is activated from within a [task](#) to trigger another subordinate task. Consider the following example: A multi-tasking Simulink behavior model has a base rate task with a sample time = 1 ms and a periodic task with a sample time = 4 ms. In this case, the periodic task is triggered on every fourth execution of the base rate task via a software event. Software events are available in ConfigurationDesk after [model analysis](#).

Source Code Editor A Python editor that lets you open and edit Python scripts that you open from or create in a ConfigurationDesk project in a window in the [working area](#). You cannot run a Python script in a Source Code Editor window. To run a Python script you can use the Run Script command in the [Interpreter](#) or on the Automation ribbon or the Run context menu command in the [Project Manager](#).

Structured data port A hierarchically structured port of a Data Input block or a Data Output block. Each structured data port consists of more structured and/or unstructured data ports. The structured data ports can consist of signals with different data types (single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, Boolean).

SystemDesk A dSPACE software product for development of distributed automotive electrics/electronics systems according to the AUTOSAR approach. SystemDesk is able to provide a V-ECU implementation container (as a [VECU file](#)) to be used in ConfigurationDesk.

T

Table A type of pane that offers access to a specific subset of elements and properties of the active [ConfigurationDesk application](#) in rows and columns.

TargetLink A dSPACE software product for production code generation. It lets you generate highly efficient C code for microcontrollers in electronic control

units (ECUs). It also helps you implement control systems that have been modeled graphically in a Simulink/Stateflow diagram on a production ECU. TargetLink is able to provide a V-ECU implementation container (as a [VECU file](#)) or a Simulink implementation container (SIC file) to be used in ConfigurationDesk.

Task A piece of code whose execution is controlled by a real-time operating system (RTOS). A task is usually triggered by an [event](#), and executes one or more [runnable functions](#). In a ConfigurationDesk application, there are predefined tasks that are provided by [executable application components](#). In addition, you can create user-defined tasks that are triggered, for example, by I/O events. Regardless of the type of task, you can configure the tasks by specifying the priority, the DAQ raster name, etc.

Task Configuration table A pane that lets you configure the [tasks](#) of an [executable application](#).

Temporary working view A [working view](#) that can be used for drafting a [signal chain](#) segment, like a notepad.

Timer event A periodic [event](#) with a sample rate and an optional offset.

Topology A hierarchy that serves as a repository for creating a [signal chain](#). All the elements of a topology can be used in the signal chain, but not every element needs to be used. You can export each topology from and import it to a [ConfigurationDesk application](#). Therefore a topology can be used in several applications.

A ConfigurationDesk application can contain the following topologies:

- [Device topology](#)
- [Hardware topology](#)
- [Model topology](#)

TRC file A variable description file that contains all variables (signals and parameters) which can be accessed via the experiment software. It is created during the [build process](#).

TX Communication data that is transmitted by a bus member.

U

User function An external function or program that is added to the Automation – User Functions ribbon group for quick and easy access during work with ConfigurationDesk.

V

VECU file A ZIP file that contains a V-ECU implementation. A VECU file can contain data packages for different platforms. VECU files are exported by [TargetLink](#) or [SystemDesk](#). You can add a V-ECU implementation based on a VECU file to the [model topology](#) in the same way as adding a Simulink model based on an [SLX file](#).

VEOS The dSPACE software product for performing [offline simulation](#). VEOS is a PC-based simulation platform which allows offline simulation independently from real time.

VEOS Player A software running on the host PC for building [offline simulation applications](#). Offline simulation applications can be downloaded to [VEOS](#) to perform [offline simulation](#). ConfigurationDesk lets you generate a [bus simulation container](#) (BSC file) via the [Bus Manager](#). You can then import the BSC file into the VEOS Player.

View set A specific arrangement of some of ConfigurationDesk's panes. You can switch between view sets by using the [navigation bar](#). ConfigurationDesk provides preconfigured view sets for specific purposes. You can customize existing view sets and create user-defined view sets.

VSET file A file that contains all view sets and their settings from the current ConfigurationDesk installation. A VSET file can be exported and imported via the View Sets page of the Customize dialog.

W

Working area The central area of ConfigurationDesk's user interface.

Working view A view of the [signal chain](#) elements (blocks, ports, mappings, etc.) used in the active [ConfigurationDesk application](#). A working view can be opened in the [Signal Chain Browser](#) or the [Model Communication Browser](#). ConfigurationDesk provides two default working views: The [Global working view](#) and the [Temporary working view](#). In the [Working View Manager](#), you can also create user-defined working views that let you focus on specific signal chain segments according to your requirements.

Working View Manager A pane that lets you manage the [working views](#) of the active [ConfigurationDesk application](#). You can use the Working View Manager for creating, renaming, and deleting working views, and also to open a working view in the [Signal Chain Browser](#) or the [Model Communication Browser](#).

XLSX file A Microsoft Excel™ file format that is used for the following purposes:

- Creating or configuring a [device topology](#) outside of ConfigurationDesk.
- Exporting the wiring information for the [external cable harness](#).
- Exporting the configuration data of the currently active [ConfigurationDesk application](#) for documentation purposes.

B

- behavior model 43
 - add 43
- build process 52

C

- channel 35
- channel set 35
- Common Program Data folder 6, 68
- conflicts 56
 - resolve 56
- Conflicts Viewer 56

D

- demo project 13
 - activate application 16
 - open 15
 - overview 13
 - ways to use 9
- device blocks 22
- device connector 23
- device pins 23
- device port mapping 36
- Documents folder 6, 70

E

- external cable harness 38
- external device interface
 - configure 19
- external device topology
 - copy elements 21
 - create 18

F

- function blocks 32
 - add to signal chain 32
 - configure 33

H

- hardware resource assignment 35
- hardware topology
 - create 28

L

- lesson overview 10
- Local Program Data folder 6, 76

M

- model port blocks 42

R

- real-time application 52

U

- use scenario 14

W

- wiring information 38
- working views
 - create 48

